caesar Documentation

Release latest

Simone Riggi

Dec 21, 2021

INSTALLATION

CAESAR is a C++ software tool for automated source finding in astronomical maps. It is distributed for research use only under the GNU General Public License v3.0.

CHAPTER

ONE

INSTALLATION

1.1 Prerequisites

Install the project mandatory dependencies:

- **ROOT** [https://root.cern.ch/], to be built with FITSIO, PyROOT, RInterface options enabled. Make sure that the FindROOT.cmake is present in $ROOTSYS/etc/cmake directory after installation.
- **OpenCV** [http://opencv.org/]
- **log4cxx** [https://logging.apache.org/log4cxx/]
- **boost** [http://www.boost.org/]
- **cfitsio** [https://heasarc.gsfc.nasa.gov/fitsio/fitsio.html], to be built with multithread support (e.g. give –enable-reentrant option in configure)
- **protobuf** [https://github.com/google/protobuf]
- **jsoncpp** [https://github.com/open-source-parsers/jsoncpp]
- **cmake** (>=2.8) [https://cmake.org]

Optional dependencies are:

- **MPICH** [https://www.mpich.org/] or **OpenMPI** [https://www.open-mpi.org/], needed when the build option ENABLE_MPI=ON
- **OpenMP** [http://www.openmp.org/], needed when the build option BUILD_WITH_OPENMP=ON
- **R** [https://www.r-project.org/] and additional packages: RInside, Rcpp, rrcovHD, truncnorm, FNN, akima. Needed when the build option ENABLE_R=ON
- **GoogleTest** [https://github.com/google/googletest], needed for unit testing when the build option ENABLE_TEST=ON
- **Doxygen** [www.doxygen.org/], needed to generate the API documentation
- **Sphinx** [http://www.sphinx-doc.org] & **Breathe** [https://pypi.org/project/breathe], needed to generate the Sphinx API & wiki documentation

Dependencies for the provided scripts are:

- **python** (>=2.7) [https://www.python.org/] and these additional modules: pyfits, astropy, scipy, getopt, argparse, collections, matplotlib, pylab
- **CASA** [https://casa.nrao.edu/]

Make sure you have set the following environment variables to the external library installation dirs:

- ROOTSYS

- OPENCV_DIR
- BOOST_ROOT
- LOG4CXX_ROOT
- JSONCPP_ROOT
- R_DIR (optional)

Add also the following paths to the PKG_CONFIG_PATH environment var:

- $LOG4CXX_ROOT/lib/pkgconfig
- $JSONCPP_ROOT/lib/pkgconfig

CAESAR depends also on the wcstools and linterp libraries which are already provided in the external/ directory.

Warning: The provided wcslib was slightly modified with respect to the original release to avoid naming conflicts with the R package due to some #define macros used in WCS.

cmake should find all needed include dirs and libraries used to build the project.

1.2 Build and install

To build and install the project:

- Clone this repository into your local $SOURCE_DIR:

```
git clone https://github.com/SKA-INAF/caesar.git $SOURCE_DIR
```

- Create the build and install directories: $BUILD_DIR, $INSTALL_DIR
- Configure build. In the build directory type:

```
cmake -DCMAKE_INSTALL_PREFIX=$INSTALL_DIR -DBUILD_WITH_OPENMP=ON -DENABLE_MPI=ON \
    -DBUILD_APPS=ON -DENABLE_TEST=ON $SOURCE_DIR
```

- Building project. In the build directory type:

```
make
```

- Install project in the $INSTALL_DIR directory specified in the configuration step above. in the build directory type:

```
make install
```

In the installation directory you should find the following directories:

- *include*: Directory in which project source headers are installed.
- *bin*: Directory in which app executables are installed.
- *lib*: Directory in which Caesar shared library is installed.
- *share*: Directory in which documentation and other misc files are installed.
- *script*: Directory in which python & shell scripts (simulation, submission, etc) are installed.
- *macros*: Directory in which util macros (e.g. ROOT macros, atc) are installed.
- *data*: Directory in which test data images are installed.

1.3 Documentation generation

To generate and install the API documentation you must have Doxygen installed. Enter the build directory and type:

```
make doc
```

To generate and install the Sphinx wiki documentation you must have sphinx+breathe installed. Enter the build directory and type:

```
make doc-sphinx
```

The generated documentation will be installed in the $INSTALL_DIR/doc directory.

1.4 Running unit tests

To build the unit tests you must have Google Test installed and the ENABLE_TEST option set to ON when building Caesar. To run the test:

```
make test
```

or alternatively run the script *runUnitTests* installed in the Caesar installation dir.

CHAPTER

TWO

DISTRIBUTION

To support users having problems to install CAESAR from source in their system we provide Docker and Singularity container recipe files and pre-built images with all software installed.

2.1 Docker base container

We assume here that Docker is available in your system. If not, install it following the instruction at https://docs.docker.com.

Recipe files to build a base Docker image with all CAESAR dependencies installed can be downloaded at: https://github.com/SKA-INAF/caesar-base-docker. For example the following command builds a Docker image on your system with name *caesar/base* (tag=latest) from *Dockerfile.xenial*:

```
docker build -t 'caesar/base:latest' -f 'Dockerfile.xenial' .
```

A pre-built Docker image (size approximately ~16 GB), ready to be used, can be download from *Docker Hub* using the following command:

```
docker pull sriggi/caesar-base
```

The image is built over a Ubuntu 16:04 (xenial) base OS image. Software dependencies are installed under the `/opt/Software` directory.

A Docker container with CAESAR installed will be provided in the future.

2.2 Singularity container

We assume here that Singularity is available in your system. If not, install it from https://singularity.lbl.gov.

Recipe files to build the CAESAR singularity image can be downloaded at: https://github.com/SKA-INAF/caesar-singularity. Singularity image is built using the base Docker image described above. For example the following command builds a production container (e.g. non writable) on your system with name *caesar.simg* from a recipe file *Singularity.xenial*:

```
sudo singularity build caesar.simg Singularity.xenial
```

A pre-built container image (size approximately ~2 GB), ready to be used, can be download from *Singularity Hub* using the following command:

```
singularity pull --name caesar.simg shub://SKA-INAF/caesar-singularity:xenial
```

To enter the container type:

```
singularity shell caesar.simg
```

CAESAR is installed in the `/opt/Software/CAESAR/trunk/` container directory. Software dependencies are installed under the `/opt/Software` directory.

CAESAR applications are available for batch processing as Singularity apps. To list available apps type:

```
singularity apps caesar.simg
```

To run one of these applications (say with name *APP_NAME* and with input arguments *APP_ARGS*) type:

```
singularity run --app APP_NAME caesar.simg APP_ARGS
```

For example source finding (see Tutorial section) can be run inside container with:

```
singularity run --app sfinder caesar.simg --config=[FILE]
```

CHAPTER

THREE

HOW TO USE CAESAR

I managed to build the project with success or I got a copy of CAESAR container. Now what should I do? CAESAR library is built upon the ROOT framework and can be used in Linux OS (in principle usable also in MacOS but not tested) in different ways for simple tasks as well to build more complex applications:

- Interactively from the ROOT CLI
- In C++ macros run by the ROOT interpreter
- In C++ standalone programs
- Interactively from the python/ipython CLI
- In python scripts run by the python interpreter

If you have worked with ROOT you will find these guidelines familiar.

3.1 Using CAESAR from the ROOT CLI

CAESAR generates dictionaries for all relevant classes, including the *Image* class. This enables to use the CAESAR classes in the ROOT CLI. For this you should put these lines in your `.rootlogon.C` file.

```
gSystem->Load("[path to CAESAR library libCaesar.so]");
gROOT->ProcessLine(".include [path to CAESAR include dir]");
gInterpreter->AddIncludePath("[path to CAESAR include dir]");
using namespace Caesar;
```

Additionally add CAESAR library (`libCaesar.so`) and path to dictionary file (`CaesarDict_rdict.pcm`) to your LD_LIBRARY_PATH environment variable. For example for a Bash shell:

```
export LD_LIBRARY= $LD_LIBRARY_PATH:[path-to-CAESAR-lib]:[path-to-CAESAR-dict-file]
```

The *.rootlogon.C* is loaded every time the ROOT console is started from Linux prompt. To start ROOT CLI type `root` at the Linux shell prompt. At this point you should be ready to use CAESAR inside ROOT. The following example shows how to create a Caesar image from a FITS file (say it is named *map.fits*), compute statistics, background and noise maps:

```
Image* img= new Image
img->ReadFITS("map.fits")
img->ComputeStats(true)
ImgBkgData* bkgData= img->ComputeBkg(eMedianBkg,true,100,100,10,10)
```

This is useful for simple tasks or for drawing purposes. For more complex tasks, you should actually write macros or higher level classes/applications using CAESAR API, like it is briefly discussed below and done in the examples reported in the Tutorial section.

3.2 Using CAESAR in ROOT macros

ROOT macros are simply C/C++ code run by the ROOT interpreter. Once you have followed the configuration steps illustrated in the previous paragraph you should be able to run a macro using CAESAR objects in ROOT. Let's prepare a simple macro (file named `MyMacro.C`) below:

```
//== MyMacro.C ==
#include <Image.h> //needed when compiling the macro

void MyMacro()
{
  Image* img= new Image;//create a new empty image
  img->ReadFITS("map.fits");//fill image from fits
  img->ComputeStats(true);//compute standard & robust stats
  ImgBkgData* bkgData= img->ComputeBkg(eMedianBkg,true,100,100,10,10);//compute bkg data

  Image* rmsMap= bkgData->NoiseMap;//get compute rms map
  rmsMap->GetHisto2D("histo")->Draw("COLZ");//draw rms map as histo 2d

  // etc etc...
}
```

To execute this macro in ROOT you can do the following:

- Run from the Linux shell prompt:

```
root MyMacro.C
```

- Run from the ROOT prompt:

```
.x MyMacro.C
```

- Compile the macro and run from the ROOT prompt:

```
.L MyMacro.C+
MyMacro()
```

We refer the reader to the ROOT manual for more details on running macros, passing arguments to them, etc.

3.3 Using CAESAR in C++ applications

To use CAESAR library in your C++ application you just need to add the CAESAR headers (in the _include_ directory) in your compilation and link against CAESAR library (_libCaesar.so_). Let's prepare a simple C++ program (file named `MyApp.cc`:

```
#include <Image.h>
#include <TApplication.h>//if you need to draw in ROOT canvas
```

(continues on next page)

(continued from previous page)

```
int main(int argc, char **argv)
{
  //Needed if you want interactivity and draw in ROOT (not needed for batch applications)
  TApplication* app= new TApplication("Application", 0, 0);

  Image* img= new Image;//create a new empty image
  img->ReadFITS("map.fits");//fill image from fits
  img->ComputeStats(true);//compute standard & robust stats
  ImgBkgData* bkgData= img->ComputeBkg(eMedianBkg,true,100,100,10,10);//compute bkg data

  Image* rmsMap= bkgData->NoiseMap;//get compute rms map
  rmsMap->GetHisto2D("histo")->Draw("COLZ");//draw rms map as histo 2d

  //This will draw the image and suspend execution (remove in batch apps)
  app->Run();

  return 0;
}
```

Now compile and execute the program:

```
g++ -std=c++11 -g -o MyApp MyApp.cc \
    -I[path-to-CAESAR-headers] -I`root-config --incdir` \
    `pkg-config $OPENCV_DIR/lib/pkgconfig/opencv.pc --cflags` \
    -L[path-to-CAESAR-lib-dir] -lCaesar `root-config --libs` \
    `pkg-config $OPENCV_DIR/lib/pkgconfig/opencv.pc --libs`

./MyApp
```

3.4 Using CAESAR in python CLI/scripts

PyROOT interface enables using Caesar classes in python. For example:

```
from ROOT import gSystem
gSystem.Load('libCaesar')
from ROOT import Caesar

img= Caesar.Image()
img.ReadFITS('recmap.fits')
img.ComputeStats(True)
bkgData= img.ComputeBkg(Caesar.eMedianBkg,True,100,100,10,10)
```

python support is currently experimental and not fully tested.

3.5 Running CAESAR tasks

Long-running tasks and applications such as source finding should be run in batch mode directly on the operating system or inside a Singularity container. A number of applications are available in the *bin* installation directory. Task run can be customized via a configuration file passed as argument. For example to run source finding:

```
FindSourceMPI --config=config.cfg
```

For "container" run (assuming to have a caesar container image named *caesar.simg*) source finding can be run as:

```
singularity run --app sfinder caesar.simg --config=config.cfg
```

Submission shell scripts, available in the *scripts* installation directory, enable running tasks on batch systems (PBS, SLURM).

That's it! See the Tutorial section for more examples.

CHAPTER

FOUR

CONFIGURATION OPTIONS

Most of the example applications provided can be configured from command line arguments, as described in the Tutorial section. Some applications, like source finding, however, require a large set of configuration options, specified inside a configuration file, passed to the application as a command line argument `--config=[FILE]`.

In this section we report a list of the main configuration options defined in CAESAR to customize tasks. The full list of options defined is kept in `ConfigParser.cc` class. To print the full list of defined options use the *ConfigParser::PrintOptions()* method. For example from ROOT prompt type:

`Caesar::ConfigParser::Instance().PrintOptions()`

or from the python CLI:

```
from ROOT import gSystem
gSystem.Load('libCaesar')
from ROOT import Caesar

Caesar.ConfigParser.Instance().PrintOptions()
```

4.1 Input Options

These options enable control of input data to be given to CAESAR applications.

Option	Description	Default	Values
inputFile	Input image filename (.root/.fits)	""	
inputImage	Image name to be read in input ROOT file	""	
readTileImage	Read sub-image If false read the entire image	false	true false
tileMinX	Min image x pixel coordinate to be read Used only when readTileImage is true	0	
tileMaxX	Max image x pixel coordinate to be read Used only when readTileImage is true	0	
tileMinY	Min image y pixel coordinate to be read Used only when readTileImage is true	0	
tileMinY	Max image y pixel coordinate to be read Used only when readTileImage is true	0	

4.2 Output Options

These options enable control of information & data reported in output by CAESAR applications.

Option	Description	Default	Values
saveToFile	Save results & maps to output ROOT file	true	true false
saveConfig	Save config options to output ROOT file	true	true false
saveSources	Save sources to output ROOT file	true	true false
saveInputMap	Save input map to output ROOT file	false	true false
saveBkgMap	Save computed background map to output ROOT file	true	true false
saveNoiseMap	Save computed rms map to output ROOT file	true	true false
saveResidualMap	Save computed residual map to output ROOT file	true	true false
saveSignificanceMap	Save computed significance map to output ROOT file	true	true false
saveSaliencyMap	Save computed saliency map to output ROOT file	true	true false
saveSegmentedMap	Save computed segmented map to output ROOT file	true	true false
4.2. Output Options			**15**
outputFile	Name of ROOT file	*output.root*	

4.3 Run & Distributed Processing Options

These options enable control of application run (e.g. logging levels) and distributed processing (e.g. number of threads).

Option	Description	Default	Values
logLevel	Log level threshold	INFO	DEBUG INFO WARN ERROR FATAL
nThreads	Number of threads used if OPENMP is enabled. If set to -1 a number of threads equal to the available cores is used	-1	
splitInTiles	Split input image in tiles for parallel processing	false	true false
tileSizeX	Size of tile in pixels along X coordinate used for partition	1000	
tileSizeY	Size of tile in pixels along Y coordinate used for partition	1000	
useTileOverlap	Enable tile overlap in image partition for parallel processing	false	true false
tileStepSizeX	Tile overlap fraction along X coordinate to partition the input image for parallel processing (1=no overlap, 0.5=half overlap)	1	
tileStepSizeY	Tile overlap fraction along Y coordinate to partition the input image for parallel	1	

4.4 Stats & Background Compute Options

These options enable control of image background calculation. Background can be either computed globally or locally. Local background maps (bkg, rms) are obtained by interpolating background estimator values computed on a grid of sampling image rectangular boxes.

Option	Description	Default	Values
bkgEstimator	Stat estimator used to compute image background & noise image background & noise image background & noise	2	1=Mean/RMS 2=Median/MAD 3=BiWeight 4=Clipped Median/RMS
useParallelMedianAlgo	Use C++ parallel algorithm to compute median estimator	true	true false
useLocalBkg	Compute local background and noise maps and use them instead of global bkg info	true	true false
use2ndPassInLocalBkg	Use 2nd pass to refine local noise map	true	true false
skipOutliersInLocalBkg	Exclude pixels belonging to detected bright blobs when computing local background estimators. Blob find seed thr parameters are reported in source finding option table below	false	true false
boxSizeX	Size of sampling box along x coordinate for local bkg calculation in pixels. Size is instead assumed as multiple of beam size if useBeamInfoInBkg is	20	

4.5 Source Finding Options

These options enable control of source detection. This is performed using a flood-fill algorithm aggregating pixels around significant seeds if above a given merge threshold. Detected blobs form a collection of candidate sources.

Option	Description	Default	Values
searchCompactSources	Enable/disable search of compact sources	true	true false
minNPix	Minimum number of pixels to consider a blob as source candidate	5	
seedThr	Seed threshold in blob finding given as number of sigmas above background	5	
mergeThr	Merge/aggregation threshold in blob finding given as number of sigmas above background. Pixels above this threshold are added to the blob	2.6	
mergeBelowSeed	Add to blob only pixels above merge threshold but below seed threshold	false	true false
searchNegativeExcess	Search for holes (i.e. blobs with negative significance) along with "positive" blobs	false	true false
compactSourceSearchNIters	Number of iterations to be performed in compact source search. At each iteration the seed threshold is decreased by seedThrStep	1	

4.6 Nested Source Finding Options

These options enable control of nested source detection. Nested sources are blobs inside another mother blobs. Detection of nested blob uses a blob detection algorithm, based on the thresholding of a filter blob map (LoG or Gaus2D smoothed), which increases the computation time, particularly if blob search is done at multiple spatial scales. In presence of extended/diffuse object you can consider turning off this calculation. If however you have extended and bright object and you turn off nested source search you may see that compact/point-source located inside the extended one will be included in the mother and not fitted.

Option	Description	Default	Values
searchNestedSources	Enable/disable search of compact nested sources	true	true false
blobMaskMethod	Filter map used in nested blob finder to search blobs	2	1=gaus smoothed Lapl 2=multi-scale LoG
nestedBlobKernFactor	Filter kernel size factor par so that kern size= factor x sigma (sigma is the filter scale par in pixels)	6	
sourceToBeamAreaThrToSearchNested	Mother source area/beam thr to add nested sources. If npix<=thr*beamArea no nested sources are added to the mother source even if detected. If thr=0 nested sources are always added if searchNestedSources is enabled	10	
nestedBlobThrFactor	Threshold factor param used in blob filter map to create mask (thr=thrFactor*<img>).	0	
minNestedMotherDist	Minimum distance in pixels (in x or y) between nested and parent blob centroids below which nested source is skipped as most probably equal to the parent (avoid duplicates)	2	
maxMatchingPixFraction		0.5	

4.7 Source Selection Options

These options enable control of quality selection cuts applied to detected blobs to select good source candidates and tag point-source candidates (used later in source residual map and fitting stage). Options are also provided to select sources to be stored in the final catalog.

Option	Description	Default	Values
applySourceSelection	Enable/disable source selection	true	true false
useMinBoundingBoxCut	Apply minimum bounding box cut to detected blobs	false	true false
sourceMinBoundingBox	Minimum bounding box cut value in pixel. Blobs with minimum bounding box size below the threshold are tagged as bad	2	
useCircRatioCut	Apply cut on blob circular ratio param to detected blobs	false	true false
psCircRatioThr	Circular ratio cut value. in pixel. Blobs with circ ratio above this threshold passed the point-like cut (1=circle, 0=line)	0.4	0 1
useElongCut	Apply cut on blob elongation param to detected blobs	false	true false
psElongThr	Elongation cut value. Blobs with elongation param below this threshold passed the point-like cut	0.7	0 1
useMaxNPixCut	Apply cut on blob maximum number of pixels.	false	true false

psMaxNPix	Max number of pixels cut value.	1000	

4.8 Source Fitting Options

These options enable control of source fitting stage: minimization algorithm and relative parameters, starting parameters and limits, etc.

Option	Description	Default	Values
fitSources	Enable/disable source fitting stage	false	true false
fitUseThreads	Split source fitting among multiple threads. Multithread is not supported by Minuit minimizer	false	true false
fitScaleDataToMax	Scale source flux data to max peak flux if true, otherwise scale to mJy units	false	true false
fitMinimizer	Minimizer used in source fitting	Minuit2	Minuit Minuit2
fitMinimizerAlgo	Minimization algorithm used in source fitting	minimize	migrad simplex scan minimize fumili
fitPrintLevel	Minimizer printout level	1	
fitStrategy	Minimizer strategy parameter (larger means more accurate minimization but more fcn calls)	2	

continues on next page

Table 1 – continued from previous page

Option	Description	Default	Values
`fitFcnTolerance`	Fit function minimization tolerance (smaller means more accurate minimization but more fcn calls)	1.e-2	
`fitMaxIters`	Maximum number of iterations that can be done by minimizer before giving up and returning not converged fit	100000	
`fitImproveConvergence`	Try to improve convergence by iterating fit if not converged or converged with pars at limits	true	true false
`fitNRetries`	Number of times fit is repeated (with enlarged limits) if improve convergence flag is enabled	1000	
`fitParBoundIncreaseStepSize`	Par bound rel increase step size set when trying to improve convergence: parmax= parmax_old+(1+nretry)* *fitParBoundIncreaseStepSize* *0.5*|max-min|	0.1	

continues on next page

Table 1 – continued from previous page

Option	Description	Default	Values
fitDoFinalMinimizerStep	If enabled run HESSE minimizer at convergence to improve minimum and par error estimate limits	true	true false
fitRetryWithLessComponents	If fit fails to converge, repeat it iteratively with one component less at each cycle until convergence or until no more components are available	true	true false
nBeamsMaxToFit	Maximum number of beams for a compact source to be fitted (if above this threshold the fit is not performed)	20	
fitUseNestedAsComponents	If true use nested sources (if any) as starting fit components, otherwise estimate blended components in blob using a peak finding + segmentation algorithm	false	true false
fitMaxNComponents	Maximum number of components fitted in a blob	5	

continues on next page

Table 1 – continued from previous page

Option	Description	Default	Values
peakMinKernelSize	Minimum dilation kernel size in pixels used to detect start fit components	3	
peakMaxKernelSize	Maximum dilation kernel size in pixels used to detect start fit components	7	
peakKernelMultiplicityThr	Requested peak multiplicity across different dilation kernels. A multiplicity=-1 imposes that a peak must be found in all given dilation kernels (within a tolerance) to be considered a component	1	
peakShiftTolerance	Peak max position offset in pixels above which two peaks are considered distincs. Used to compare peaks found in different dilation kernels	2	

continues on next page

Table 1 – continued from previous page

Option	Description	Default	Values
peakZThrMin	Minimum peak flux significance (in nsigmas wrt source avg bkg and rms) below which peak is skipped and not considered as a fit component	1	
fitWithCentroidLimits	Apply limits to source centroid pars in fit	true	true false
fixCentroidInPreFit	Fix source centroid pars in pre-fit	false	true false
fitCentroidLimit	Source centroid par limits given as max offset in pixel with respect to starting fit centroid pars	3	
fitWithFixedBkg	Fix bkg level par in fit	true	true false
fitWithBkgLimits	Apply limits to bkg level par in fit	true	true false
fitUseEstimatedBkgLevel	Use estimated (avg bkg) as starting bkg level par in fit	true	true false
fitUseBkgBoxEstimate	Use bkg estimated in a box around source (if available) as bkg level par in fit	true	true false false

continues on next page

Table 1 – continued from previous page

Option	Description	Default	Values
fitBkgLevel	Starting bkg level par in fit (used when option fitParBoundIncreaseStepSize is false	0	
fitWithAmplLimits	Apply limits to amplitude par in fit	true	true false
fixAmplInPreFit	Fix amplitude par in pre-fit	true	true false
fitAmplLimit	Amplitude par limit given as max relative offset with respect to starting source component peak Speak*(1+-fitAmplLimit))	0.3	
fitWithSigmaLimits	Apply limits to sigma pars in fit	true	true false
fixSigmaInPreFit	Fix sigma pars in pre-fit	false	true false
fitSigmaLimit	Sigma par limit given as max relative offset with respect to starting component sigma pars	0.3	
fitWithFixedSigma	Fix sigma pars in fit	false	true false

continues on next page

Table 1 – continued from previous page

Option	Description	Default	Values
fitWithThetaLimits	Apply limits to theta par in fit	true	true false
fixThetaInPreFit	Fix theta par in pre-fit	false	true false
fitWithFixedTheta	Fix theta par in fit	false	true false
fitThetaLimit	Theta par limit given as max offset in degrees with respect to starting component theta par	5	
useFluxZCutInFit	If enabled only blob pixels above a significance threshold are included in chi2. Pixels below threshold are included in a regularization chi2 term	false	true false
fitZCutMin	Blob significance threshold below which pixels are included in the regularization chi2 term but not in the chi2	2.5	

continues on next page

Table 1 – continued from previous page

Option	Description	Default	Values
`fitChi2RegPar`	Fit chi2 regularization par so that total chi2 is given by chi2(Z>thr)+regPar*chi2(Z<thr)	0	

4.9 Source Fit Selection Cuts

These options enable control of source fit selection cuts. These cuts are used to assign flag to source fitted components.

Option	Description	Default	Values
fitApplyRedChi2Cut	Apply fit Chi2/NDF cut. Used to set fit quality flag. If Chi2/NDF>cut the good fit cut is not passed	true	true false
fitRedChi2Cut	Chi2/NDF cut value	5	
fitApplyFitEllipseCuts	Apply fit ellipse selection cuts. Used to set component flags. If not passed, fit component is tagged as fake	false	true false
fitEllipseEccentricityRatioMinCut	Ellipse eccentricity ratio (fit/beam) min cut value	0.5	
fitEllipseEccentricityRatioMaxCut	Ellipse eccentricity ratio (fit/beam) max cut value	1.5	
fitEllipseAreaRatioMinCut	Ellipse area ratio (fit/beam) min cut value	0.01	
fitEllipseAreaRatioMaxCut	Ellipse area ratio (fit/beam) max cut value	10	
fitEllipseRotAngleCut	Ellipse rot angle diff (|**fit-beam**|) cut value in degrees	45	

4.10 Source Residual Options

These options enable control of source residual map. Residual map is made by removing and/or subtracting detected sources from the input map. Source removal is done by replacing source pixel flux values (along with surrounding pixel around them, controlled by a dilation filter) with a residual model value, chosen among: average estimated background, median of source pixels. Residual model value can be randomized if desired. Source removal is controlled by two significance thresholds. Sources with fluxes above the higher threshold are removed regardless of any other conditions (e.g. on source type, etc). Sources with fluxes above the lower threshold (but below the higher threshold) are removed conditionally on chosen source type assigned in the finding process (e.g. point-like, compact, extended). Sources tagged as point-like can be removed with two different algorithms. The first one is described above and consists of replacing source pixel values by model values. The second method uses source fit model (if available) and subtract flux model from the input image. Removal of sources with nested components is controlled by the `removeNestedSources` flag. If enabled, the removal/subtraction process is done on nested sources and not on parent source pixels. On the contrary, sources are removed as described above and nested sources are removed, being part of the parent.

Option	Description	Default	Values
computeResidualMap	Compute compact source residual map (after compact source search)	false	true false
residualZHighThr	High source significance threshold (in nsigmas wrt bkg) used to remove sources	10	
residualZThr	Source significance threshold (in nsigmas wrt bkg) used to remove sources	5	
removeNestedSources	Remove nested sources instead of parent source is not supported by Minuit minimizer	true	true false
dilateKernelSize	Dilation filter kernel size in pixels used to remove sources. NB: Must be an odd number >1 This option controls the halo size around source to be removed	9	
removedSourceType	Type of sources to be removed threshold (in nsigmas wrt bkg) used to remove sources	2	-1=all types 1=compact 2=point-like 3=extended
residualModel	Residual model used to replace source pixel values	1	1=bkg 2=source median
residualModelRandomize	Randomize residual	false	true

4.11 Extended Source Finding Options

These options enable control of extended source search. Specific options for the available algorithms are reported in the Tables below. Superpixel Hierarchical Clustering algorithm is not currently available (not ported yet from CAESAR old repository).

Option	Description	Default	Values
searchExtendedSources	Enable/disable search of extended sources after compact source finding	false	true false
extendedSearchMethod	Extended source search method	4	1=Wavelet Transform 2=SP Hier Clustering 3=Active Contour 4=Saliency Filter
useResidualInExtendedSearch	Use residual map as input for extended source search source finding	true	true false
usePreSmoothing	Apply smoothing to residual map before performing extended source finding	true	true false
smoothFilter	Filter used to smooth residual map	2	1=gaus 2=guided
gausFilterKernSize	Gaussian filter kernel size in pixels. NB: Must be an odd value	5	
gausFilterSigma	Gaussian filter sigma par in pixels	1	
guidedFilterRadius	Guided filter radius par in pixels	12	
guidedFilterColorEps	Guided filter epsilon par (regularization parameter)	0.04	

4.12 Wavelet Transform Algorithm Options

These options enable control of extended source search with the Wavelet Transform method.

Option	Description	Default	Values
`wtScaleSearchMin`	Minimum Wavelet scale to be used for extended source search	3	
`wtScaleSearchMax`	Maximum Wavelet scale to be used for extended source search	6	

4.13 Active Contour Algorithm Options

These options enable control of extended source search with the Active Contour method. Two algorithms are provided: Chan-Vese, Linear Region-based Active Contour (LRAC).

Option	Description	Default	Values
acMethod	Active contour method	1	1=Chan-Vese 2=LRAC
acNIters	Maximum number of iterations	1000	
acInitLevelSetMethod	Level set initialization method	1	1=circle 2=checkerboard 3=saliency
acInitLevelSetSizePar	Level set size fraction wrt to minimum image size (e.g. circle radius=fraction x image size)	0.1	
acTolerance	Tolerance parameter to stop main iteration loop	0.1	0 1
cvNItersInner	Number of iteration done in inner cycle in Chan-Vese algo	5	
cvNItersReInit	Number of iteration done in re-initialization step in Chan-Vese algo	5	
cvTimeStepPar	Chan-Vese time step par	0.007	
cvWindowSizePar	Chan-Vese window size par	1	
cvLambda1Par	Chan-Vese lambda1 par	1	
cvLambda2Par	Chan-Vese lambda2 par	2	

4.14 Saliency Filtering Algorithm Options

These options enable control of extended source search with the Saliency Filtering method.

Option	Description	Default	Values
spBeta	Superpixel regularization parameter	1	
spMinArea	Superpixel min area parameter in pixels	10	
saliencyResoMin	Superpixel min scale par in pixels used in multi-scale saliency calculation	20	
saliencyResoMax	Superpixel max scale par in pixels used in multi-scale saliency calculation	60	
saliencyResoStep	Superpixel scale step par in pixels used in multi-scale saliency calculation	10	
saliencyNNFactor	Fraction of most similar superpixel neighbors used in saliency map computation	0.2	0 1
saliencyUseRobustPars	Use robust stats pars in saliency map computation	false	true false
saliencyDissExpFalloffPar	Superpixel dissimilarity exponential decay parameter used in saliency map computation	100	
saliencySpatialDistRegPar	Regularization parameter controlling superpixel spatial-intensity balance in in distance measure used for	1	

CHAPTER

FIVE

DATA PRODUCTS

Several data outputs are produced by caesar{} at the end of the processing if corresponding options are activated in the configuration file:

5.1 ROOT Output

If *saveToFile* option is enabled, a ROOT file is produced with different stored objects (depending on the activated options):

- ROOT TTree (named *SourceInfo*) with caesar Source objects, containing summary parameters plus detailed information at pixel level (see Source API section);
- ROOT TTree (named *ConfigInfo*) with a list of configuration options used in the run;
- ROOT TTree (named *PerformanceInfo*) with a list of run parameters (runtimes at different stages, used memory, etc);
  - Input map, stored as a caesar Image object (see Image API section)
  - Background, noise and significance maps, stored as caesar Image objects (see Image API section)
  - Residual map, stored as a caesar Image object (see Image API section)
  - Segmentation and saliency maps, stored as a caesar Image object (see Image API section)

5.2 DS9 Output

If *saveDS9Region* option is enabled, two ds9 region files are produced with the list of catalogued source islands and fitted components, respectively reported as labeled polygon or ellipse regions, each with a series of tags assigned (e.g. compact/point-like vs extended, real vs spurious, etc).

5.3 Ascii Output

If *saveToCatalogFile* option is enabled, two tabular ascii files are produced with a series of summary parameters for each catalogued source islands and fitted components, respectively.

Table format for source islands is described below:

Col No.	Name	Unit	Description
1	name		Source name assigned by finder
2	iauName		Source name in IAU notation
3	nPix		Number of pixels in island
4	nComp		Number of fitted components (=0 if fit not performed or failed)
5	nNested		Number of nested sources found in island
6-7	(x,y)		Source x,y position
8-9	(xw,yw)		Source x,y position weighted by pixel fluxes
10-11	(x_wcs,y_wcs)	deg	Source x,y sky position
12-13	(xw_wcs, yw_wcs)	deg	Source x,y sky position weighted by pixel fluxes
14-15	(xmin,xmax)		Source pixel min, max x coord.
16-17	(ymin,ymax)		Source pixel min, max y coord.
18-19	(xmin_wcs, xmax_wcs)	deg	Source pixel min, max x sky coord.
20-21	(ymin_wcs, ymax_wcs)	deg	Source pixel min, max y sky coord.
22	nu	GHz	Frequency value extracted from image header
23	Ssum	Jy/beam	Sum of island pixel brightness
24	Smax	Jy/beam	Max pixel brightness in island
25-26	(S,Serr)	Jy/beam	Source fitted flux brightness (not corrected by beam area) and its error
27	npix_beam		Number of pixels in beam
28	bkg_sum	Jy/beam	Background summed over island pixels
29	rms_sum	Jy/beam	Background noise summed over island pixels
30	F1		Source morph. flag {1=COMPACT,2=POINT-LIKE, 3=EXTENDED,4=COMPACT-EXTENDED}
31	F2		Sourceness flag{1=REAL,2=CANDIDATE,3=FAKE}
32	F3		Boolean flag indicating if source was tagged as good (true) or bad (false) in finding process
33	F4		Source depth level flag (nested source if >0)

Table format for source components is described below:

Col No.	Name	Unit	Description
1	name		Source name assigned by finder
2	nPix		Number of pixels in island
3	compId		Component id
4	iauName		Source name in IAU notation
5-6	(x,y)		Component x,y position
7-8	(xerr,yerr)		Component x,y position errors
9-10	(x_wcs,y_wcs)	deg	Component x,y sky position
11-12	(xerr_wcs,yerr_wcs)	deg	Component x,y sky position errors
13	nu	GHz	Frequency value extracted from image header
14-15	(Speak,Speak_err)	Jy/beam	Component peak brightness and error
16-17	(S,Serr)	Jy/beam	Component brightness (not corrected by beam area) and error
18-19	(S_i,Serr_i)	Jy/beam	Island brightness (not corrected by beam area) and error
20	npix_beam		Number of pixels in beam
21-23	(a,b,theta)		Component ellipse pars
24-26	(a_err,b_err,theta_err)		Component ellipse par errors
27-29	(a_wcs,b_wcs,theta_wcs)	“,”,deg	Component ellipse pars in sky coords
30-32	(a_err_wcs,b_err_wcs, theta_err_wcs)	“,”,deg	Component ellipse par errors in sky coords
33-35	(bmaj,bmin,pa)	“,”,deg	Beam ellipse pars in sky coords
36-38	(a_deconv_wcs, b_deconv_wcs, theta_deconv_wcs)	“,”,deg	Component ellipse par in sky coords, deconvolved by beam
39	E/E_beam		Ratio of fitted and beam ellipse eccentricities
40	A/A_beam		Ratio of fitted and beam ellipse areas
41	psi	deg	Rotation angle between fit and beam ellipse
42	bkg_sum	Jy/beam	Background summed over island pixels
43	rms_sum	Jy/beam	Background noise summed over island pixels
44-45	(chi2,ndf)		Source island fit chisquare and degrees of freedom
46	F1		Source fit quality flag {0=BAD,1=LOW,2=MEDIUM,3=HIGH}
47	F2		Component sourceness flag {1=REAL,2=CANDIDATE,3=FAKE}
48	F3		Component morph. flag {1=COMPACT,2=POINT-LIKE,3=EXTENDED, 4=COMPACT-EXTENDED}

5.4 FITS Output

If *saveToFITSFile* option is enabled, a series of FITS image files are produced (if corresponding options are activated):

- Input map
- Background, noise and significance maps
- Residual map
- Saliency map

CHAPTER

SIX

TUTORIALS

In this wiki section you find some guidance on how to run the applications provided with CAESAR in the *apps* directory and installed in the installation *bin* directory.

6.1 Compute image background info

Say you have a FITS image (e.g. *input.fits*) with your radio observations and you want to estimate the background and noise maps. You can use the executable `FindBkg` installed in the CAESAR *bin* directory to this aim:

```
$ ./FindBkg [options]

========== USAGE ==========
Usage: FindBkg [options]

*** Mandatory options ***
--input=[FILENAME] - Input file name containing image to be read (NB: .fits/.root␣
↪supported)

*** Optional options ***
-h, --help - Show help message and exit
--boxsize=[SIZE] - Size of sampling box in pixels or expressed as a multiple of the␣
↪image beam size (if --sizeinbeam option is given) (default=100 pixels)
--sizeinbeam - Consider box size option expressed in multiple of beam size (beam info␣
↪read from image) (default=no)
--gridsize=[SIZE] - Size of the interpolation grid expressed as fraction of the sampling␣
↪box (default=0.25)
--estimator=[ESTIMATOR] - Bkg estimator used in the sampling box (1=mean, 2=median,␣
↪3=biweight, 4=clipped median) (default=2)
--2ndpass - If given, perform a 2nd pass in bkg calculation (default=no)
--skipblobs - If given, skip blobs using a flood-fill algorithm (default=no)
--seedthr=[NSIGMAS] - Seed threshold in flood-fill algorithm in nsigmas significance␣
↪(default=5)
--mergethr=[NSIGMAS] - Merge threshold in flood-fill algorithm in nsigmas significance␣
↪(default=2.6)
--minnpixels=[NPIX] - Minimum number of pixels in a blob (default=5)
--nthreads=[N] - Number of threads to be used for reading (-1=all available threads)␣
↪(default=1)
--output=[FILENAME] - ROOT file where to save output maps (default=bkg.root)
--output-bkg=[FILENAME] - FITS file where to save bkg map (if --fitsout is given)␣
↪(default=bkg.fits)
```

(continues on next page)

(continued from previous page)

```
--output-rms=[FILENAME] - FITS file where to save rms map (if --fitsout is given)␣
↪(default=rms.fits)
--significance - Save the significance map (along with bkg and noise maps) in output␣
↪file (default=no)
--output-significance=[FILENAME] - FITS file where to save significance map (if --
↪fitsout is given) (default=significance.fits)
--fitsout - Write results in FITS files (default=no)
--parallel - Use parallel std algorithms for median (default=no)
-v [LEVEL], --verbosity=[LEVEL] - Log level (<=0=OFF, 1=FATAL, 2=ERROR, 3=WARN, 4=INFO, >
↪=5=DEBUG) (default=INFO)
==========================
```

As you can imagine, the *FindBkg* example simply uses the CAESAR `Image` and `BkgFinder` class methods. The meaning of each command line option is briefly reported in the program help in brief and in the configuration option section. More details on the algorithm used to compute the background maps are given in the CAESAR reference papers (see citing & reference section).

The example produces a ROOT file (with name specified in the *output* option) with three CAESAR image object stored:

- *bkgMap*: the background map
- *rmsMap*: the noise map
- *significanceMap*: the significance map computed as (img-bkg)/noise

You can access (read, plot, manipulate) these maps using ROOT CLI or macros. if you are not familiar with ROOT here's a macro example (say it is called macro.C).

```
//Open file (you can see file content in ROOT CLI by typing '.ls' at the root prompt)
TFile* inputFile= new TFile("bkg.root","READ");

//Get access to significance map
Image* zmap= (Image*)inputFile->Get("significanceMap");

//Draw the map with COLZ option
zmap->SetStats(0);//disable drawing of stats box in canvas
zmap->GetHisto2D("")->Draw("COLZ");

//Do whatever you want with Image API, e.g. compute & print stats
zmap->ComputeStats(true);
zmap->PrintStats();
```

If the `--fitsout` option is given three distinct fits files are produced with background, rms and significance maps (if `--significance` option is given). The images below show the results of this background finder run on a sample image provided in this repository (*data/ScorpioSNRField.fits*):

```
FindBkg --input=ScorpioSNRField.fits --boxsize=20 --sizeinbeam --significance --fitsout
```

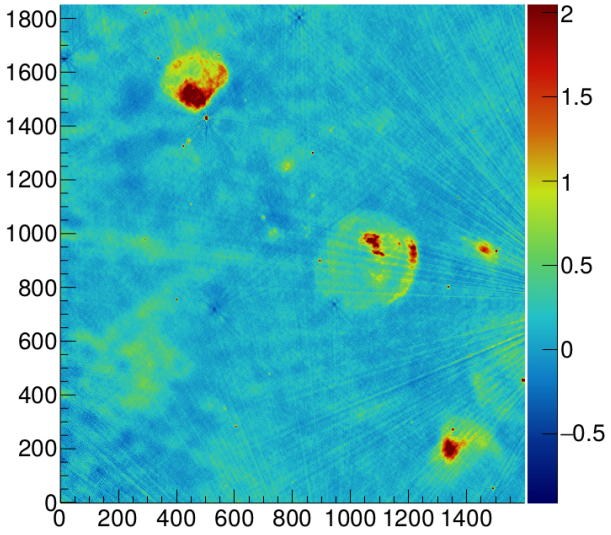

Fig. 1: Input image (units: mJy/beam)

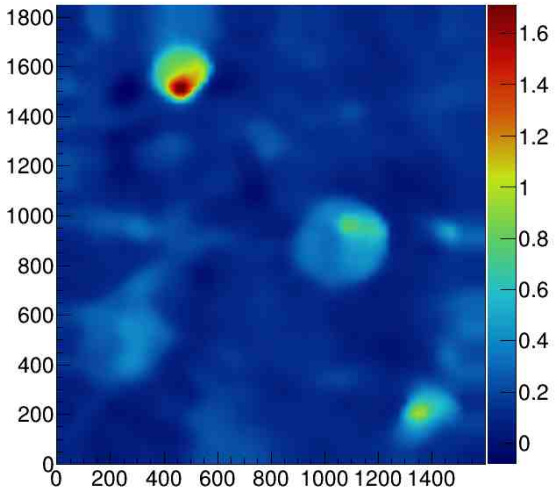

Fig. 2: Bkg map (units: mJy/beam)

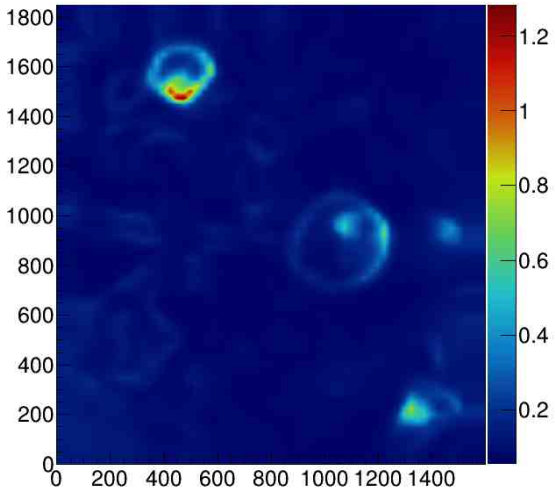

Fig. 3: Noise map (units: mJy/beam)

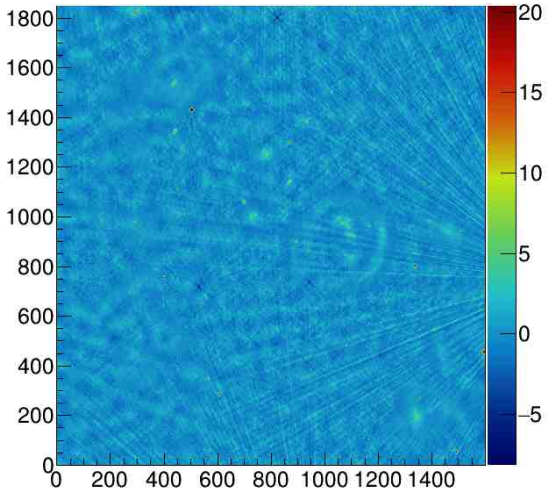

Fig. 4: Significance map (units: nsigmas)

6.2 Extracting compact sources

Say you have a FITS image (e.g. *input.fits*) with your radio observations and you want to extract compact and point-sources. You can use the executable `FindSource` or `FindSourceMPI` (which supports parallel computation) installed in the CAESAR *bin* directory to this aim:

```
$ ./FindSource --config=[config file]
```

```
$ mpirun -np [NPROC] ./FindSourceMPI --config=[config file] [--no-mpi]
```

A sample configuration file to search sources in the test image *ScorpioSNRField.fits* is reported below:

```
#==================
#==    MAIN       ==
#==================
inputFile = ScorpioSNRField.fits                    | Input image filename (ROOT or FITS)
##
//===========================
//==   BKG OPTIONS          ==
//===========================
useLocalBkg = true                                                                  ␣
↪                                                   | Use local background␣
↪calculation instead of global bkg (T/F)
bkgEstimator = 2                                                                    ␣
↪                                                               | Background␣
↪estimator (1=Mean,2=Median,3=BiWeight,4=ClippedMedian)
useBeamInfoInBkg = true                             | Use beam information in bkg box␣
↪definition (if available) (T/F)
boxSizeX = 20                                                                       ␣
↪                                                                   | X Size␣
↪of local background box in #pixels
boxSizeY = 20                                                                       ␣
↪                                                                   | Y Size␣
↪of local background box in #pixels
gridSizeX = 0.2                                                                     ␣
↪                                                                 | X Size of␣
↪local background grid used for bkg interpolation
gridSizeY = 0.2                                                                     ␣
↪                                                                 | Y Size of␣
↪local background grid used for bkg interpolation
###
###
//===================================
//==  SOURCE FINDING OPTIONS       ==
//===================================
searchCompactSources = true                                                         ␣
↪                                    | Search compact sources (T/F)
minNPix = 5                                                                         ␣
↪                                                                   | Minimum␣
↪number of pixel to consider a source
seedThr = 5                                                                         ␣
↪                                                                          |␣
↪Seed threshold in flood filling algo for faint sources
```

(continues on next page)

(continued from previous page)

```
mergeThr = 2.6                                                                  ␣
↪                                                                  | Merge/
↪aggregation threshold in flood filling algo
compactSourceSearchNIters = 2                          | Number of iterations to be␣
↪performed in compact source search (default=10)
seedThrStep = 0.5                                      | Seed threshold decrease step size␣
↪between iteration (default=1)
###
###
//=========================================
//==  NESTED SOURCE FINDING OPTIONS        ==
//=========================================
searchNestedSources = false                                                     ␣
↪                                  | Search for nested sources inside candidate␣
↪sources (T/F)
//=================================
//==  SOURCE FITTING OPTIONS      ==
//=================================
fitSources = false                                     | Fit sources with multi-component␣
↪gaus fit (T/F)
```

With the above configuration, after computing the significance map, the image is scanned for compact sources with a flood-fill algorithm, assuming a seed significance threshold of 5 sigmas and an aggregation threshold of 2.6 sigmas. A minimum number of pixels equal to 5 is considered in blob finding. Two search iterations are performed, decreasing detection threshold by 0.5 sigmas at each iterations. No negative excess wrt background are searched. No nested sources are searched. No fitting is performed over the detected sources.

A region file is created with the detected source as well as a ROOT file with full source information stored in the *SourceInfo* ROOT TTree. Each source in the tree is a *Source* object. You can view the original image with detected sources in DS9 as:

```
$ ds9 ScorpioSNRField.fits -regions ds9.reg
```

The image below shows the compact sources detected in the sample field

You can also get full access to detailed source info (parameters, contours, etc) from the ROOT output file using macros, CLI, etc. Source info is stored in the *SourceInfo* ROOT tree. For example, the following code gets access to all detected sources and dumps some basic source parameters:

```
{
  //Open output file with source finding results
  TFile* f= new TFile("output.root","READ");

  //Get access to source data tree
  TTree* data= (TTree*)f->Get("SourceInfo");

  Source* source= 0;
  data->SetBranchAddress("Source",&source);

  //Loop over all detected sources
  for(int i=0;i<data->GetEntries();i++)
  {
     //Get i-th source
     data->GetEntry(i);
```

(continues on next page)

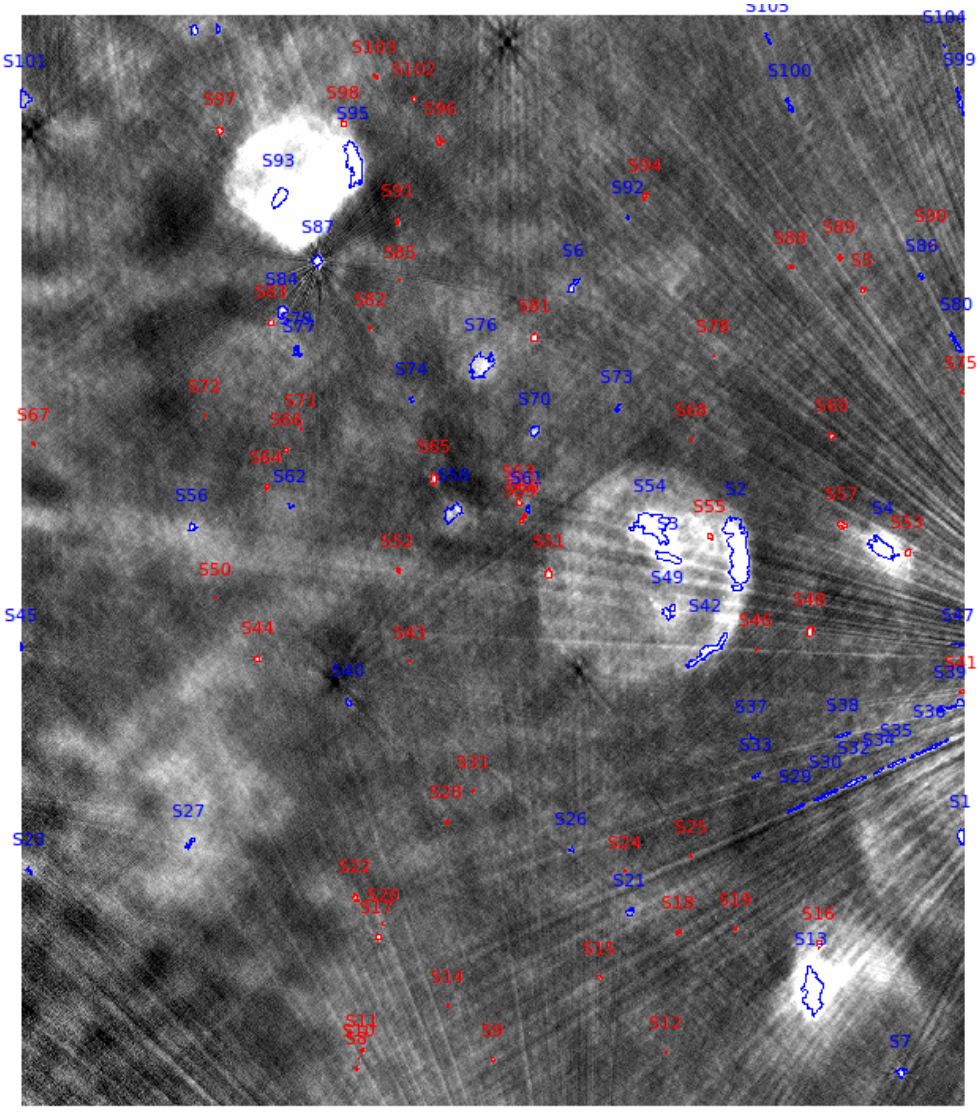

Fig. 5: Input image with detected compact sources (blue lines) and point-sources (red lines)

(continued from previous page)

```
    //Dump source info
    source->Print();

    //Do other stuff with i-th source object (see Source API)
    //...
  }
}
```

6.3 Remove/subtract compact sources from image

Say you have a FITS image (e.g. *input.fits*) with your radio observations and you want to produce a residual image with compact sources subtracted or removed. This is typically used as a pre-processing step before searching for extended sources. You can use the executable `FindSourceResidual` installed in the CAESAR *bin* directory to this aim:

```
$ ./FindSourceResidual [options]

========== USAGE ==========
Usage: FindSourceResidual [options]

*** Mandatory options ***
-i, --inputfile=[FILENAME]   Filename (fits/root) with input image.


*** Optional options ***
-h, --help   Show help message and exit
-s, --sourcefile=[FILENAME]          Caesar ROOT file with source list. If provided no␣
↪sources will be searched.
-o, --outputfile=[FILENAME]          Filename where to store output residual image␣
↪(default=resmap.fits)
-O, --outputfile_mask=[FILENAME]     Filename where to store output source mask image␣
↪(default=smask.fits)
-p, --psSubtractionMethod=[METHOD] - Method used to remove point sources (1=DILATION,
↪2=MODEL SUBTRACTION) (default=1)
-r, --resZThr=[NSIGMAS] - Significance threshold (in sigmas) above which sources are␣
↪removed (if selected for removal) (default=5)
-R, --resZHighThr=[NSIGMAS] - Significance threshold (in sigmas) above which sources are␣
↪always removed (even if they have nested or different type) (default=10)
-l, --removedSourceType=[TYPE] - Type of bright sources to be dilated from the input␣
↪image (-1=ALL,1=COMPACT,2=POINT-LIKE,3=EXTENDED)
-a, --removeNestedSources - If a source has nested sources, remove nested rather than␣
↪mother source (default=no)
-k, --dilateKernelSize=[SIZE] - Kernel size in pixel used to dilate image around sources␣
↪(default=9)
-z, --bkgAroundSource - Use bkg computed in a box around source and not from the bkg map␣
↪(default=use bkg map)
-Z, --bkgBoxThickness=[THICKNESS] - Bkg box thickness in pixels (default=20)
-j, --randomizeBkg - Randomize bkg in dilated pixels (default=no)
-T, --seedthr=[NSIGMAS] - Seed threshold in flood-fill algorithm in nsigmas significance␣
↪(default=5)
```

(continues on next page)

(continued from previous page)

```
-t, --mergethr=[NSIGMAS] - Merge threshold in flood-fill algorithm in nsigmas␣
↪significance (default=2.6)
-m, --minnpixels=[NPIX] - Minimum number of pixels in a blob (default=5)
-N, --no-nested - Do not search nested sources (default=search)
-b, --boxsize=[SIZE] - Size of sampling box in pixels or expressed as a multiple of the␣
↪image beam size (if --sizeinbeam option is given) (default=100 pixels)
-B, --sizeinbeam - Consider box size option expressed in multiple of beam size (beam␣
↪info read from image) (default=no)
-g, --gridsize=[SIZE] - Size of the interpolation grid expressed as fraction of the␣
↪sampling box (default=0.25)
-e, --estimator=[ESTIMATOR] - Bkg estimator used in the sampling box (1=mean, 2=median,␣
↪3=biweight, 4=clipped median) (default=2)
-P, --2ndpass - If given, perform a 2nd pass in bkg calculation (default=no)
-S, --skipblobs - If given, skip blobs using a flood-fill algorithm (default=no)
-A, --no-selection - Do not select and retag input sources (default=apply selection)
-d, --no-maxnpixcut - Do not apply max n pixel cut (default=apply)
-D, --maxnpix=[MAX_NPIX] - max number of pixel to consider a source as point-like␣
↪(default=1000)
-f, --no-elongcut - Do not apply elongation cut (default=apply)
-G, --no-circratiocut - Do not apply circular ratio cut (default=apply)
-q, --no-ellipsearearatiocut - Do not apply ellipse area ratio cut (default=apply)
-u, --no-nbeamscut - Do not apply nbeams cut (default=apply)
-U, --maxnbeams=[MAX_NBEAMS] - Max number of beams in source to consider it as point-
↪like (default=10)
-n, --nthreads       Number of threads to be used (default=1)
-v [LEVEL], --verbosity=[LEVEL] - Log level (<=0=OFF, 1=FATAL, 2=ERROR, 3=WARN, 4=INFO, >
↪=5=DEBUG) (default=INFO)

Alternatively you can provide options in a configuration file with:
-c, --config=[FILENAME]      Config file containing option settings

==============================
```

FindSourceResidual app mainly uses the CAESAR `Image`, `BkgFinder` and `MorphFilter` class methods. The meaning of each command line option is briefly reported in the program help. The main algorithm steps are:

1) Extract compact sources using flood-fill algorithm with options provided in the command line (`--seedthr`, `--mergeThr`, etc) or configuration file. This step is skipped if you provide an input file with the list of sources to be removed (`--sourcefile` option);

2) Select sources to be removed from the input image. These are the sources passing the quality criteria, if selection is enabled (`--no-selection`, `--no-maxnpixcut`, etc.), and sources above a chosen significance threshold. Sources above a high significance threshold (`--resZHighThr`) are always removed no matter what their type flag is. Sources above a lower significance threshold (`--resZThr`) are removed only if their type is equal to desired type (`--removedSourceType`). If `--removeNestedSources` option is given, only the nested sources are removed, not the mother source;

3) Selected sources are "removed" as follows. Sources with available fit information are "subtracted" if `--psSubtractionMethod=2` option is given, e.g. the fitted gaussian model is subtracted from the input image. In the other cases, source pixel and neighbouring pixel (depending on the dilation kernel size) values are replaced with the estimated background values. Background values can be obtained either from the computed local background map (see *FindBkg* app) or from the pixel median value (excluding source pixels) computed in a box centred around the source (if `--bkgAroundSource` option is given);

This application produces two outputs:

- *resMap*: a FITS file with residual image;
- *mask*: a FITS file with a binary source mask;

CHAPTER

SEVEN

IMAGE API

7.1 Image

This class defines the image base structure providing methods for: * reading from file (fits, root, png/gif/...) * writing to file (fits, root, ...) * filtering map * computing bkg & noise map * computing significance map * searching blobs/sources * TBD

Defined in `src/img/Image.h`

class `Caesar::Image` : public TNamed

Public Functions

Image()
Class constructor: initialize structures.

virtual **~Image**()
Class destructor: free allocated memory.

inline void **SetName**(std::string name)
Set image name.

inline std::string **GetName**()
Get image name.

inline long int **GetNPixels**()
Get npixels.

inline const std::vector<float> &**GetPixels**() const
Return pixel list.

inline long int **GetPixelDataSize**()
Get pixel vector size.

inline bool **HasPixelData**()
Has pixel data.

int **SetSize**(long int Nx, long int Ny, float xlow = 0, float ylow = 0)
Set image size (NB: this cleares pixel vector and allocated space!!!)

inline void **SetXmin**(float val)
Set image Xmin.

inline void **SetYmin**(float val)
Set image Ymin.

inline void **GetSize**(long int &Nx, long int &Ny)
Get size.

inline long int **GetNx**()
Get size x.

inline long int **GetNy**()
Get size y.

inline float **GetXmin**()
Get xmin.

inline float **GetXmax**()
Get xmax.

inline float **GetYmin**()
Get ymin.

inline float **GetYmax**()
Get ymax.

inline void **GetRange**(float &xmin, float &xmax, float &ymin, float &ymax)
Get range.

inline float **GetX**(long int binx)
Get x value corresponding to pixel bin x (NB: if no x offset is given it should return the same bin)

inline float **GetY**(long int biny)
Get y value corresponding to pixel bin y (NB: if no y offset is given it should return the same bin)

inline long int **GetBin**(long int binx, long int biny)
Get global bin corresponding to x & y bins.

inline long int **GetBinX**(long int gbin)
Get bin x corresponding to global bin.

inline long int **GetBinY**(long int gbin)
Get bin y corresponding to global bin.

inline long int **HasSameBinning**(*Image* *img, bool checkRange = false)
Check if this and a given image have the same binnings (and also range if required)

inline float **GetPixelValue**(long int gbin)
Get pixel value.

inline float **GetPixelValue**(long int ix, long int iy)
Get pixel value.

inline int **SetPixelValue**(long int gbin, double w)
Set pixel value.

inline int **SetPixelValue**(long int ix, long int iy, double w)
Set pixel value.

inline float **GetBinContent**(long int ix, long int iy)
Get bin content (equivalent to GetPixelValue)

inline float **GetBinContent**(long int gbin)
Get bin content (equivalent to GetPixelValue)

inline int **SetBinContent**(long int ix, long int iy, double w)
Set bin content (equivalent to SetPixelValue)

inline int **SetBinContent**(long int gbin, double w)
Set bin content (equivalent to SetPixelValue)

inline *Image* ***GetCloned**(std::string name, bool copyMetaData = true, bool resetStats = true)
Clone image (return a new image)

inline void **Reset**()
Reset image (zero all pixels and clear stats, size & meta-data preserved)

inline float **GetMinimum**()
Get pixel minimum.

inline float **GetMaximum**()
Get pixel maximum.

inline bool **HasBin**(long int gbin)
Check if given bin id is within image range.

inline bool **HasBin**(long int binIdX, long int binIdY)
Check if given bin with image coordinates (ix,iy) is within image range.

inline bool **HasBin**(double x, double y)
Check if given bin with physical coordinates (x,y) is within image range.

inline bool **HasBinX**(long int ix)
Check if given bin with image coordinate ix is within image range.

inline bool **HasBinX**(double x)
Check if given physical coordinate x is within image range.

inline bool **HasBinY**(long int iy)
Check if given bin with image coordinate iy is within image range.

inline bool **HasBinY**(double y)
Check if given physical coordinate y is within image range.

inline long int **FindBin**(double x, double y)
Find bin corresponding to physical coordinates (x,y)

inline bool **IsBinContentInRange**(long int binIdX, long int binIdY, double minThr, double maxThr)
Check if bin content is within given value range.

inline bool **IsEdgeBin**(long int binIdX, long int binIdY)
Check if bin is at edge.

inline bool **IsEdgeBin**(long int gbin)
Check if bin is at edge.

int **CheckFillPixel**(long int gbin, double w)
Check filled pixel.

int **FillPixel**(long int ix, long int iy, double w, bool useNegativePixInStats = true)
Fill pixels (to be used to compute stats at fill time).

NB: use only in single-thread otherwise computed moments are not correct (+ race conditions)!!!

int **FillFromMat**(cv::Mat&, bool useNegativePixInStats = true)
Fill pixel (multithreaded version).

This updated the moments per thread. To get the cumulative moments use the *StatsUtils* Fill image from OpenCV mat object

int **FillFromTMatrix**(TMatrixD&, bool useNegativePixInStats = true)
Fill image from ROOT TMatrixD object.

int **FillFromTH2**(const TH2&, bool useNegativePixInStats = true)
Fill image from ROOT TH2 object.

int **ReadFITS**(std::string filename, int hdu_id = 1, int ix_min = -1, int ix_max = -1, int iy_min = -1, int iy_max = -1)
Read image from FITS file.

int **WriteFITS**(std::string outfilename)
Write image to FITS file.

Image ***GetTile**(long int ix_min, long int ix_max, long int iy_min, long int iy_max, std::string imgname = "")
Get image subregion or tile.

int **GetTilePixels**(std::vector<float> &pixels, long int ix_min, long int ix_max, long int iy_min, long int iy_max, bool useRange = false, double minThr = -std::numeric_limits<double>::infinity(), double maxThr = std::numeric_limits<double>::infinity(), std::vector<float> maskedValues = {}, bool requireFinitePixValues = false)
Get tile pixels.

It excludes NAN/inf and masked pixels (=0 by default) and optionally also pixels outside range

Image ***GetSourceCutout**(*Source* *source, int boxThickness = 1)
Get image cutout around a source.

int **ReadFile**(std::string filename, bool invert = false)
Read image from an image file.

inline void **CopyMetaData**(*ImgMetaData* *data)
Set image metadata information.

inline *ImgMetaData* ***GetMetaData**()
Get image metadata information.

inline bool **HasMetaData**()
Has metadata information.

int **ComputeMoments**(bool useRange = false, double minThr = -std::numeric_limits<double>::infinity(), double maxThr = std::numeric_limits<double>::infinity(), std::vector<float> maskedValues = {0})
Update moments (multithreaded version).

It excludes NAN/inf and masked (e.g. =0 by default) pixels and optionally also pixels outside given range

inline *ImgStats* ***GetPixelStats**()
Get stats information.

inline bool **HasStats**()
Check if stats has been computed.

int **ComputeStats**(bool computeRobustStats, bool forceRecomputing = false, bool useRange = false, double minThr = -std::numeric_limits<double>::infinity(), double maxThr = std::numeric_limits<double>::infinity(), bool useParallelVersion = false, std::vector<float> maskedValues = {0})
Compute stats information.

It excludes NAN/inf and masked (e.g. =0 by default) pixels and optionally also pixels outside given range

inline void **PrintStats**()
Print stats information.

inline void **LogStats**(std::string level = "INFO")
Print stats information.

inline void **SetMoments**(Caesar::StatMoments<double> moments)
Set stat moments.

inline Caesar::StatMoments<double> **GetMoments**()
Get stat moments.

int **GetTileMeanStats**(float &mean, float &stddev, long int &npix, long int ix_min, long int ix_max, long int iy_min, long int iy_max, bool useRange = false, double minThr = -std::numeric_limits<double>::infinity(), double maxThr = std::numeric_limits<double>::infinity(), std::vector<float> maskedValues = {})
Get tile mean stats.

int **GetTileMedianStats**(float &median, float &mad, long int &npix, long int ix_min, long int ix_max, long int iy_min, long int iy_max, bool useRange = false, double minThr = -std::numeric_limits<double>::infinity(), double maxThr = std::numeric_limits<double>::infinity(), std::vector<float> maskedValues = {})
Get tile median stats.

int **GetTileClippedStats**(Caesar::ClippedStats<float> &clipped_stats, long int &npix, double clipsigma, long int ix_min, long int ix_max, long int iy_min, long int iy_max, bool useRange = false, double minThr = -std::numeric_limits<double>::infinity(), double maxThr = std::numeric_limits<double>::infinity(), std::vector<float> maskedValues = {})
Get tile median stats.

int **GetTileBiWeightStats**(float &bwLocation, float &bwScale, long int &npix, double C, long int ix_min, long int ix_max, long int iy_min, long int iy_max, bool useRange = false, double minThr = -std::numeric_limits<double>::infinity(), double maxThr = std::numeric_limits<double>::infinity(), std::vector<float> maskedValues = {})
Get tile biweight stats.

ImgBkgData ***ComputeBkg**(int estimator = eMedianBkg, bool computeLocalBkg = true, int boxSizeX = 100, int boxSizeY = 100, double gridStepSizeX = 10, double gridStepSizeY = 10, bool use2ndPass = true, bool skipOutliers = false, double seedThr = 5, double mergeThr = 2.6, int minPixels = 10, bool useRange = false, double minThr = -std::numeric_limits<double>::infinity(), double maxThr = std::numeric_limits<double>::infinity())
Compute image bkg.

ImgBkgData ***ComputeBkg**(*ImgBkgPars* pars)
Compute image bkg (version with bkg parameter class argument)

Image ***GetSignificanceMap**(*ImgBkgData* *bkgData, bool useLocalBkg = false)
Compute significance map.

TH1D ***GetPixelHisto**(int nbins = 100, bool normalize = false)
Compute pixel histo.

double **FindOtsuThreshold**(int nbins = 100)
Find Otsu threshold.

double **FindOtsuThreshold**(TH1D *pixelHisto)
Find Otsu threshold.

double **FindValleyThreshold**(int nbins = 100, bool smooth = true)
Find valley threshold.

double **FindMedianThreshold**(double thrFactor)
Find median global threshold.

double **FindOptimalGlobalThreshold**(double thrFactor, int nbins = 100, bool smooth = true)
Find optimal global threshold.

Image ***GetBinarizedImage**(double threshold, double fgValue = 1, bool isLowerThreshold = false)
Get binarized image.

int **ApplyThreshold**(double thr_min, double thr_max = 1.e+99, double maskedValue = 0)
Apply threshold to image and replace pixel outside thr range with masked value.

double **FindCumulativeSumThr**(double threshold, bool useRange = false, double minThr = -std::numeric_limits<double>::infinity(), double maxThr = std::numeric_limits<double>::infinity())
Find image threshold at which the cumulative pixel sum (scaled to image sum) is lower than desired threshold.

int **FindCompactSource**(std::vector<*Source**>&, double thr, int minPixels = 10)
Find compact sources by thresholding.

int **FindCompactSource**(std::vector<*Source**> &sources, *Image* *floodImg = 0, *ImgBkgData* *bkgData = 0, double seedThr = 5, double mergeThr = 2.6, int minPixels = 10, bool findNestedSources = false, *Image* *blobMask = 0, double minNestedMotherDist = 2, double maxMatchingPixFraction = 0.5, long int nPixThrToSearchNested = 0)
Find compact sources.

int **FindNestedSource**(std::vector<*Source**> &sources, *Image* *blobMask, *ImgBkgData* *bkgData = 0, int minPixels = 5, double minNestedMotherDist = 2, double maxMatchingPixFraction = 0.5, long int nPixThrToSearchNested = 0)
Find nested sources.

int **FindBlendedSources**(std::vector<*Source**> &deblendedSources, std::vector<*ImgPeak*> &deblendedPeaks, double sigmaMin, double sigmaMax, double sigmaStep, int minBlobSize = 5, double thrFactor = 0, int kernelFactor = 1)
Find blended sources.

int **FindExtendedSource_CV**(std::vector<*Source**>&, *Image* *initSegmImg = 0, *ImgBkgData* *bkgData = 0, int minPixels = 10, bool findNegativeExcess = false, double dt = 0.1, double h = 1, double lambda1 = 1.0, double lambda2 = 2.0, double mu = 0.5, double nu = 0, double p = 1, int niters = 1000, double tol = 1.e-2, int niters_inner = 1, int niters_reinit = 10)
Find extended sources with ChanVese method.

int **AddSimPointSources**(long int nGenSources, float Smin = 1.e-4, float Smax = 1., float Sslope = 1.6, int marginX = 0, int marginY = 0, std::string genFileName = "gensources.dat")
Find extended sources with Hierarchical Clustering method.

Image ***GetMask**(*Image* *mask, bool isBinary = false)
Get masked image.

Image ***GetSourceMask**(std::vector<*Source**> const &sources, bool isBinary = false, bool invert = false, bool searchSourceCoords = false)
Get source masked image.

int **MaskSources**(std::vector<*Source**> const &sources, float maskValue = 0.)
Mask sources.

int **GetBkgInfoAroundSource**(*BkgSampleData* &bkgSampleData, *Source* *source, int boxThickness = 20, int bkgEstimator = eMedianBkg, *Image* *mask = 0, bool useParallelVersion = false, std::vector<float> maskedValues = {})
Measure bkg in a box around given source bounding box.

Box thickness= bkgbox_max-sourcebox_max

Source ***GetMorphTransformedSource**(*Source* *source, int morphTransform = eMORPH_DILATION, int kernSize = 3, *Image* *mask = 0)*

Returns a morph trasnformed source (e.g.

enlarged or shrinked, obtained by dilating/eroding given source with a square kernel, eventually accounting masked pixels)

Image ***GetSourceFitModelImage**(const std::vector<*Source**> &sources, bool useNested = true, double nsigmaTrunc = 5)*

Create image with bin contents equal to fitted sources.

int ***FillFromSourceFitModel**(*Source* *source, bool useNested = true, double nsigmaTrunc = 5)*

Fill image bins from source fit model.

int ***SubtractFittedSources**(const std::vector<*Source**> &sources, bool subtractNested = true, bool recomputeStats = true, double nsigmaTrunc = 5)*

Subtract fitted sources from image.

int ***SubtractFittedSource**(*Source* *source, bool subtractNested = true, bool recomputeStats = true, double nsigmaTrunc = 5)*

Subtract fitted source from image.

int ***FindPeaks**(std::vector<Caesar::*ImgPeak*> &peakPoints, std::vector<int> kernelSizes = {3, 5, 7}, int peakShiftTolerance = 1, bool skipBorders = true, int multiplicityThr = -1)*

Find image peaks.

TGraph ***ComputePeakGraph**(std::vector<int> kernelSizes = {3, 5, 7}, int peakShiftTolerance = 1, bool skipBorders = true, int multiplicityThr = -1)*

Find graph with image peaks.

Image ***GetSourceResidual**(std::vector<*Source**> const &sources, int KernSize = 5, int dilateModel = eDilateWithBkg, int dilateSourceType = -1, bool skipToNested = false, *ImgBkgData* *bkgData = 0, bool useLocalBkg = false, bool randomize = false, double zThr = 5, double zBrightThr = 20, int psSubtractionMethod = ePS_DILATION, *Image* *mask = 0, int bkgBoxThickness = 20)*

Returns a residual image obtained by dilating given sources with a random background.

int ***Scale**(double c)*

Scale image by a factor 'c'.

int ***Add**(*Image* *img, double c = 1, bool computeStats = false)*

Add an image to this (this= this + img*c)

Image ***GetNormalizedImage**(std::string normScale = "LINEAR", int normmin = 1, int normmax = 256, bool skipEmptyBins = true)*

Get normalized image.

Image ***GetLaplacianImage**(bool invert = false)*

Get image laplacian.

Image ***GetGuidedFilterImage**(int radius = 12, double eps = 0.04)*

Get guided filter image.

Image ***GetSmoothedImage**(int size_x = 3, int size_y = 3, double sigma_x = 1, double sigma_y = 1)*

Smooth image.

Image ***GetSaliencyMap**(int reso = 20, double regFactor = 1, int minRegionSize = 10, double knnFactor = 1, bool useRobust = false, double expFalloffPar = 100, double distanceRegPar = 1)*

Get single-reso saliency map.

Image ***GetMultiResoSaliencyMap**(int resoMin = 20, int resoMax = 60, int resoStep = 10, double beta = 1, int minRegionSize = 10, double knnFactor = 0.2, bool useRobustPars = false, double expFalloffPar = 100, double distanceRegPar = 1, double salientMultiplicityThrFactor = 0.7, bool addBkgMap = true, bool addNoiseMap = true, *ImgBkgData* *bkgData = 0, double saliencyThrFactor = 2, double imgThrFactor = 1, bool useOptimalThr = false)*

Get multi-reso saliency map.

std::vector<*Image**> **GetWaveletDecomposition**(int nScales)

Get image wavelength decomposition.

Image ***GetGradientImage**(bool invert = false)*

Get image gradient.

Image ***GetKirschImage**()*

Get kirsch image.

Image ***GetLoGImage**(bool invert = false)*

Get laplacian of gaussian image.

Image ***GetNormLoGImage**(int size = 3, double scale = 1, bool invert = false)*

Get scale-normalized laplacian of gaussian image.

Image ***GetBeamConvolvedImage**(double bmaj, double bmin, double bpa, int nsigmas = 5, double scale = 1)*

Get image convolved with an elliptical gaussian beam.

Image ***GetMorphDilatedImage**(int kernSize = 3, int niters = 1, bool skipZeroPixels = true)*

Get dilated filtered image.

Image ***GetMorphErodedImage**(int kernSize = 3, int niters = 1, bool skipZeroPixels = true)*

Get eroded filtered image.

Image ***GetMorphTopHatImage**(int kernSize = 3, int niters = 1, bool skipZeroPixels = true)*

Get tophat filtered image.

Image ***GetMorphClosingImage**(int kernSize = 3, int niters = 1, bool skipZeroPixels = true)*

Get morph closing filtered image.

Image ***GetMorphOpeningImage**(int kernSize = 3, int niters = 1, bool skipZeroPixels = true)*

Get morph opening filtered image.

Image ***GetMorphGradientImage**(int kernSize = 3, int niters = 1, bool skipZeroPixels = true)*

Get morph gradient filtered image.

Image ***GetMorphRecoImage**(double baseline, int kernSize = 3, double tol = 1.e-6)*

Get morph reco filtered image.

Image ***GetHDomeImage**(double baseline, int kernSize = 3)*

Get H-dome image.

TH2D ***GetHisto2D**(std::string histoname = "")*

Export to ROOT TH2.

TH2D ***GetWCSHisto2D**(std::string histoname = "", WCS *wcs = 0, bool useImageCoord = true)*

Export to ROOT TH2 converted to WCS (TO BE DEPRECATED)

Export to ROOT TH2 converted to WCS

cv::Mat **GetOpenCVMat**(std::string encoding = "64")

Convert image to OpenCV mat float image.

TMatrixD *GetMatrix()
Get ROOT matrix from image.

std::string GetPixelNumpyArrayStr()
Get numpy pixel array string.

int Draw(int palette = Caesar::eRAINBOW, bool drawFull = false, bool useCurrentCanvas = false, std::string units = "")
Draw image (plain image, no WCS, no sources)

int Draw(std::vector<*Source**> const &sources, int palette = Caesar::eRAINBOW, bool drawFull = false, bool useCurrentCanvas = false, std::string units = "")
Draw image and sources (no WCS axis)

int Plot(std::vector<*Source**> const&, bool useCurrentCanvas = true, bool drawFull = false, int paletteStyle = Caesar::eRAINBOW, bool drawColorPalette = true, bool putWCAxis = false, int coordSystem = -1, std::string units = "", bool save = false, std::string outFileName = "plot.png")
Draw image.

class Caesar::ImgPeak : public TObject

Public Functions

inline ImgPeak()
Constructor.

inline ImgPeak(double _x, double _y, double _S, long int _ix, long int _iy)
Parametric Constructor.

inline virtual ~ImgPeak()
Destructor.

class Caesar::ImgBkgPars : public TObject

Public Functions

inline ImgBkgPars()
Constructor.

inline virtual ~ImgBkgPars()
Destructor.

inline void SetLocalBkg(bool choice)
Turn on/off local bkg.

inline void SetDataRange(double minVal = -std::numeric_limits<double>::infinity(), double maxVal = std::numeric_limits<double>::infinity())
Set data range thresholds (min/max)

class Caesar::ImgRange : public TObject

Public Functions

inline **ImgRange**()
Constructor.

inline **ImgRange**(long int _nx, long int _ny, float _smin, float _smax, float _xlow, float _ylow)
Parametric constructor.

inline **ImgRange**(long int _nx, long int _ny, float _smin, float _smax)
Parametric constructor.

inline **ImgRange**(float _smin, float _smax, float _xlow, float _xup, float _ylow, float _yup)
Parametric constructor.

inline virtual **~ImgRange**()
Destructor.

inline void **GetRange**(float &xmin, float &xmax, float &ymin, float &ymax)
Get range.

inline void **GetSRange**(float &_smin, float &_smax)
Get signal range.

7.2 Image Metadata

This class defines the metadata structure stored in image. Defined in `src/img/ImgMetaData.h`

class `Caesar::ImgMetaData` : public TObject

Public Functions

ImgMetaData()
Constructor.

virtual **~ImgMetaData**()
Destructor.

void **SetFITSCards**(Caesar::*FITSFileInfo* &fits_info)
Set cards from FITS file.

inline std::string **GetWCSType**()
Get current WCS type.

inline void **SetWCSType**(std::string value)
Set current WCS type.

WCS ***GetWCS**(int coordSystem = -1)
Get world coordinate system (TO BE DEPRECATED)

Get world coordinate system

inline double **GetPixelArea**()
Get pixel area in deg^2.

inline double **GetBeamWidth**()
Get width of synthetic beam.

inline double **GetBeamArea**()
Get area of synthetic beam.

inline int **GetBeamWidthInPixel**()
Get pixel scale.

inline double **GetBeamFluxIntegral**()
Get flux correction from beam.

bool **HasBeamInfo**()
Check if beam info have been specified and are valid.

bool **HasFrequencyInfo**()
Check if frequency info have been specified and are valid.

7.3 Image Stats

This class defines the structure holding image statistical estimators. Defined in `src/img/ImgStats.h`

class `Caesar::ImgStats` : public TObject

Public Functions

ImgStats()
Constructor.

virtual **~ImgStats**()
Destructor.

void **Reset**()
Reset stats params.

void **Log**(std::string level = "INFO")
Log stats.

void **Print**()
Print stats to stdout.

std::string **GetPrintable**()
Get a printable string.

CHAPTER

EIGHT

IMAGE IO API

This section reports the image IO API.

8.1 FITS Image Importer

These classes provide methods for creating a Caesar image from a fits image file.

Defined in `src/imgio/FITSReader.h`

class `Caesar::`**`FITSReader`** : public TObject

Public Functions

`FITSReader``()`
Class constructor: initialize structures.

virtual **`~FITSReader`**`()`
Class destructor: free allocated memory.

Public Static Functions

static int **`Read`**(Caesar::*Image* &img, Caesar::*FITSFileInfo* &info, std::string filename, int hdu_id = 1, int ix_min = -1, int ix_max = -1, int iy_min = -1, int iy_max = -1, bool checkFile = true)
Read a FITS image & header and store it in Caesar img format (based on CFITSIO)

class **`FITSHeader`** : public TObject

class **`FITSFileInfo`** : public TObject

8.2 FITS Image Exporter

These classes provide methods for exporting a Caesar image to a fits image file.

Defined in `src/imgio/FITSWriter.h`

class `Caesar::`**`FITSWriter`** : public TObject

Public Functions

FITSWriter()
Class constructor: initialize structures.

virtual **~FITSWriter**()
Class destructor: free allocated memory.

Public Static Functions

static int **WriteFITS**(*Image* *img, std::string outfilename, bool recreate = true)
Write image to FITS file.

CHAPTER

NINE

IMAGE FILTERING API

This section reports the image filtering API.

9.1 Gaussian Filter

This class provides methods to apply a gaussian filter to an image. Defined in `src/imgfilt/GausFilter.h`

class `Caesar::GausFilter` : public TObject

Public Functions

GausFilter()
Class constructor: initialize structures.

virtual **~GausFilter**()
Class destructor: free allocated memory.

Public Static Functions

static *Image* ***GetGausFilter**(*Image* *image, double bmaj, double bmin, double bpa, int nSigmas = 5, double scale = 1)
Return gaussian-filtered image.

NB: If image is in Jy/beam the flux are not normalized properly. Normalization factor (see CASA https://open-bitbucket.nrao.edu/projects/CASA/repos/casa/browse/code/imageanalysis/ImageAnalysis/Image2DConvolver.tcc) should be f/A, where f=beamArea_final/beamArea_orig and A=2 pi sigma_x sigma_y. Here, the normalization factor is 1/A. TO BE DONE: Add method to compute the final beam area.

9.2 Gradient Filter

This class provides methods to apply a gradient filter to an input image. Defined in `src/imgfilt/GradientFilter.h`

class `Caesar::GradientFilter` : public TObject

Public Functions

GradientFilter()
Class constructor: initialize structures.

virtual **~GradientFilter**()
Class destructor: free allocated memory.

Public Static Functions

static *Image* ***GetGradientFilter**(*Image* *image)
Apply a gradient filter to image and return filtered image.

static *Image* ***GetLaplaceFilter**(*Image* *image)
Apply a laplacian filter to image and return filtered image.

9.3 Guided Filter

This class provides methods to apply a guided filter to an input image. Defined in `src/imgfilt/GuidedFilter.h`

class **GuidedFilter**

9.4 Kirsch Filter

This class provides methods to apply a Kirsch filter to an input image. Defined in `src/imgfilt/KirschFilter.h`

class `Caesar::`**KirschFilter** : public TObject

Public Functions

KirschFilter()
Class constructor: initialize structures.

virtual **~KirschFilter**()
Class destructor: free allocated memory.

Public Static Functions

static *Image* ***GetKirschFilter**(*Image* *image)
Apply Kirsh filter to input image and return filtered image.

9.5 LoG Filter

This class provides methods to apply a Logarith-of-Gaussian (LoG) filter to an image. Defined in `src/imgfilt/LoGFilter.h`

class `Caesar::LoGFilter` : public TObject

Public Functions

LoGFilter()
Class constructor: initialize structures.

virtual **~LoGFilter**()
Class destructor: free allocated memory.

9.6 Morphological Filter

This class provides methods to apply morphological filters (dilation, top-hat, ...) to an image. Defined in `src/imgfilt/MorphFilter.h`

class `Caesar::MorphFilter` : public TObject

Public Functions

MorphFilter()
Class constructor: initialize structures.

virtual **~MorphFilter**()
Class destructor: free allocated memory.

Public Static Functions

static *Image* ***ComputeWatershedFilter**(*Image* *img, *Image* *markerImg)
Compute Watershed filter.

static *Image* ***ComputeWatershedFilter**(std::vector<*Contour**> &contours, *Image* *img, *Image* *markerImg)
Compute Watershed filter map and get contours.

static *Image* ***ComputeHDomeFilter**(*Image* *img, double baseline, int kernSize = 3)
Compute H-dome filter.

static *Image* ***ComputeMorphRecoFilter**(*Image* *img, double baseline, int kernSize = 3, double tol = 1.e-6)
Compute H-dome filter.

Apply image reconstruction filter to image and return filtered image

static *Image* ***ComputeMorphRecoFilter**(*Image* *img, *Image* *marker_img, int kernSize = 3, double tol = 1.e-6)
Apply morph filter to image and return filtered image.

static *Image* ***ComputeMorphFilter***(*Image* *img, int morphOp, int kernSize = 3, int structElementType = eMORPH_RECT, int niters = 1, bool skipZeroPixels = true)
Apply morph filter to image and return filtered image.

static int ***FindPeaks***(std::vector<*ImgPeak*> &peakPoints, *Image* *img, std::vector<int> kernelSizes = {3, 5}, int peakShiftTolerance = 1, bool skipBorders = true, int peakKernelMultiplicityThr = -1)
Find peaks in image by combining local peaks found with different dilation kernel sizes.

static *Image* ***Dilate***(std::vector<long int> &peakPixelIds, *Image* *img, int KernSize, bool skipBorders = true)
Dilate image with specified kernel.

static *Image* ***GetFiltered***(std::vector<long int> &peakPixelIds, *Image* *img, int KernSize, int morphOp, int structElementType, int niters, bool skipBorders)
Apply a morphological operation (DILATE/ERODE, OPENING/CLOSING, ...) to input image with configurable kernel and other pars.

static int ***DilateAroundSources***(*Image* *img, std::vector<*Source*> const &sources, int KernSize = 5, int dilateModel = eDilateWithBkg, int dilateSourceType = -1, bool skipToNested = false, *ImgBkgData* *bkgData = 0, bool useLocalBkg = false, bool randomize = false, double zThr = 5, double zBrightThr = 20)
Dilate image around specified sources position.

static int ***DilateAroundSource***(*Image* *img, *Source* *source, int KernSize = 21, int dilateModel = eDilateWithBkg, *ImgBkgData* *bkgData = 0, bool useLocalBkg = true, bool randomize = false, *Image* *mask = 0, int bkgBoxThickness = 20)
Dilate image around a given source.

9.7 Saliency Filter

This class provides methods to apply Saliency filter to an input image. Defined in `src/imgfilt/SaliencyFilter.h`

class `Caesar::SaliencyFilter` : public TObject

Public Functions

SaliencyFilter()
Class constructor: initialize structures.

virtual ***~SaliencyFilter***()
Class destructor: free allocated memory.

Public Static Functions

static *Image* ***ComputeSaliencyMap***(*Image* *img, int reso = 20, double regFactor = 1, int minRegionSize = 10, double knnFactor = 1, bool useRobust = false, double expFalloffPar = 100, double distanceRegPar = 1)
Compute saliency map for one resolution.

static *Image* *ComputeMultiResoSaliencyMap(*Image* *img, int resoMin = 20, int resoMax = 60, int resoStep = 10, double beta = 1, int minRegionSize = 10, double knnFactor = 1, bool useRobustPars = false, double expFalloffPar = 100, double distanceRegPar = 1, double salientMultiplicityThrFactor = 0.7, bool addBkgMap = true, bool addNoiseMap = true, *ImgBkgData* *bkgData = 0, double saliencyThrFactor = 2, double imgThrFactor = 1, bool useOptimalThr = false)

Compute multi resolution saliency map.

9.8 Wavelet Transform Filter

This class provides methods to apply Wavelet Transform (WT) filter to an input image. Defined in `src/imgfilt/WTFilter.h`

class `Caesar::WTFilter` : public TObject

Public Functions

WTFilter()

Class constructor: initialize structures.

virtual **~WTFilter**()

Class destructor: free allocated memory.

Public Static Functions

static std::vector<*Image**> **GetDecomposition**(*Image* *image, int nScales)

Get Wavelet Transform decomposition of input image.

CHAPTER

TEN

IMAGE PROCESSING API

This section reports the image processing API.

10.1 Background Finder

These classes define image background data structure and algorithms to compute the image background and noise maps. Defined in `src/imgproc/BkgData.h` and `src/imgproc/BkgFinder.h`

class `Caesar::BkgFinder` : public TObject

Public Functions

BkgFinder()
Class constructor: initialize structures.

virtual **~BkgFinder**()
Class destructor: free allocated memory.

Public Static Functions

static *ImgBkgData* ***FindBkg**(*Image* *img, int estimator = eMedianBkg, bool computeLocalBkg = true, int boxSizeX = 100, int boxSizeY = 100, double gridStepSizeX = 10, double gridStepSizeY = 10, bool use2ndPass = true, bool skipOutliers = false, double seedThr = 5, double mergeThr = 2.6, int minPixels = 10, bool useRange = false, double minThr = -std::numeric_limits<double>::infinity(), double maxThr = std::numeric_limits<double>::infinity())
Find image background.

class `Caesar::ImgBkgData` : public TObject

Public Functions

ImgBkgData()
Standard constructor.

virtual **~ImgBkgData**()
Destructor.

ImgBkgData(const *ImgBkgData* &data)
Copy constructor.

ImgBkgData &**operator=**(const *ImgBkgData* &data)
Assignment Operator.

void **Copy**(TObject &data) const
Copy method.

inline void **ClearSamplings**()
Clear sampling data.

inline void **ClearBkgMap**()
Clear bkg map.

inline void **ClearNoiseMap**()
Clear noise map.

inline void **Clear**()
Clear data.

void **CopyBkgMap**(Caesar::*Image* *aMap)
Copy bkg map.

void **CopyNoiseMap**(Caesar::*Image* *aMap)
Copy noise map.

inline bool **HasLocalBkg**()
Has local bkg.

class **Caesar::BkgSampleData** : public TObject

Public Functions

inline **BkgSampleData**()
Constructor.

inline virtual **~BkgSampleData**()
Destructor.

inline void **CopyBkgData**(*BkgSampleData* aBkgSample)
Copy bkg data.

inline void **Log**(std::string level = "INFO")
Log info.

inline void **Print**()
Print info to stdout.

inline std::string **GetPrintable**()
Get printable string.

10.2 Blob/Source API

These classes define image pixel data structure. Defined in `src/imgproc/Pixel.h`

class `Caesar::Pixel` : public TObject

Public Types

enum **PixelType**
Pixel type.

Values:

enumerator **eNormal**

enumerator **eSeed**

enumerator **eHalo**

Public Functions

Pixel()
Class constructor: initialize structures.

Pixel(long int _gbin, long int _ix, long int _iy, double _x, double _y, double _S)
Parametric constructor.

Pixel(const *Pixel* &pixel)
Copy constructor.

virtual **~Pixel**()
Class destructor: free allocated memory.

Pixel &**operator=**(const *Pixel* &pixel)
Assignment Operator.

inline bool **operator==**(const *Pixel* &pixel) const
Equality Operator.

inline bool **operator<**(const *Pixel* &pixel) const
< Operator

inline int **AddPixelFlux**(*Pixel* *aPixel)
Sum pixel.

void **Copy**(TObject &pixel) const
Copy method.

inline void **SetPhysCoords**(double xx, double yy)
Set physical coordinates.

inline void **SetCoords**(long int i, long int j)
Set coordinates.

inline void **SetBkg**(double bkg, double noise)
Set bkg and rms value.

inline void **SetCurv**(double val)
Set curvature information.

inline void **SetEdge**(double val)
Set edgeness information.

inline std::pair<double, double> **GetBkg**()
Get bkg and noise.

inline double **GetCurv**()
Get curvature information.

inline double **GetEdge**()
Get edgeness information.

inline void **Print**()
Dump pixel info.

This class defines contour (point collection) data structure. Defined in `src/imgproc/Contour.h`

class `Caesar::Contour` : public TObject

Public Functions

Contour()
Class constructor: initialize structures.

Contour(const *Contour* &contour)
Copy constructor.

virtual **~Contour**()
Class destructor: free allocated memory.

Contour &**operator=**(const *Contour* &contour)
Assignment Operator.

void **Copy**(TObject &contour) const
Copy method.

inline int **GetN**()
Get number of points.

inline bool **HasPoints**()
Check if contour has points.

inline void **SetPoints**(Points p)
Set contour points.

inline Points **GetPoints**()
Get contour points.

inline TVector2 ***GetPoint**(int i)
Get contour point with given index.

TVector2 ***FindPoint**(double x, double y, double tol = 0)
Find point (with tolerance)

bool **IsPointInsideContour**(double x, double y)
Check if point is inside contour.

int **ApplyOffset**(double offsetX, double offsetY)
Apply offset to points and centroid.

inline int **GetPointXY**(double &x, double &y, int i)
Get contour point x & y.

inline void **AddPoint**(TVector2 p)
Add contour points.

inline int **SortPointsCounterClockWise**()
Sort points counter-clockwise.

inline void **Reset**()
Reset contour.

TGraph ***GetGraph**(bool addLastPoint = true)
Return a graph object with contour points.

TPolyLine ***GetBoundingBoxLine**()
Return a polyline object with bounding box.

TPaveText ***GetParamInfoBox**()
Return a info box with parameter values.

int **ComputeParameters**()
Compute contour parameters.

void **ComputeShapeParams**(std::vector<cv::Point2f> const &points)
Compute shape parameters.

void **ComputeMomentParams**(std::vector<cv::Point2f> const &points)
Compute moments pars.

int **ComputeFittedEllipse**()
Compute fitted ellipse.

TEllipse ***GetFittedEllipse**()
Return an ellipse object fitted to contour.

void **ComputeFourierDescriptors**()
Compute Fourier descriptors.

void **ComputeCentroidDistanceFD**()
Compute centroid distance FD.

void **ComputeBendingEnergy**()
Compute bending energy.

inline void **Dump**()
Dump.

inline void **Log**(std::string level = "INFO")
Log.

inline std::string **GetPrintable**()
GetPrintable.

This class defines an image blob data structure. Defined in `src/imgproc/Blob.h`

class `Caesar::Blob` : public TNamed
Subclassed by *Caesar::Region*, *Caesar::Source*

Public Functions

Blob()
Class constructor: initialize structures.

Blob(std::string name)
Parametric constructor.

Blob(std::vector<*Pixel**> const &pixels, std::string name = "")
Parametric constructor.

Blob(const *Blob* &blob)
Copy constructor.

virtual **~Blob**()
Class destructor: free allocated memory.

Blob &**operator=**(const *Blob* &blob)
Assignment Operator.

void **Copy**(TObject &blob) const
Copy method.

inline void **SetId**(int id)
Set blob id.

inline void **SetName**(std::string name)
Set blob name.

inline bool **IsAtEdge**()
Is blob at image edge.

inline void **SetEdgeFlag**(bool choice)
Set edge flag.

inline int **SetImageMetaData**(*ImgMetaData* *data, double crpix1Offset = 0, double crpix2Offset = 0)
Set image metadata.

inline *ImgMetaData* ***GetImageMetaData**()
get image metadata

inline bool **HasImageMetaData**()
Set image metadata.

inline long int **GetNPixels**()
Get number of pixels.

inline const PixelCollection &**GetPixels**() const
Get pixel collection.

inline void **SetPixels**(PixelCollection &pixels)
Set pixel collection.

int **AddPixel**(*Pixel* *pixel, bool makeCopy = false)
Add a pixel to collection.

inline bool **HasPixels**()
Has pixels stored?

inline *Pixel* ***GetPixel**(int index)
Get access to pixel by index.

inline std::vector<int> **GetSeedPixelIndexes**()
Get access to seed pixel indexes.

inline *Pixel* ***GetSeedPixel**()
Find and return largest-significance seed pixel.

inline void **MarkHaloPixels**(double ZThr)
Mark pixels with significance within given thresholds as halo pixels.

int **ComputeStats**(bool computeRobustStats = true, bool forceRecomputing = false, bool useParallelMedian = false)
Compute stats.

inline bool **HasStats**()
Has stats?

inline void **SetHasStats**(bool value)
Has stats?

int **ComputeMorphologyParams**()
Compute morphology parameters.

int **ComputeZernikeMoments**(int order = 6)
Compute Zernike moments.

inline bool **HasParameters**()
Has morphology parameters computed.

inline void **SetHasParameters**(bool value)
Set has morphology pars.

inline double **GetM1**()
Get 1st pixel moment (= mean)

inline void **SetM1**(double value)
Set 1st pixel moment (= mean)

inline double **GetM2**()
Get 2nd pixel moment.

inline void **SetM2**(double value)
Set 2nd pixel moment.

inline double **GetM3**()
Get 3rd pixel moment.

inline void **SetM3**(double value)
Set 3rd pixel moment.

inline double **GetM4**()
Get 4th pixel moment.

inline void **SetM4**(double value)
Set 4th pixel moment.

inline double **GetM1Curv**()
Get 1st pixel curvature moment (= mean)

inline void **SetM1Curv**(double value)
Set 1st pixel curvature moment (= mean)

inline double **GetM2Curv**()
Get 2nd pixel curvature moment.

inline void **SetM2Curv**(double value)
Set 2nd pixel curvature moment.

inline double **GetS**()
Get pixel flux sum.

inline void **SetS**(double value)
Set pixel flux sum.

inline double **GetSmax**() const
Get pixel flux max.

inline void **SetSmax**(double value)
Set pixel flux max.

inline double **GetSmin**() const
Get pixel flux min.

inline void **SetSmin**(double value)
Set pixel flux min.

inline double **GetSxx**()
Get pixel flux XX correlation.

inline void **SetSxx**(double value)
Set pixel flux XX correlation.

inline double **GetSyy**()
Get pixel flux YY correlation.

inline void **SetSyy**(double value)
Get pixel flux YY correlation.

inline double **GetSxy**()
Get pixel flux XY correlation.

inline void **SetSxy**(double value)
Set pixel flux XY correlation.

inline double **GetSx**()
Get pixel flux sum weighted by position x.

inline void **SetSx**(double value)
Set pixel flux sum weighted by position x.

inline double **GetSy**()
Get pixel flux sum weighted by position y.

inline void **SetSy**(double value)
Set pixel flux sum weighted by position y.

inline long int **GetSmaxPixId**()
Get id of maximum flux pixel.

inline void **SetSmaxPixId**(long int value)
Set id of maximum flux pixel.

inline long int **GetSminPixId**()
Get id of minimum flux pixel.

inline void **SetSminPixId**(long int value)
Set id of minimum flux pixel.

inline double **GetScurv**()
Get pixel sum of curvature.

inline void **SetScurv**(double value)
Set pixel sum of curvature.

inline double **GetSedge**()
Get pixel sum of edgeness.

inline void **SetSedge**(double value)
Set pixel sum of edgeness.

inline void **GetSourceRange**(float &xmin, float &xmax, float &ymin, float &ymax)
Get source x-y range.

inline long int **GetXmin**()
Get source pixel x_min.

inline long int **GetXmax**()
Get source pixel x_max.

inline long int **GetYmin**()
Get source pixel y_min.

inline long int **GetYmax**()
Get source pixel y_max.

inline void **SetSourceRange**(float xmin, float xmax, float ymin, float ymax)
Set source x-y range.

inline void **GetSourcePixelRange**(long int &ixmin, long int &ixmax, long int &iymin, long int &iymax)
Get source pixel coordinate range.

inline long int **GetIxMin**()
Get source pixel ix_min.

inline long int **GetIxMax**()
Get source pixel ix_max.

inline long int **GetIyMin**()
Get source pixel iy_min.

inline long int **GetIyMax**()
Get source pixel iy_max.

inline void **SetSourcePixelRange**(long int ixmin, long int ixmax, long int iymin, long int iymax)
Set source pixel coordinate range.

int **GetSampleStdDev**(double &sigmaX, double &sigmaY, double &covXY)
Get sample source standard deviations.

inline void **Print**()
Dump blob info.

Image ***GetImage**(ImgType mode, int pixMargin = 1, bool includeHaloPixels = false)
Generate an image from source pixel.

TH2D ***GetWCSHisto**(ImgType mode, int pixMargin = 1, int coordSyst = -1)
Generate an image from source pixel.

Image ***GetNestedBlobMask**(*Image* *sourceImage = 0, int pixMargin = 1, double nestedBlobMinScale = 1, double nestedBlobMaxScale = 3, double nestedBlobScaleStep = 1, double nestedBlobPeakZThr = 5, double nestedBlobPeakZMergeThr = 2.5, int NMinPix = 5, double nestedBlobThrFactor = 0, double nestedBlobKernFactor = 6)
Get blob mask.

inline bool **HasContours**()
Has contours.

inline std::vector<*Contour**> **GetContours**()
Return contours.

NB: Do not delete pointers.

inline *Contour* ***GetContour**(int index)
Return contour with index.

std::vector<*Contour**> **GetWCSContours**(WCS *wcs = 0, int coordSystem = -1, double pixOffset = 0, bool computePars = false, bool castCoordsToInt = false)
Return contours converted in WCS.

Contour ***GetWCSContour**(int index, WCS *wcs = 0, int coordSystem = -1, double pixOffset = 0, bool computePars = false, bool castCoordsToInt = false)
Return contour with index and convert to WCS.

inline void **AddContour**(*Contour* *aContour)
Add contour.

bool **IsPointOnContour**(double x, double y, double tol = 1)
Is point on contour?

inline double **GetBkgSum**()
Get bkg sum.

inline double **GetBkgRMSSum**()
Get bkg rms sum.

inline void **SetBkgSum**(double bkg)
Set bkg sum (USED BY SERIALIZER)

inline void **SetBkgRMSSum**(double rms)
Set bkg rms sum (USED BY SERIALIZER)

inline bool **HasBoxBkgInfo**()
Check if bkg info in a box around the blob has ben set.

inline void **SetHasBoxBkgInfo**(bool flag)
Set has box bkg info flag (USED BY SERIALIZER)

inline double **GetBoxBkg**()
Get bkg estimate in a box around the blob.

inline double **GetBoxBkgRMS**()
Get bkg rms estimate in a box around the blob.

inline void **SetBoxBkgInfo**(double bkg, double rms)
Set bkg and rms estimate in a box around the blob.

inline WCS ***GetWCS**(int coordSystem = -1)
Get WCS from stored metadata (TO BE DEPRECATED)

Get WCS from stored metadata

This class defines a source data structure. Defined in `src/imgproc/Source.h` and inheriting from Blob class.

class `Caesar::Source` : public Caesar::*Blob*

Public Functions

Source()
Class constructor: initialize structures.

Source(std::string name)
Parametric constructor.

Source(std::vector<*Pixel**> const &pixels, std::string name = "")
Parametric constructor.

Source(const *Source* &source)
Copy constructor.

virtual **~Source**()
Class destructor: free allocated memory.

Source &**operator=**(const *Source* &source)
Assignment Operator.

void **Copy**(TObject &source) const
Copy method.

inline void **SetType**(SourceType choice)
Set source type.

inline void **SetFlag**(SourceFlag choice)
Set source flag.

inline void **SetSimType**(SimSourceType choice)
Set source sim type.

inline void **SetSimMaxScale**(float val)
Set source sim max scale.

inline void **SetBeamFluxIntegral**(double val)
Set beam flux integral.

inline double **GetBeamFluxIntegral**()
Get beam flux integral.

inline bool **IsGoodSource**()
Is a "good" source.

inline void **SetGoodSourceFlag**(bool flag)
Set source as "good".

inline void **SetDepthLevel**(int level)
Set source depth level (0=mother, 1=nested)

inline int **GetDepthLevel**()
Get source depth level (0=mother, 1=nested)

inline void **AddNestedSource**(*Source* *aNestedSource)
Add nested sources.

inline bool **HasNestedSources**()
Has nested sources?

inline void **SetHasNestedSources**(bool val)
Set has nested sources.

inline std::vector<*Source**> &**GetNestedSources**()
Get nested sources.

inline int **ClearNestedSources**()
Clear nested sources.

inline int **SetNestedSources**(std::vector<*Source**> &sources, bool clear_existing = true)
Set nested sources.

inline int **GetNestedSourceNumber**()
Get nested source number.

inline *Source* ***GetNestedSource**(int index)
Get nested source.

int **FindNestedSources**(std::vector<*Source**> &nestedSources, double nestedBlobMinScale = 1, double nestedBlobMaxScale = 3, double nestedBlobScaleStep = 1, double nestedBlobPeakZThr = 5, double nestedBlobPeakZMergeThr = 2.5, int minPixels = 5, double nestedBlobThrFactor = 0, double nestedBlobKernFactor = 6, double minNestedMotherDist = 2, double maxMatchingPixFraction = 0.5)
Find nested sources.

void **Draw**(bool drawBoundingBox = false, bool drawFittedEllipse = false, bool drawNested = false, int lineColor = kBlack, int lineStyle = kSolid)
Draw contours.

int **Draw**(int pixMargin = 0, ImgType imgType = eFluxMap, bool drawImage = true, bool drawContours = true, bool drawNested = true, bool drawFitComponents = true, int lineColor = kBlack, int lineStyle = kSolid, bool useWCS = false, int coordSyst = 0)
Draw source.

const std::string **GetDS9Region**(bool dumpNestedSourceInfo = false, bool convertToWCS = false, WCS *wcs = 0, int coordSystem = -1)
Get DS9 region info.

const std::string **GetDS9EllipseRegion**(bool dumpNestedSourceInfo = false)
Get DS9 ellipse info.

const std::string **GetDS9FittedEllipseRegion**(bool useFWHM = true, bool dumpNestedSourceInfo = false, bool convertToWCS = false, WCS *wcs = 0, int coordSystem = -1, bool useWCSSimpleConversion = true)
Get DS9 fitted ellipse info.

inline std::string **GetDS9RegionColor**()
Get DS9 region color according to source type.

inline std::string **GetDS9RegionTag**()
Get DS9 region tag according to source type.

inline void **Print**()
Dump source info.

inline bool **IsInsideSource**(*Source* *aSource)
Is source inside given source.

inline bool **IsAtBoxEdge**(float xmin, float xmax, float ymin, float ymax)
Check if source share boundary with given box.

inline bool **HasBoxOverlap**(float xmin, float xmax, float ymin, float ymax)
Is source inside given box.

bool **CheckBoxOverlapping**(*Source**)
Check if this source bounding box overlaps with another given source.

bool **IsAdjacentSource**(*Source* *aSource)
Check if this source is adjacent to another given source.

int **MergeSource**(*Source* *aSource, bool copyPixels = false, bool checkIfAdjacent = true, bool computeStatPars = true, bool computeMorphPars = true, bool sumMatchingPixels = false)
Merge this source with given source.

long int **GetNMatchingPixels**(std::vector<*Pixel**> &matching_pixels, *Source* *aSource, bool sorted = false)
Get collection of matching pixels between this and another source.

inline long int **GetNMatchingPixels**(*Source* *aSource, bool sorted = false)
Get number of matching pixels between this and another source.

bool **FindSourceMatchByOverlapArea**(SourceOverlapMatchPars &pars, const std::vector<*Source**> &sources, float overlapThr)
Find source match in a collection by overlapping area.

bool **FindSourceMatchByPos**(std::vector<SourcePosMatchPars> &pars, const std::vector<*Source**> &sources, float posThr)
Find source match in a collection by position.

float **GetCentroidDistance**(*Source* *aSource)
Get distance in pixels between source centroids.

int **Fit**(SourceFitOptions &fitOptions)
Fit source with a multi-component gaussian model.

int **Fit**(SourceFitOptions &fitOptions, SourceFitPars &initfitPars)
Fit source with a multi-component gaussian model using provided start fit parameters.

inline void **SetTrueInfo**(double S_true, double X0_true, double Y0_true)
Set true source info.

inline bool **HasTrueInfo**()
Has true source info.

inline double **GetTrueFlux**()
Get true source flux.

inline void **GetTruePos**(double &x, double &y)
Get true source position.

inline bool **HasFitInfo**()
Has fit info.

inline void **SetHasFitInfo**(bool flag)
Set Has fit info (for serialization scopes)

inline SourceFitPars &**GetFitPars**()
Get fit pars.

inline void **SetFitPars**(SourceFitPars &fitPars)
Set fit pars (for serialization scopes)

inline int **SetFitComponentFlag**(int componentId, int flag)
Set fit component flag.

inline int **GetFitComponentFlag**(int &flag, int componentId)
Get fit component flag.

inline int **GetFitStatus**()
Get fit status.

inline void **SetFitStatus**(int fitStatus)
Set fit status (for serialization scopes)

inline int **GetFluxDensity**(double &fluxDensity)
Get integrated flux density.

inline int **GetFluxDensityErr**(double &fluxDensityErr)
Get integrated flux density error.

inline int **GetFitQuality**()
Get fit quality flag.

inline int **GetCondonComponentFluxDensityErr**(std::vector<double> &fluxDensityErrList)
Get integrated flux density error on components according to Condon (1997) formula 14.

int **GetFitEllipses**(std::vector<TEllipse*> &fitEllipses, bool useFWHM = true, bool convertToWCS = false, WCS *wcs = 0, int coordSystem = -1, double pixOffset = 0, bool useWCSSimpleConversion = true)
Get fit ellipses.

inline int **GetNFitComponents**()
Get number of fit components.

inline int **GetNSelFitComponents**()
Get number of selected fit components.

int **FindComponentPeaks**(std::vector<*ImgPeak*> &peaks, double peakZThr = 0, int maxPeaks = -1, int peakShiftTolerance = 2, std::vector<int> kernels = {3, 5, 7}, int peakKernelMultiplicityThr = 1, bool invertSearch = false)
Find component peaks.

int **FindBlendedComponents**(std::vector<*Source**> &deblendedComponents, std::vector<*ImgPeak*> &deblendedPeaks, double peakZThr = 0, int maxPeaks = -1, double sigmaMin = 3, double sigmaMax = 3, double sigmaStep = 1, int minBlobSize = 5, double thrFactor = 0, int kernelFactor = 1, int pixMargin = 10)
Find blended source components.

inline int **GetWCSPos**(double &xwcs, double &ywcs, WCS *wcs = 0, int coordSystem = eJ2000)
Return source position in WCS coordinates.

inline int **GetWCSWeightedPos**(double &xwcs, double &ywcs, WCS *wcs = 0, int coordSystem = eJ2000)
Return signal-weighted source position in WCS coordinates.

inline int **GetWCSSourceRange**(double &xmin_wcs, double &xmax_wcs, double &ymin_wcs, double &ymax_wcs, WCS *wcs = 0, int coordSystem = eJ2000)
Return bounding box in WCS coordinates.

std::string **GetIAUName**(bool useWeightedPos = false, WCS *wcs = 0, int coordSystem = eJ2000)
Return source name following IAU convention.

int **GetSpectralAxisInfo**(double &val, double &dval, std::string &units)
Return spectral axis info.

inline bool **HasSpectralIndexData**()
Has spectral index data.

inline void **SetHasSpectralIndexData**(bool val)
Set has spectral index data flag.

inline SpectralIndexData &**GetSpectralIndexData**()
Get spectral index data.

inline void **SetSpectralIndexData**(SpectralIndexData &data)
Set spectral index data.

inline bool **HasComponentSpectralIndexData**()
Has component spectral index data.

inline void **SetHasComponentSpectralIndexData**(bool val)
Set has spectral index data flag.

inline std::vector<SpectralIndexData> &**GetComponentSpectralIndexData**()
Get component spectral index data.

inline void **SetComponentSpectralIndexData**(std::vector<SpectralIndexData> &data)
Set component spectral index data.

inline bool **HasAstroObjects**()
Has astro object data.

inline void **SetHasAstroObjects**(bool choice)
Has astro object data.

inline std::vector<AstroObject> &**GetAstroObjects**()
Get astro objects data.

inline void **SetAstroObjects**(std::vector<AstroObject> &data)
Set astro objects data.

int **AddAstroObject**(AstroObject &astroObject)
Add astro objects data.

inline bool **HasComponentAstroObjects**()
Has component astro object data.

inline void **SetHasComponentAstroObjects**(bool choice)
Has astro object data.

inline std::vector<std::vector<AstroObject>> &**GetComponentAstroObjects**()
Get astro objects data.

inline void **SetComponentAstroObjects**(std::vector<std::vector<AstroObject>> &data)
Set astro objects data.

int **AddComponentAstroObject**(int componentIndex, AstroObject &astroObject)
Add component astro objects data.

int **ComputeObjClassId**()
Compute object class id from astro object data (if available)

int **ComputeComponentObjClassId**()
Compute component object class id from astro object data (if available)

inline bool **IsResolved_XXLSurveyMethod**(double p0 = 1.08, double p1 = 2.03)
Compute if source island is resolved according to XXL survey criterion.

inline bool **IsComponentResolved_XXLSurveyMethod**(int componentId, double p0 = 1.08, double p1 = 2.03)
Compute if source fit component is resolved according to XXL survey criterion.

This class defines a source cube (source at different frequency channels) data structure. Defined in `src/imgproc/SourceCube.h`

class `Caesar::`**SourceCube** : public TNamed

Public Functions

SourceCube()
Class constructor: initialize structures.

SourceCube(std::string name)
Parametric constructor.

virtual **~SourceCube**()
Class destructor: free allocated memory.

SourceCube(const *SourceCube* &scube)
Copy constructor.

SourceCube &**operator=**(const *SourceCube* &scube)
Assignment Operator.

void **Copy**(TObject &scube) const
Copy method.

inline int **GetSize**()
Get cube size.

inline int **GetNMatches**()
Get number of matches.

inline int **AddSource**(*Source* *aSource)
Add source to cube.

inline void **AddComponentMatch**()
Add component.

int **AddIndexToComponent**(int cubeComponentIndex, size_t sindex, size_t componentIndex)
Add index to existing component.

int **DoSourceImagePlot**(bool useWCS = true, int coordSyst = 0)
Draw source images.

int **DoSourceSEDs**()
Draw source SEDs.

10.3 Superpixel API

These classes define image superpixel data structures. Defined in `src/imgproc/Region.h`, `src/imgproc/SLICData.h`, `src/imgproc/SLIC.h`

class `Caesar::Region` : public Caesar::*Blob*

Public Types

enum **RegionTag**
Region tag enumeration.

Values:

enumerator **eBkgTag**

enumerator **eSignalTag**

enumerator **eUntagged**

Public Functions

Region()
Class constructor: initialize structures.

Region(std::string name)
Parametric constructor.

Region(std::vector<*Pixel**> const &pixels, std::string name = "")
Parametric constructor.

Region(const *Region* &region)
Copy constructor.

virtual **~Region**()
Class destructor: free allocated memory.

Region &**operator=**(const *Region* &region)
Assignment Operator.

void **Copy**(TObject &region) const
Copy method.

Region::*RegionPars* ***GetParams**(bool includeCurvPar = true)
Get region parameters.

int **GetDistance**(DistPars &distPars, *Region* *aRegion, bool useRobustParams = false)
Get distance squared between this and given region.

int **GetAsymmDistance**(DistPars &distPars, DistPars &distPars_neighbor, *Region* *aRegion, bool useRobustParams = false)
Get asymmetric color & space symmetric distance between this and given region.

int **AddRegion**(*Region* *aRegion, bool addPixels = true, bool copyPixels = false)
Get color & space symmetric distance among two regions.

Get color & space asymmetric distance among two regions Merge a region to this

int **AddSubRegionId**(long int id)
Add a sub-region id to list.

inline int **GetNSubRegions**()
Get number of sub-regions present in list.

inline const std::vector<long int> &**GetSubRegionIds**() const
Get the list of sub-region ids.

inline long int **GetSubRegionId**(int index)
Get sub-region id at given index.

struct **RegionPars**
Region pars.

class `Caesar::RegionCollection` : public TObject

Public Functions

inline **RegionCollection**()
Constructor.

inline virtual **~RegionCollection**()
Destructor.

inline void **Add**(*Region* *aRegion)
Add a region to collection.

inline int **GetN**()
Get number of regions present in collection.

inline *Region* ***FindRegionById**(int id)
Get region by id.

inline int **FindRegion**(int id)
Find region (return index with given id)

inline *Region* ***GetRegion**(int index)
Get region with given index.

inline std::map<int, int> **GetRegionIdMap**() const
Get region map regionId>index.

inline std::map<int, int> **GetRegionIndexMap**() const
Get region index map (index>regionId)

class `Caesar::SLICData` : public TObject

Public Functions

SLICData()
Class constructor: initialize structures.

SLICData(const *SLICData* &data)
Copy constructor.

virtual **~SLICData**()
Class destructor: free allocated memory.

SLICData &**operator=**(const *SLICData* &region)
Assignment Operator.

void **Copy**(TObject &region) const
Copy method.

void **Clear**()
Clear data.

void **ClearImages**()
Clear images.

void **ClearRegions**()
Clear regions.

int **SetData**(*Image* *img, *Image* *lapl_img, *Image* *edge_img = 0)
Set data (NB: Pointer ownership is taken by this class)

int **SPGenerator**(*Image* *img, int regionSize = 10, double regParam = 1, int minRegionSize = 10, bool useLogScaleMapping = false, *Image* *edgeImg = 0)
Generate a superpixel partition given the passed options.

Image ***GetSegmentedImage**(*Image* *image, int selectedTag = -1, bool normalize = false, bool binarize = false)
Get segmented image.

inline int **GetNRegions**() const
Get number of regions.

inline std::vector<*Region**> **GetRegions**()
Return regions.

inline void **SetRegions**(std::vector<*Region**> &list)
Set regions.

inline *Region* ***GetRegion**(int index)
Get region.

inline int **AddRegion**(*Region* *aRegion)
Add region.

inline long int **GetRegionId**(long int index)
Get region id from index (NB: No check done)

inline long int **GetRegionSize**(long int index)
Get region size from index (NB: No check done)

int **ComputeRegionParameters**()
Compute region parameters.

void **RemoveEmptyRegions**()
Remove regions without pixels.

inline void **DeleteRegions**(std::vector<size_t> const &delete_indexes)
Delete regions specified in the list (NB: No check done)

void **GetRegionIdMap**(std::map<long int, long int> &regionIdMap) const
Get region map regionId>index.

void **GetRegionIndexMap**(std::map<long int, long int> &regionIndexMap) const
Get region index map (index>regionId)

inline void **SetPixelLabels**(std::vector<long int> list)
Set pixel labels.

int **SetPixelLabel**(long int gBin, long int label, bool check = true)
Set pixel label (NB: No check done)

inline void **ScalePixelLabel**(long int gBin, long int scale)
Scale pixel label (NB: No check done)

inline long int **GetPixelLabel**(long int ix, long int iy)
Get pixel label (NB: No check is done)

inline long int **GetPixelLabel**(long int gBin)
Get pixel label (NB: No check is done)

inline *ImgStats* ***GetCurvStats**()
Get image curvature stats.

inline *ImgStats* ***GetStats**()
Get image stats.

inline *ImgStats* ***GetEdgeStats**()
Get image edge stats.

inline double **GetScurv**(long int ix, long int iy)
Get curvature pixel value (NB: No check is done!)

double **GetSedge**(long int ix, long int iy)
Get edgeness pixel value.

inline double **GetS**(long int ix, long int iy)
Get pixel value (NB: No check is done!)

inline void **SetRegionId**(int index, int id)
Set region id (NB: No checks performed)

inline void **AddPixelToRegion**(int index, *Pixel* *pixel)
Add pixel to region (NB: No checks performed)

class `Caesar::`**SLICContourData** : public TObject

Public Functions

inline **SLICContourData**()
SLIC contour data constructor.

inline virtual **~SLICContourData**()
SLIC contour data destructor.

inline void **ResetContour**()
Reset *SLIC* contour.

inline void **ResetList**()
Reset connected regions list and clear *SLIC* boundary data.

class `Caesar::`**SLICNeighborData** : public TObject

Public Functions

inline **SLICNeighborData**()
SLIC neighbor data constructor.

inline virtual **~SLICNeighborData**()
SLIC neighbor data destructor.

inline bool **operator==**(const *SLICNeighborData* &aNeighborData) const
SLIC neighbor data equality operator for comparison.

inline void **Print**()
Print neighbor data info.

class Caesar::**SLICNeighborCollection** : public TObject

Public Functions

inline **SLICNeighborCollection**()
SLIC neighbor collection constructor.

inline virtual **~SLICNeighborCollection**()
SLIC neighbor collection destructor.

inline void **Add**(*SLICNeighborData* nn)
Add a *SLIC* neighbor data to collection.

inline void **SortByDiss**()
Sort *SLIC* neighbor data in collection by dissimilarity measure.

inline void **SortByEdgeness**()
Sort *SLIC* neighbor data in collection by edgeness measure.

inline void **SortByTotDiss**()
Sort *SLIC* neighbor data in collection by total dissimilarity measure.

inline int **GetN**()
Get number of neighbor data present in collection.

inline int **FindById**(int id)
Find a *SLIC* neighbor data in collection by id.

inline int **FindByIndex**(int index)
Find a *SLIC* neighbor data in collection by index.

inline int **FindCloserByDiss**()
Find the closest neighbor data in collection wrt dissimilarity.

inline int **FindCloserByDissTot**()
Find the closest neighbor data in collection wrt total dissimilarity.

inline std::vector<*SLICNeighborData*> **GetNSortedByDiss**(int N)
Find the closest neighbor data in collection in dissimilarity.

inline std::vector<*SLICNeighborData*> **GetNSortedByTotDiss**(int N)
Returns the number of *SLIC* neighbor data sorted by total dissimilarity.

inline bool **IsIdAmongNClosersByDiss**(int id, int N)
Check if neighbor data with given id is among the closest N in collection wrt dissimilarity measure.

inline bool **IsIdAmongNClosersByTotDiss**(int id, int N)
Check if neighbor data with given id is among the closest N in collection wrt total dissimilarity measure.

inline bool **IsIndexAmongNClosersByDiss**(int index, int N)
Check if neighbor data with given index is among the closest N in collection wrt dissimilarity measure.

inline bool **IsIndexAmongNClosersByTotDiss**(int index, int N)
Check if neighbor data with given index is among the closest N in collection wrt total dissimilarity measure.

inline void **Print**()
Print neighbor data collection info.

inline const std::vector<*SLICNeighborData*> &**GetNeighbors**() const
Returns *SLIC* neighbor data collection.

inline *SLICNeighborData* *GetNeighbor(int index)
Returns *SLIC* neighbor data with given index in collection.

int SetDtot(int index, double Dtot, double Dtot_n)
Set total dissimilarity info of neighbor data with the given index in collection.

class Caesar::SLICSimilarityData : public TObject

Public Functions

inline SLICSimilarityData()
SLIC similarity data constructor.

inline virtual ~SLICSimilarityData()
SLIC similarity data destructor.

class Caesar::SLIC : public TObject

Public Functions

SLIC()
Class constructor: initialize structures.

virtual ~SLIC()
Class destructor: free allocated memory.

Public Static Functions

static *SLICData* *SPGenerator(*Image* *img, int regionSize = 20, double regParam = 1, int minRegionSize = 10, bool normalizeImage = true, bool useLogScaleMapping = false, *Image* *laplImg = 0, *Image* *edgeImg = 0)
Generate superpixel partition.

static *SLICContourData* *ComputeBoundaryContours(*SLICData* *slicData)
Compute superpixel boundary contours.

static *SLICSimilarityData* *ComputeRegionSimilarity(*SLICData* *slicData, std::vector<*SLICNeighborCollection*> &neighbors, double beta = 0.5)
Compute region similarities.

static int FindNeighbors(std::vector<*SLICNeighborCollection*> &neighbors, *SLICData* *slicData, *SLICContourData* *contourData, bool get2ndNeighbors = true, int selectedTag = -1, bool includeSpatialDist = false, bool normalizeParams = true, bool useRobustParams = false, bool addCurvDist = true)
Find superpixel neighbors.

static *Image* *GetSegmentedImage(*Image* *img, std::vector<*Region**> const &regions, int selectedTag = -1, bool normalize = false, bool binarize = false)
Compute segmented image given a list of tagged regions.

static int CountTaggedRegions(std::vector<*Region**> const &regions, int &NSig, int &NBkg, int &NUntagged)
Count number of regions per tag.

static int **TagRegions**(std::vector<*Region**> &regions, *Image* *binaryMap_bkg, *Image* *binaryMap_signal)
Tag regions into signal/bkg according to signal & bkg marker images.

static int **ComputeRegionDistance**(double &dist, double &dist_spatial, *Region* *region_i, *Region* *region_j, RegionDistNormData normPars, bool normalizeParams = true, bool useRobustParams = false, bool addCurvDist = false)
Compute distance between regions.

static int **ComputeRegionAsymmDistance**(double &dist, double &dist_neighbor, *Region* *region_i, *Region* *region_j, RegionDistNormData normPars, bool normalizeParams = true, bool useRobustParams = false, bool addCurvDist = false, bool addSpatialDist = false)
Compute asymmetric distance between regions.

10.4 Blob Finder

These classes define blob data structure and algorithms to find blobs in image. Defined in `src/imgproc/Blob.h` and `src/imgproc/BlobFinder.h`

class `Caesar::BlobFinder` : public TObject

Public Functions

BlobFinder()
Class constructor: initialize structures.

virtual **~BlobFinder**()
Class destructor: free allocated memory.

Public Static Functions

template<class **T**>
static int **FindBlobs**(*Image* *inputImg, std::vector<*T**> &blobs, *Image* *floodImg = 0, *ImgBkgData* *bkgData = 0, double seedThr = 5, double mergeThr = 2.6, int minPixels = 10, bool findNegativeExcess = false, bool mergeBelowSeed = false, *Image* *curvMap = 0)
Find blobs.

template<class **T**>
static int **FindBlobsMT**(*Image* *inputImg, std::vector<*T**> &blobs, *Image* *floodImg = 0, *ImgBkgData* *bkgData = 0, double seedThr = 5, double mergeThr = 2.6, int minPixels = 10, bool findNegativeExcess = false, bool mergeBelowSeed = false, *Image* *curvMap = 0)
Find blobs (multithreaded version)

template<class **T**>
static int **FindBlobsST**(*Image* *inputImg, std::vector<*T**> &blobs, *Image* *floodImg = 0, *ImgBkgData* *bkgData = 0, double seedThr = 5, double mergeThr = 2.6, int minPixels = 10, bool findNegativeExcess = false, bool mergeBelowSeed = false, *Image* *curvMap = 0)
Find blobs (single-thread version)

static int **FloodFill**(*Image* *img, std::vector<long int> &clusterPixelIds, long int seedPixelId, double floodMinThr, double floodMaxThr)
Flood fill algorithm.

static *Image* ***ComputeBlobMask**(*Image* *img, double Bmaj, double Bmin, double Bpa = 0, double kernNSigmaSize = 2.5, double peakZThr = 5, double peakZMergeThr = 2.5, int minBlobSize = 5, double thrFactor = 0, int bkgEstimator = eMedianBkg, int bkgBox = 100, double bkgGridStepSize = 20)
Compute single-scale blob mask using elliptical gaus smoothing + laplacian filter map.

static *Image* ***ComputeMultiScaleBlobMask**(*Image* *img, double sigmaMin, double sigmaMax, double sigmaStep, double peakZThr = 5, double peakZMergeThr = 2.5, int minBlobSize = 5, double thrFactor = 0, int kernelFactor = 1, bool useLocalBkg = true, int bkgEstimator = eMedianBkg, int bkgBox = 100, double bkgGridStepSize = 20)
Compute multiscale blob mask using LoG filter maps.

static int **FindBlendedBlobs**(std::vector<*Source**> &deblendedBlobs, std::vector<*ImgPeak*> &deblendedPeaks, *Image* *img, double sigmaMin, double sigmaMax, double sigmaStep, int minBlobSize = 5, double thrFactor = 0, int kernelFactor = 1)
Compute blob blended components.

class `Caesar::Blob` : public TNamed
Subclassed by *Caesar::Region*, *Caesar::Source*

Public Functions

Blob()
Class constructor: initialize structures.

Blob(std::string name)
Parametric constructor.

Blob(std::vector<*Pixel**> const &pixels, std::string name = "")
Parametric constructor.

Blob(const *Blob* &blob)
Copy constructor.

virtual **~Blob**()
Class destructor: free allocated memory.

Blob &**operator=**(const *Blob* &blob)
Assignment Operator.

void **Copy**(TObject &blob) const
Copy method.

inline void **SetId**(int id)
Set blob id.

inline void **SetName**(std::string name)
Set blob name.

inline bool **IsAtEdge**()
Is blob at image edge.

inline void **SetEdgeFlag**(bool choice)
Set edge flag.

inline int **SetImageMetaData**(*ImgMetaData* *data, double crpix1Offset = 0, double crpix2Offset = 0)
Set image metadata.

inline *ImgMetaData* ***GetImageMetaData**()
get image metadata

inline bool **HasImageMetaData**()
Set image metadata.

inline long int **GetNPixels**()
Get number of pixels.

inline const PixelCollection &**GetPixels**() const
Get pixel collection.

inline void **SetPixels**(PixelCollection &pixels)
Set pixel collection.

int **AddPixel**(*Pixel* *pixel, bool makeCopy = false)
Add a pixel to collection.

inline bool **HasPixels**()
Has pixels stored?

inline *Pixel* ***GetPixel**(int index)
Get access to pixel by index.

inline std::vector<int> **GetSeedPixelIndexes**()
Get access to seed pixel indexes.

inline *Pixel* ***GetSeedPixel**()
Find and return largest-significance seed pixel.

inline void **MarkHaloPixels**(double ZThr)
Mark pixels with significance within given thresholds as halo pixels.

int **ComputeStats**(bool computeRobustStats = true, bool forceRecomputing = false, bool useParallelMedian = false)
Compute stats.

inline bool **HasStats**()
Has stats?

inline void **SetHasStats**(bool value)
Has stats?

int **ComputeMorphologyParams**()
Compute morphology parameters.

int **ComputeZernikeMoments**(int order = 6)
Compute Zernike moments.

inline bool **HasParameters**()
Has morphology parameters computed.

inline void **SetHasParameters**(bool value)
Set has morphology pars.

inline double **GetM1**()
Get 1st pixel moment (= mean)

inline void **SetM1**(double value)
Set 1st pixel moment (= mean)

inline double **GetM2**()
Get 2nd pixel moment.

inline void **SetM2**(double value)
Set 2nd pixel moment.

inline double **GetM3**()
Get 3rd pixel moment.

inline void **SetM3**(double value)
Set 3rd pixel moment.

inline double **GetM4**()
Get 4th pixel moment.

inline void **SetM4**(double value)
Set 4th pixel moment.

inline double **GetM1Curv**()
Get 1st pixel curvature moment (= mean)

inline void **SetM1Curv**(double value)
Set 1st pixel curvature moment (= mean)

inline double **GetM2Curv**()
Get 2nd pixel curvature moment.

inline void **SetM2Curv**(double value)
Set 2nd pixel curvature moment.

inline double **GetS**()
Get pixel flux sum.

inline void **SetS**(double value)
Set pixel flux sum.

inline double **GetSmax**() const
Get pixel flux max.

inline void **SetSmax**(double value)
Set pixel flux max.

inline double **GetSmin**() const
Get pixel flux min.

inline void **SetSmin**(double value)
Set pixel flux min.

inline double **GetSxx**()
Get pixel flux XX correlation.

inline void **SetSxx**(double value)
Set pixel flux XX correlation.

inline double **GetSyy**()
Get pixel flux YY correlation.

inline void **SetSyy**(double value)
Get pixel flux YY correlation.

inline double **GetSxy**()
Get pixel flux XY correlation.

inline void **SetSxy**(double value)
Set pixel flux XY correlation.

inline double **GetSx**()
Get pixel flux sum weighted by position x.

inline void **SetSx**(double value)
Set pixel flux sum weighted by position x.

inline double **GetSy**()
Get pixel flux sum weighted by position y.

inline void **SetSy**(double value)
Set pixel flux sum weighted by position y.

inline long int **GetSmaxPixId**()
Get id of maximum flux pixel.

inline void **SetSmaxPixId**(long int value)
Set id of maximum flux pixel.

inline long int **GetSminPixId**()
Get id of minimum flux pixel.

inline void **SetSminPixId**(long int value)
Set id of minimum flux pixel.

inline double **GetScurv**()
Get pixel sum of curvature.

inline void **SetScurv**(double value)
Set pixel sum of curvature.

inline double **GetSedge**()
Get pixel sum of edgeness.

inline void **SetSedge**(double value)
Set pixel sum of edgeness.

inline void **GetSourceRange**(float &xmin, float &xmax, float &ymin, float &ymax)
Get source x-y range.

inline long int **GetXmin**()
Get source pixel x_min.

inline long int **GetXmax**()
Get source pixel x_max.

inline long int **GetYmin**()
Get source pixel y_min.

inline long int **GetYmax**()
Get source pixel y_max.

inline void **SetSourceRange**(float xmin, float xmax, float ymin, float ymax)
Set source x-y range.

inline void **GetSourcePixelRange**(long int &ixmin, long int &ixmax, long int &iymin, long int &iymax)
Get source pixel coordinate range.

inline long int **GetIxMin**()
Get source pixel ix_min.

inline long int **GetIxMax**()
Get source pixel ix_max.

inline long int **GetIyMin**()
Get source pixel iy_min.

inline long int **GetIyMax**()
Get source pixel iy_max.

inline void **SetSourcePixelRange**(long int ixmin, long int ixmax, long int iymin, long int iymax)
Set source pixel coordinate range.

int **GetSampleStdDev**(double &sigmaX, double &sigmaY, double &covXY)
Get sample source standard deviations.

inline void **Print**()
Dump blob info.

Image ***GetImage**(ImgType mode, int pixMargin = 1, bool includeHaloPixels = false)
Generate an image from source pixel.

TH2D ***GetWCSHisto**(ImgType mode, int pixMargin = 1, int coordSyst = -1)
Generate an image from source pixel.

Image ***GetNestedBlobMask**(*Image* *sourceImage = 0, int pixMargin = 1, double nestedBlobMinScale = 1, double nestedBlobMaxScale = 3, double nestedBlobScaleStep = 1, double nestedBlobPeakZThr = 5, double nestedBlobPeakZMergeThr = 2.5, int NMinPix = 5, double nestedBlobThrFactor = 0, double nestedBlobKernFactor = 6)
Get blob mask.

inline bool **HasContours**()
Has contours.

inline std::vector<*Contour**> **GetContours**()
Return contours.

NB: Do not delete pointers.

inline *Contour* ***GetContour**(int index)
Return contour with index.

std::vector<*Contour**> **GetWCSContours**(WCS *wcs = 0, int coordSystem = -1, double pixOffset = 0, bool computePars = false, bool castCoordsToInt = false)
Return contours converted in WCS.

Contour ***GetWCSContour**(int index, WCS *wcs = 0, int coordSystem = -1, double pixOffset = 0, bool computePars = false, bool castCoordsToInt = false)
Return contour with index and convert to WCS.

inline void **AddContour**(*Contour* *aContour)
Add contour.

bool **IsPointOnContour**(double x, double y, double tol = 1)
Is point on contour?

inline double **GetBkgSum**()
Get bkg sum.

inline double **GetBkgRMSSum**()
Get bkg rms sum.

inline void **SetBkgSum**(double bkg)
Set bkg sum (USED BY SERIALIZER)

inline void **SetBkgRMSSum**(double rms)
Set bkg rms sum (USED BY SERIALIZER)

inline bool **HasBoxBkgInfo**()
Check if bkg info in a box around the blob has ben set.

inline void **SetHasBoxBkgInfo**(bool flag)
Set has box bkg info flag (USED BY SERIALIZER)

inline double **GetBoxBkg**()
Get bkg estimate in a box around the blob.

inline double **GetBoxBkgRMS**()
Get bkg rms estimate in a box around the blob.

inline void **SetBoxBkgInfo**(double bkg, double rms)
Set bkg and rms estimate in a box around the blob.

inline WCS ***GetWCS**(int coordSystem = -1)
Get WCS from stored metadata (TO BE DEPRECATED)

Get WCS from stored metadata

10.5 Source Finding & Fitting API

These classes provide source finding and fitting functionalities. Defined in `src/imgproc/SFinder.h`, `src/imgproc/SourceFitter.h`

class `Caesar::SFinder` : public TObject

Public Functions

SFinder()
Class constructor: initialize structures.

virtual **~SFinder**()
Class destructor: free allocated memory.

int **Run**()
Run source finder.

inline int **SetTileRead**(double xmin, double xmax, double ymin, double ymax)
Read only a tile from image.

class `Caesar::SourceFitter` : public TObject

Public Functions

SourceFitter()
Class constructor: initialize structures.

virtual **~SourceFitter**()
Class destructor: free allocated memory.

int **FitSource**(*Source* *source, SourceFitOptions &fitOptions)
Fit source with automated pars initialization.

int **FitSource**(*Source* *source, SourceFitOptions &fitOptions, std::vector<std::vector<double>> &fitPars_start)
Fit source starting from provided init pars.

inline SourceFitPars **GetFitPars**()
Get fit pars.

inline int **GetFitStatus**()
Get fit status.

double **Gaus2DMixtureFcn**(double *x, const double *p)
2D Gaussian mixture model used for the fit

double **Gaus2DFcn**(double *x, const double *p)
2D Gaussian model used for the fit

struct **SourceFitData**
Fit status enum flag.

Source fit data

class **Caesar::TaskData** : public TObject

Public Functions

TaskData()
Standard constructor.

TaskData(const *TaskData* &data)
Copy constructor.

virtual **~TaskData**()
Destructor.

TaskData &**operator=**(const *TaskData* &data)
Operator =.

void **Copy**(TObject &obj) const
Copy method.

void **ClearSources**()
Clear sources.

inline int **SetTile**(long int xmin, long int xmax, long int ymin, long int ymax)
Set tile.

inline void **AddNeighborInfo**(long int tid, long int wid)
Add neighbor info.

inline bool **IsInsideTaskTile**(double x, double y)
Check if point is inside task tile.

inline bool **IsTaskTileAdjacent**(*TaskData* *aTask)
Check if this task tile is adjacent to another given task.

inline bool **IsTaskTileOverlapping**(*TaskData* *aTask)
Check if this task tile is overlapping with another given task.

inline bool **IsTaskTileNeighbor**(*TaskData* *aTask)
Check if this task tile is neighbor (adjacent or overlapping) to another given task.

10.6 Image Segmentation API

These classes provide image segmentation algorithm. Defined in `src/imgproc/ChanVeseSegmenter.h`, `src/imgproc/SLICSegmenter.h` and `src/imgproc/LRACSegmenter.h`

class `Caesar::`**SLICSegmenter** : public TObject

Public Functions

SLICSegmenter()
Class constructor: initialize structures.

virtual **~SLICSegmenter**()
Class destructor: free allocated memory.

Public Static Functions

static int **FindSegmentation**(*SLICData* const &slicData, *SLICData* &segmSlicData, double SPMergingRegularization, bool use2ndNeighborsInSPMerging, int minMergedSP = 1, double SPMergingRatio = 0.3, double SPMergingMaxDissRatio = 1000, double SPMergingMaxDissRatio_2nd = 1.05, double SPMergingDissThreshold = 3, bool SPMergingIncludeSpatialPars = true, bool SPMergingUseRobustPars = false, bool SPMergingUseCurvDist = true)
Merge the superpixels (main methods)

class `Caesar::`**ChanVeseSegmenter** : public TObject

Public Functions

ChanVeseSegmenter()
Class constructor.

~ChanVeseSegmenter()
Class destructor.

Public Static Functions

static *Image* *FindSegmentation(*Image* *img, *Image* *initSegmImg = 0, bool returnContourImg = false, double dt = 0.1, double h = 1, double lambda1 = 1.0, double lambda2 = 2.0, double mu = 0.5, double nu = 0, double p = 1, long int nIterations = 1000, double tol = 1.e-2, long int nIterationsInner = 1, long int nIterationsReInit = 10)

Find the ChanVese segmentation of input image.

static void SetCheckerBoardLevelSet(TMatrixD *M, double square_size = 10)

Set level set to checker board model.

static void SetCircleLevelSet(TMatrixD *M)

Set level set to circle model.

struct CVdata

struct CVsetup

class Caesar::LRACSegmenter : public TObject

Public Functions

LRACSegmenter()

Class constructor.

~LRACSegmenter()

Class destructor.

Public Static Functions

static *Image* *FindSegmentation(*Image* *img, *Image* *initSegmImg, int iterations = 1000, double lambda = 0.1, double radius = 1, double eps = 1.e-2)

Find the ChanVese segmentation of input image.

CHAPTER

ELEVEN

SOURCE API

This section reports the source API.

CHAPTER

TWELVE

SOURCE IO API

This section reports the source exporter and serialization API.

12.1 Source Exporter API

These classes provide source exporting (ds9, ascii, ...) functionalities Defined in `src/imgproc/SourceExporter.h`

class `Caesar::SourceExporter` : public TObject

Public Functions

SourceExporter()
Class constructor: initialize structures.

virtual **~SourceExporter**()
Class destructor: free allocated memory.

Public Static Functions

static int **WriteToAscii**(std::string filename, const std::vector<*Source**> &sources, bool dumpNestedSourceInfo = true, int wcsType = eJ2000, WCS *wcs = 0, bool writeAdditionalSourceInfo = false, bool convertBrightnessToFlux = true, char delimiter = '\t')
Write source collection to ascii file.

static const std::vector<std::string> **SourceToAscii**(*Source* *source, bool dumpNestedSourceInfo = true, int wcsType = eJ2000, WCS *wcs = 0, bool writeAdditionalSourceInfo = false, bool convertBrightnessToFlux = true, char delimiter = '\t')
Get source ascii string.

static int **WriteComponentsToAscii**(std::string filename, const std::vector<*Source**> &sources, bool dumpNestedSourceInfo = true, int wcsType = eJ2000, WCS *wcs = 0, bool writeAdditionalSourceInfo = false, bool convertBrightnessToFlux = true, char delimiter = '\t')
Write ascii file from source component collection.

static const std::vector<std::string> **SourceComponentsToAscii**(*Source* *source, bool dumpNestedSourceInfo = true, int wcsType = eJ2000, WCS *wcs = 0, bool writeAdditionalSourceInfo = false, bool convertBrightnessToFlux = true, char delimiter = '\t')

Get source component ascii string.

static int **WriteToFullJson**(std::string filename, const std::vector<*Source**> &sources, bool dumpNestedSourceInfo = true, int wcsType = eJ2000, WCS *wcs = 0, bool convertBrightnessToFlux = true)

Write json file from source collection.

static int **FillJsonMetadata**(Json::Value &json, *ImgMetaData* *metadata)

Fill json metadata fields.

static int **FillJsonSource**(Json::Value &json, *Source* *source, bool dumpNestedSourceInfo = true, WCS *wcs = 0, int index = -1, int parent_index = -1, int parent_island_index = -1, bool convertBrightnessToFlux = true)

Fill json source island fields.

static int **FillJsonSourceIsland**(Json::Value &json, *Source* *source, WCS *wcs = 0, int parent_index = -1, int parent_island_index = -1, bool convertBrightnessToFlux = true)

Fill json source island fields.

static int **FillJsonSourceComponents**(Json::Value &json, *Source* *source, WCS *wcs = 0, bool convertBrightnessToFlux = true)

Fill json source components fields.

static int **WriteToJson**(std::string filename, const std::vector<*Source**> &sources, bool dumpNestedSourceInfo = true, int wcsType = eJ2000, WCS *wcs = 0)

Write json file from source collection.

static int **SourceToJson**(std::vector<Json::Value> &jsonValues, *Source* *source, bool dumpNestedSourceInfo = true, int wcsType = eJ2000, WCS *wcs = 0)

Get source json object.

static int **WriteComponentsToJson**(std::string filename, const std::vector<*Source**> &sources, bool dumpNestedSourceInfo = true, int wcsType = eJ2000, WCS *wcs = 0)

Write json file from source component collection.

static int **SourceComponentsToJson**(std::vector<Json::Value> &jsonValues, *Source* *source, bool dumpNestedSourceInfo = true, int wcsType = eJ2000, WCS *wcs = 0)

Get source component json object.

static int **WriteToROOT**(std::string filename, const std::vector<*Source**> &sources, bool dumpNestedSourceInfo = true, int wcsType = eJ2000, WCS *wcs = 0, bool writeAdditionalSourceInfo = false)

Write ROOT file with TTree from source collection.

static int **WriteComponentsToROOT**(std::string filename, const std::vector<*Source**> &sources, bool dumpNestedSourceInfo = true, int wcsType = eJ2000, WCS *wcs = 0, bool writeAdditionalSourceInfo = false)

Write ROOT file with TTree from source fit component collection.

static int **WriteToDS9**(std::string filename, const std::vector<*Source**> &sources, bool convertDS9RegionsToWCS = false, int ds9WCSType = eJ2000, int ds9RegionFormat = ePolygonRegion, WCS *wcs = 0, std::string ds9RegionColor = "red")

Write DS9 regions from source collection.

static int **WriteComponentsToDS9**(std::string filename, const std::vector<*Source**> &sources, bool convertDS9RegionsToWCS = false, int ds9WCSType = eJ2000, WCS *wcs = 0, std::string ds9RegionColor = "red")
Write DS9 regions for source fitted components.

static const std::string **SourceToDS9Region**(*Source* *source, bool dumpNestedSourceInfo = false, bool convertToWCS = false, WCS *wcs = 0, int coordSystem = -1)
Get DS9 region info.

static const std::string **SourceToDS9EllipseRegion**(*Source* *source, bool dumpNestedSourceInfo = false)
Get DS9 ellipse info.

static const std::string **SourceToDS9FittedEllipseRegion**(*Source* *source, bool useFWHM = true, bool dumpNestedSourceInfo = false, bool convertToWCS = false, WCS *wcs = 0, int coordSystem = -1)
Get DS9 fitted ellipse info.

12.2 Source Serializer API

These classes provide source serialization (to/from Google protobuf) functionalities for distributed processing. Defined in `src/utils/Serializer.h`

class `Caesar::`**Serializer** : public TObject

Public Functions

Serializer()
Class constructor: initialize structures.

~Serializer()
Class destructor: free allocated memory.

Public Static Functions

static int **EncodeMetaDataToProtobuf**(CaesarPB::ImgMetaData &metadata_pb, *ImgMetaData* *metadata)
Serialize image metadata to protobuf.

static int **EncodePointToProtobuf**(CaesarPB::Point &point_pb, TVector2 &point)
Serialize TVector2 point object to protobuf.

static int **EncodeContourToProtobuf**(CaesarPB::Contour &contour_pb, *Contour* *contour)
Serialize contour object to protobuf.

static int **EncodePixelToProtobuf**(CaesarPB::Pixel &pixel_pb, *Pixel* *pixel)
Serialize image pixel object to protobuf.

static int **EncodeBlobToProtobuf**(CaesarPB::Blob &blob_pb, *Source* *source)
Serialize blob object to protobuf.

static int **EncodeSourceToProtobuf**(CaesarPB::Source &source_pb, *Source* *source)
Serialize source object to protobuf.

static int **EncodeSourceCollectionToProtobuf**(CaesarPB::SourceCollection &sources_pb, const std::vector<*Source**> &sources)
Serialize source collection to protobuf.

static int **SourceToBuffer**(*SBuffer* &buffer, *Source* *source)
Serialize source object to *SBuffer* object.

static int **EncodeSourceComponentParsToProtobuf**(CaesarPB::SourceComponentPars *sourceCompPars_pb, SourceComponentPars &sourceCompPars)
Serialize source fit component pars to protobuf.

static int **EncodeSourceFitParsToProtobuf**(CaesarPB::SourceFitPars &sourceFitPars_pb, SourceFitPars &sourceFitPars)
Serialize source fit pars to protobuf.

static int **EncodeSpectralIndexDataToProtobuf**(CaesarPB::SpectralIndexData &spectralIndexData_pb, const SpectralIndexData &spectralIndexData)
Serialize spectral index data to protobuf.

static int **EncodeSpectralIndexDataCollectionToProtobuf**(CaesarPB::SpectralIndexDataCollection &spectralIndexDataCollection_pb, const std::vector<SpectralIndexData> &spectralIndexDataCollection)
Serialize spectral index collection to protobuf.

static int **EncodeAstroObjectToProtobuf**(CaesarPB::AstroObject &astroObject_pb, const AstroObject &astroObject)
Serialize astronomical object to protobuf.

static int **EncodeAstroObjectCollectionToProtobuf**(CaesarPB::AstroObjectCollection &astroObjectCollection_pb, const std::vector<AstroObject> &astroObjectCollection)
Serialize astro object collection to protobuf.

static int **EncodeProtobufToMetaData**(*ImgMetaData* &metadata, const CaesarPB::ImgMetaData &metadata_pb)
Deserialize protobuf metadata object.

static int **EncodeProtobufToPoint**(TVector2 &point, const CaesarPB::Point &point_pb)
Deserialize protobuf point object.

static int **EncodeProtobufToContour**(*Contour* &contour, const CaesarPB::Contour &contour_pb)
Deserialize protobuf contour object.

static int **EncodeProtobufToPixel**(*Pixel* &pixel, const CaesarPB::Pixel &pixel_pb)
Deserialize protobuf pixel object.

static int **EncodeProtobufToBlob**(*Source* &source, const CaesarPB::Blob &blob_pb)
Deserialize protobuf blob object.

static int **EncodeProtobufToSource**(*Source* &source, const CaesarPB::Source &source_pb)
Deserialize protobuf source object.

static int **BufferToSource**(*Source* &source, *SBuffer* &buffer)
Convert *SBuffer* to *Source* object.

static int **EncodeProtobufToSourceComponentPars**(SourceComponentPars &sourceComponentPars, const CaesarPB::SourceComponentPars &sourceComponentPars_pb)
Deserialize protobuf source fit component pars object.

static int **EncodeProtobufToSourceFitPars**(SourceFitPars &sourceFitPars, const CaesarPB::SourceFitPars &sourceFitPars_pb)
Deserialize protobuf source fit pars object.

static int **EncodeProtobufToSpectralIndexData**(SpectralIndexData &spectralIndexData, const CaesarPB::SpectralIndexData &spectralIndexData_pb)
Deserialize protobuf spectral index data object.

static int **EncodeProtobufToSpectralIndexDataCollection**(std::vector<SpectralIndexData> &spectralIndexDataCollection, const CaesarPB::SpectralIndexDataCollection &spectralIndexDataCollection_pb)
Deserialize protobuf spectral index data collection object.

static int **EncodeProtobufToAstroObject**(AstroObject &astroObject, const CaesarPB::AstroObject &astroObject_pb)
Deserialize protobuf astro object object.

static int **EncodeProtobufToAstroObjectCollection**(std::vector<AstroObject> &astroObjectCollection, const CaesarPB::AstroObjectCollection &astroObjectCollection_pb)
Deserialize protobuf astro object collection object.

static int **EncodeTaskDataToProtobuf**(CaesarPB::TaskData &taskData_pb, *TaskData* *taskData)
Serialize task data object to protobuf.

static int **TaskDataToBuffer**(*SBuffer* &buffer, *TaskData* *taskData)
Serialize task data to *SBuffer*.

static char ***TaskDataToCharArray**(long int &buffer_size, *TaskData* *taskData)
Serialize task data object to char array.

static int **EncodeTaskDataCollectionToProtobuf**(CaesarPB::TaskDataCollection &taskDataCollection_pb, const std::vector<*TaskData**> &taskDataCollection)
Serialize task data collection to protobuf.

static int **TaskDataCollectionToBuffer**(*SBuffer* &buffer, const std::vector<*TaskData**> &taskDataCollection)
Convert task data collection to *SBuffer*.

static char ***TaskDataCollectionToCharArray**(long int &buffer_size, const std::vector<*TaskData**> &taskDataCollection)
Serialize task data collection to char array.

static int **EncodeProtobufToTaskData**(*TaskData* &taskData, const CaesarPB::TaskData &taskData_pb)
Deserialize task data protobuf object.

static int **BufferToTaskData**(*TaskData* &taskData, *SBuffer* &buffer)
Convert *SBuffer* to task data.

static int **CharArrayToTaskData**(*TaskData* &taskData, char *buffer, long int buffer_size)
Convert char array to task data.

static int **EncodeProtobufToTaskDataCollection**(std::vector<*TaskData**> &taskDataCollection, const CaesarPB::TaskDataCollection &taskDataCollection_pb, bool isTaskCollectionPreAllocated = false)
Deserialize task data collection protobuf object.

static int **BufferToTaskDataCollection**(std::vector<*TaskData**> &taskDataCollection, *SBuffer* &buffer, bool isTaskCollectionPreAllocated = false)

Convert *SBuffer* to task data collection.

static int **CharArrayToTaskDataCollection**(std::vector<*TaskData**> &taskDataCollection, char *buffer, long int buffer_size, bool isTaskCollectionPreAllocated = false)

Convert char array to task data collection.

static char ***SourceCollectionToCharArray**(long int &buffer_size, const std::vector<*Source**> &sourceCollection)

Convert source collection to char array.

static int **EncodeProtobufToSourceCollection**(std::vector<*Source**> &sources, const CaesarPB::SourceCollection &sources_pb, bool isSourceCollectionPreAllocated = false)

Deserialize source collection protobuf object.

static int **CharArrayToSourceCollection**(std::vector<*Source**> &sources, char *buffer, long int buffer_size, bool isSourceCollectionPreAllocated = false)

Convert char array to source collection.

class `Caesar::SBuffer` : public TObject

Public Functions

inline **SBuffer**()

SBuffer constructor.

inline virtual **~SBuffer**()

SBuffer destructor.

CHAPTER

THIRTEEN

CONFIGURATION API

This section reports the configuration functions API.

13.1 Configuration Options

These classes define configuration options and methods to parse configuration files. Defined in `src/utils/ConfigParser.h` and `src/utils/Option.h`

class `Caesar::ConfigParser` : public TObject

Public Functions

int **Parse**(std::string filename = "")
Read the config file, parse and set info to be used by other classes.

inline void **PrintOptions**()
Print registered options.

template<typename **T**>
inline std::shared_ptr<*Option*<*T*>> **GetOption**(std::string name)
Register options.

inline bool **HasOption**(std::string name)
Check registered options.

inline int **SetOptionFromString**(std::string name, std::string stringified_value)
Set option from string.

template<typename **T**>
inline int **GetOptionValue**(std::string name, *T* &value)
Get option value.

inline TTree ***GetConfigTree**(std::string treeName = "CfgInfo")
Get config option tree.

template<typename **T**>

class `Caesar::Option` : public Caesar::OptionBase

Public Functions

inline **Option**()
Default Class constructor (needed by ROOT IO)

inline **Option**(std::string name, *T* defaultValue, *T* minValue, *T* maxValue)
Class constructor.

inline virtual **~Option**()
Class destructor.

inline virtual int **SetValueFromStream**(std::stringstream &sstream)
Set option value from stringstream.

inline virtual int **SetValueFromString**(std::string const &s)
Set option value from string.

inline int **SetValue**(*T* value)
Set option value.

inline *T* **GetValue**()
Get option value.

inline *T* **GetDefaultValue**()
Get default value.

inline virtual int **GetJson**(Json::Value &jsonObj)
Get json object from option.

inline virtual int **GetJsonString**(std::string &jsonString, bool isMinified = true)
Get json string.

inline bool **CheckRange**(*T* &value)
Check given value against registered range (non sense for bool, string/char)?

inline virtual void **Print**()
Print option.

inline virtual int **AddBranch**(TTree *tree)
Add a TBranch to input TTree.

CHAPTER

FOURTEEN

LOGGING API

This section reports the logging API.

14.1 Logging

This class provide logging functionalities. Defined in `src/utils/Logger.h`

class `Caesar::Logger` : public TObject
Subclassed by *Caesar::ConsoleLogger*, *Caesar::FileLogger*, *Caesar::SysLogger*

Public Functions

inline **Logger**(std::string level = "OFF", std::string tag = "logger")
Constructor.

inline virtual **~Logger**()
Destructor.

virtual int **Init**() = 0
Returns log4cxx log level from log4tango log level.

Pure abstract method to initialize logger

inline virtual std::string **GetHost**() const
Get machine host name.

inline virtual void **Log**(const std::string sLevel, const std::string msg, const std::string msg_prefix = "")
Log a message (optionally prepended by a prefix) at given log level in string format.

inline virtual void **SetLogLevel**(const std::string sLevel)
Set logger log level threshold.

class `Caesar::LoggerManager` : public TObject

Public Types

enum **LogTarget**
Define log target enumerations.

Values:

enumerator **eSysLog**

enumerator **eConsole**

enumerator **eFile**

Public Static Functions

static inline *LoggerManager* &**Instance**()
Returns unique instance of logger.

static inline int **CreateSysLogger**(const std::string level, const std::string tag = "syslogger", const std::string facility = "local6")
Create a syslog logger.

static inline int **CreateConsoleLogger**(const std::string level, const std::string tag = "consolelogger", const std::string target = "System.out")
Create a console logger.

class `Caesar::ConsoleLogger` : public Caesar::*Logger*

Public Functions

inline **ConsoleLogger**(const std::string level, const std::string tag, const std::string target)
Console logger constructor.

inline virtual **~ConsoleLogger**()
Destructor.

inline virtual int **Init**()
Initialize console logger.

class `Caesar::FileLogger` : public Caesar::*Logger*

Public Functions

inline **FileLogger**(const std::string level, const std::string tag, const std::string filename, bool appendFlag, const std::string maxFileSize, int maxBackupFiles)
File logger constructor.

inline virtual **~FileLogger**()
Destructor.

inline virtual int **Init**()
Initialize file logger.

class `Caesar::SysLogger` : public Caesar::*Logger*

Public Functions

inline **SysLogger**(const std::string level, const std::string tag, const std::string facility)
Syslog logger constructor.

inline virtual **~SysLogger**()
Destructor.

inline virtual int **Init**()
Initialize syslog logger.

inline virtual int **GetFacilityCode**(const std::string syslog_facility) const
Returns syslog facility code.

class Caesar::**ScopedLogger**

Public Functions

inline **ScopedLogger**(std::string level, std::string prefix = "", std::string device_name = "")
Scoped logger constructor.

inline **~ScopedLogger**()
Destructor.

inline std::stringstream &**stream**()
Returns message string stream to be logged.

inline std::string Caesar::**getClassName**(std::string fullFuncName, std::string funcName)
Returns Tango device name in which log is emitted.

Returns class name in which log is emitted

inline std::string Caesar::**getClassNamePrefix**(std::string fullFuncName, std::string funcName)
Returns class name prefix in which log is emitted.

__CLASS__
Shortcut macro to get actual class name.

__CLASS_PREFIX__
Shortcut macro to get actual class name prefix.

CAESAR_LOG(DeviceName, Level, What)
Shortcut macro to emit a log message with desired log level inside a given device.

Parameters

- **DeviceName** – - Tango device server name
- **Level** – - Message log level
- **What** – - Message stream to be sent

LOG(Level, What)
Shortcut macro to emit a log message with desired log level.

Parameters

- **Level** – - Message log level
- **What** – - Message stream to be sent

INFO_LOG(What)
Shortcut macro to emit a log message with INFO log level.

Parameters

- **What** – - Message stream to be sent

WARN_LOG(What)
Shortcut macro to emit a log message with WARN log level.

Parameters

- **What** – - Message stream to be sent

ERROR_LOG(What)
Shortcut macro to emit a log message with ERROR log level.

Parameters

- **What** – - Message stream to be sent

FATAL_LOG(What)
Shortcut macro to emit a log message with FATAL log level.

Parameters

- **What** – - Message stream to be sent

DEBUG_LOG(What)
Shortcut macro to emit a log message with DEBUG log level.

Parameters

- **What** – - Message stream to be sent

CHAPTER

FIFTEEN

UTILITIES API

This section reports the utility functions API.

15.1 Astronomy Utilities

These classes define astronomical utility methods. Defined in `src/utils/AstroUtils.h`

class `Caesar::AstroUtils` : public TObject

Public Functions

AstroUtils()
Class constructor: initialize structures.

virtual **~AstroUtils**()
Class destructor: free allocated memory.

Public Static Functions

static int **GetIAUCoords**(std::string &iau, const std::string &s)
Get IAU naming from WCS string coordinates.

static int **WCSToPixelCoords**(double &ix, double &iy, WCS *wcs, double xpos, double ypos)
Get image coordinates corresponding to WCS coordinates.

static int **PixelToWCSCoords**(double &xpos, double &ypos, WCS *wcs, double ix, double iy)
Get WCS coordinates corresponding to image coordinates.

static int **PixelToWCSStrCoords**(std::string &wcs_str, WCS *wcs, double ix, double iy, int max_str_length = 4096)
Get WCS coordinates in string format corresponding to image coordinates.

static int **PixelToWCSCoords**(Caesar::*Image* *image, WCS *wcs, double ix, double iy, double &xpos, double &ypos, bool useImageCoords = true)
Get WCS coordinates corresponding to image coordinates.

static int **PixelToWCSStrCoords**(Caesar::*Image* *image, WCS *wcs, double ix, double iy, std::string &wcs_str, bool useImageCoords = true, int max_str_length = 4096)
Get WCS coordinates in string format corresponding to image coordinates.

static int **PixelToWCSCoords**(Caesar::*Image* *image, double ix, double iy, double &xpos, double &ypos, int coordSystem = -1, bool useImageCoords = true)
Get WCS coordinates corresponding to image coordinates.

static int **PixelToWCSStrCoords**(Caesar::*Image* *image, double ix, double iy, std::string &wcs_pos, int coordSystem = -1, bool useImageCoords = true, int max_str_length = 4096)
Get WCS coordinates in string format corresponding to image coordinates.

static inline double **GetBeamArea**(double Bmaj, double Bmin)
Get beam area from BMAJ, BMIN.

static inline double **GetBeamAreaInPixels**(double Bmaj, double Bmin, double dX, double dY)
Get beam area in pixels given Bmaj, Bmin and pixel sizes (dx, dy) in deg.

static inline int **GetBeamWidthInPixels**(double Bmaj, double Bmin, double dX, double dY)
Get beam width in pixels given Bmaj, Bmin and pixel sizes (dx, dy) in deg.

static *Contour* ***PixelToWCSContour**(*Contour* *contour, WCS *wcs, double pixOffset = 0, bool castCoordsToInt = false)
Convert contour from pixel coordinates to sky coordinates.

static int **PixelToWCSContours**(std::vector<*Contour**> &contours_wcs, std::vector<*Contour**> const &contours, WCS *wcs, double pixOffset = 0, bool castCoordsToInt = false)
Convert contour list from pixel coordinates to sky coordinates.

static TEllipse ***PixelToWCSEllipse**(TEllipse *ellipse, WCS *wcs, double pixOffset = 0)
Convert ellipse from pixel coordinates to sky coordinates.

static TEllipse ***PixelToWCSEllipseSimple**(TEllipse *ellipse, WCS *wcs, double pixOffset = 0)
Convert ellipse from pixel coordinates to sky coordinates neglecting sky projection and assuming Euclidean distances.

static TEllipse ***GetBeamDeconvolvedEllipse**(TEllipse *ellipse, TEllipse *beam)
Compute ellipse deconvolved from ellipse beam.

static int **GetBeamDeconvolvedEllipsePars**(double &bmaj_deconv, double &bmin_deconv, double &bpa_deconv, double bmaj, double bmin, double bpa, double bmaj_beam, double bmin_beam, double bpa_beam)
Compute ellipse pars deconvolved from ellipse beam.

static double **GetWCSPointDist_Haversine**(double ra1, double dec1, double ra2, double dec2)
Compute WCS beam ellipse pars from pixel ellipse pars.

Get distance (in degrees) between two points on the sky using Haversine formula

static double **GetWCSPointDist_Vincenty**(double ra1, double dec1, double ra2, double dec2)
Get distance (in degrees) between two points on the sky using Vincenty formula.

static double **GetWCSPointBearing**(double ra1, double dec1, double ra2, double dec2)
Get point bearing (in degrees) between two points on the sky.

static std::string **EllipseToDS9Region**(TEllipse *ellipse, std::string text = "", std::string color = "white", std::vector<std::string> tags = {}, double pixOffset = 0)
Convert ellipse to DS9 format.

static std::string **ContourToDS9Region**(*Contour* *contour, std::string text = "", std::string color = "white", std::vector<std::string> tags = {}, bool useImageCoords = true, double pixOffset = 0)
Convert contour to DS9 format.

static std::string **GetDS9WCSTypeHeader**(int coordSys)
Returns DS9 WCS type header from flag.

15.2 Code Utilities

These classes define code utility methods. Defined in `src/utils/CodeUtils.h`

class `Caesar::CodeUtils` : public TObject

Public Functions

CodeUtils()
Class constructor: initialize structures.

virtual **~CodeUtils**()
Class destructor: free allocated memory.

template<>
inline std::vector<bool> **StringVecToTypedVec**(const std::vector<std::string> &vec)
Convert string vector to bool vector (specialization of general method)

Public Static Functions

static inline std::string **GenerateUUID**()
Generate uuid string.

static inline std::string **GetStringCodeFromIntegers**(std::vector<int> ids, size_t ndigits)
Convert json to string (DEPRECATED JSONCPP API)

Concatenate integers and create string code given digits

static inline long int **GetCodeFromIntegers**(std::vector<int> ids, size_t ndigits)
Concatenate integers and create int code given digits.

static inline std::vector<std::string> **SplitStringInEqualParts**(const std::string &str, int splitLength)
Split a string in equal parts.

static inline int **DecodeIntCodes**(std::vector<int> &ids, long int code, size_t ndigits)
Get integer codes from string codes.

static inline int **JsonToString**(std::string &jsonString, Json::Value &jsonObj, bool isMinified = true)
Convert json to string (NEW JSONCPP API)

static inline int **StringToJson**(Json::Value &root, std::string &jsonString)
Convert string to json.

template<typename **T**>
static inline int **FindJsonValue**(int &pos, Json::Value &root, *T* value, std::string key = "name")
Find json option in array by name.

template<class **T**>
static inline bool **FindItem**(std::vector<*T*> &v, *T* itemValue, int &pos)
Find item in a vector and returns item position.

template<class **T**, typename **K**>
static inline void **DeleteItems**(std::vector<*T*> &data, const std::vector<*K*> &deleteIndices)
Delete selected items from a vector.

template<class **T**, typename **K**>
static inline void **DeletePtrItems**(std::vector<*T*> &data, const std::vector<*K*> &deleteIndices)
Delete selected items from a vector.

template<class **T**>
static inline void **DeletePtrCollection**(std::vector<*T**> &data)
Delete object pointer collection.

template<class **T**>
static inline void **DeletePtrCollection**(std::initializer_list<*T**> data)
Delete object pointer collection.

template<class **T**>
static inline void **DeletePtr**(*T* *data)
Delete object pointer.

template<class **T**>
static inline void **reorder**(std::vector<*T*> &unordered, std::vector<size_t> const &index_map, std::vector<*T*> &ordered)
Reorder vector.

template<class **T**>
static inline void **sort**(std::vector<*T*> &unsorted, std::vector<*T*> &sorted, std::vector<size_t> &index_map)
Sort vector.

template<class **T**>
static inline void **sort_descending**(std::vector<*T*> &unsorted, std::vector<*T*> &sorted, std::vector<size_t> &index_map)
Sort vector in descending order.

template<class **Iterator**, class **Comparator**>
static inline IndexPairs **FindIntersectionIndexes**(*Iterator* first1, *Iterator* last1, *Iterator* first2, *Iterator* last2, *Comparator* comp, bool sorted = false)
Find index of equal elements in two vectors.

template<class **Iterator**>
static inline *Iterator* **random_unique**(*Iterator* begin, *Iterator* end, size_t num_random)
Extract a subset of random index without repetitions from a container.

template<typename **T**>
static inline int **ExtractVectorRandomSample**(std::vector<*T*> &sample_data, const std::vector<*T*> &data, long int n = -1, bool repeate = true)
Extract sample from data vector with given sample size, with/without repetitions and uniform weights.

template<typename **T**>
static inline int **ExtractVectorRandomSamples**(std::vector<std::vector<*T*>> &data_samples, const std::vector<*T*> &data, int nSamples, long int n = -1, bool repeate = true)
Extract a number of random samples from data vector with given sample size, with/without repetitions and uniform weights.

template<class **T**, class **Comparator**>
static inline *T* **FindCumulativeSumFractionThr**(std::vector<*T*> &data, *Comparator* comp, double thr, bool sorted = false)
Find vector index at which the cumulative sum is smaller then given value (comparator version)

template<class **T**>
static inline *T* **FindCumulativeSumFractionThr**(std::vector<*T*> &data, double thr, bool sorted = false)
Find vector index at which the cumulative sum is smaller then given value.

template<class **InputIt**, class **T** = typename std::iterator_traits<*InputIt*>::value_type>
static inline *T* **FindVectorMode**(*InputIt* begin, *InputIt* end, int &nmodes)
Find vector index at which the cumulative sum is smaller then given value.

static inline void **StringFindAndReplace**(std::string &str, const std::string &oldstr, const std::string &newstr)
String find and replace.

static inline void **RemovePatternInString**(std::string &str, const std::string pattern)
Remove pattern in string.

static inline std::string **ExtractSubString**(const std::string &s, const std::string &pattern, bool extractleft = true)
Extract substring.

static inline std::vector<std::string> **SplitStringOnWhitespaces**(const std::string &s)
Split string on whitespaces.

static inline std::vector<std::string> **SplitStringOnPattern**(const std::string &s, char delim)
Split string on pattern.

static inline std::string **JoinStringVec**(const std::vector<std::string> &vec, std::string delim)
Join string from string vector using delimiter.

template<typename **T**>
static inline std::vector<*T*> **StringVecToTypedVec**(const std::vector<std::string> &vec)
Convert string vector to typed vector.

static inline bool **HasPatternInString**(std::string str, std::string pattern)
Find pattern in string.

static inline int **StripBlankSpaces**(std::string &s)
Strip blank spaces from string.

template<typename **Iter**>
static inline std::string **JoinCollection**(*Iter* begin, *Iter* end, std::string separator = "")
Collapse a collection in a string (equivalent of python join)

template<typename **T**>
static inline std::string **JoinVec**(const std::vector<*T*> &data, std::string separator = "")
Join vectors.

static inline bool **GetVecLogicalOr**(const std::vector<bool> &v)
Perform logical OR among all vector elements.

static inline bool **GetVecLogicalAnd**(const std::vector<bool> &v)
Perform logical AND among all vector elements.

static inline bool **AreEqualStringNoCase**(std::string str1, std::string str2)
Compare string case insensitive.

static inline std::string **ExtractFileNameFromPath**(const std::string &s, bool strip_extension = false)
Extract filename from path.

template<typename **T**, typename **K**>
static inline bool **HasMapKey**(std::map<*T*, *K*> const &m, *T* key)
Check if map has key.

template<class **T**>

struct **descending_index_cmp**

template<class **T**>

struct **index_cmp**
Order vectors and get ordering index.

15.3 Graphics Utilities

These classes define image graphics utility methods. Defined in `src/utils/GraphicsUtils.h` and `src/utils/ImgUtils.h`

class `Caesar::GraphicsUtils` : public TObject

Public Functions

GraphicsUtils()
Class constructor: initialize structures.

virtual **~GraphicsUtils**()
Class destructor: free allocated memory.

Public Static Functions

static void **SetPalette**(int paletteStyle, int ncolors = 999)
Get palette code.

static int **SetThermalPalette**(int ncolors = 999)
Set thermal palette.

static int **SetHotColdPalette**(int ncolors = 999)
Set hot-to-cold palette.

static int **SetColdHotPalette**(int ncolors = 999)
Set cold-to-hot palette.

static int **SetBWPalette**(int ncolors = 999)
Setblack & white palette.

static int **SetWCSAxis**(*Image* *img, TGaxis &xaxis, TGaxis &yaxis, int coordSystem = -1, bool useImageCoords = true)
Set WCS axis.

static int **SetWCSProjGrid**(*Image* *img, std::vector<TPolyLine> &gridx, std::vector<TPolyLine> &gridy, int coordSystem)
Set WCS proj grid.

static *Image* ***FindImageFromPad**()
Retrieve image from pad.

static int **PadUpdater**()
Update pad.

static int **PadUpdater_PhysCoords**()
Update pad (version without image coordinates)

static int **UpdateGAxis**(bool useImageCoords = true)
Update ROOT canvas gaxis.

class `Caesar::ImgUtils` : public TObject

Public Functions

inline **ImgUtils**()
Constructor.

inline virtual **~ImgUtils**()
Destructor.

Public Static Functions

static *Image* ***GetCircleLevelSetImage**(long int nX, long int nY, double f = 0.1)
Get image with single circle level set.

static *Image* ***GetCheckerBoardLevelSetImage**(long int nX, long int nY, double f = 0.1)
Get image with checker board level set.

15.4 Math & Stats Utilities

This class defines math and stats utility methods. Defined in `src/utils/EllipseUtils.h`, `src/imgproc/Graph.h`, `src/imgproc/ZernikeMoments.h`, `src/utils/MathUtils.h` and `src/utils/StatsUtils.h`

class `Caesar::`**EllipseUtils** : public TObject

Public Functions

EllipseUtils()
Class constructor: initialize structures.

virtual **~EllipseUtils**()
Class destructor: free allocated memory.

Public Static Functions

static double **ellipse_ellipse_overlap**(double PHI_1, double A1, double B1, double H1, double K1, double PHI_2, double A2, double B2, double H2, double K2, double X[4], double Y[4], int *NROOTS, int *rtnCode, int choice)
Compute ellipse overlap.

Compute overlap area polygon

class `Caesar::`**MathUtils** : public TObject

Public Functions

MathUtils()
Class constructor: initialize structures.

virtual **~MathUtils**()
Class destructor: free allocated memory.

Public Static Functions

static int **Compute2DGrid**(std::vector<long int> &ix_min, std::vector<long int> &ix_max, std::vector<long int> &iy_min, std::vector<long int> &iy_max, long int Nx, long int Ny, long int boxSizeX, long int boxSizeY, float gridStepSizeX, float gridStepSizeY)
Return 2D grid partition given Nx x Ny pixels and box sizes.

static int **Compute2DFloatGrid**(std::vector<float> &ix_min, std::vector<float> &ix_max, std::vector<float> &iy_min, std::vector<float> &iy_max, float xmin, float xmax, float xstep, float ymin, float ymax, float ystep)
Return 2D grid partition given x & y range and step sizes.

static int **FindGrid2DAxisBin**(float x, long int nx, float xmin, float xmax, float xstep)
Return 2D grid axis bin given grid parameters.

static long int **FindGrid2DBin**(float x, float y, long int nx, float xmin, float xmax, float xstep, long int ny, float ymin, float ymax, float ystep)
Return 2D grid global bin given grid parameters.

static int **BiLinearInterpolation**(std::vector<double> const &sampled_gridX, std::vector<double> const &sampled_gridY, std::vector<double> const &sampledZ, std::vector<double> const &interp_gridX, std::vector<double> const &interp_gridY, std::vector<double> &interpZ)
Perform bilinear interpolation on regular grid.

static cv::Mat **GetConvolution**(cv::Mat I, cv::Mat kernel)
Get opencv mat convolutions.

static cv::Mat **GetConvolution2**(cv::Mat I, cv::Mat kernel)
Get convolution (2nd version)

static cv::Mat **GetATrousConvolution**(cv::Mat I, cv::Mat kernel, int scale)
Get atrous convolution.

static int **GetMirrorIndex**(int index, int N)
Get mirror index.

static inline double **GetMatrixTrace**(TMatrixD *T)
Compute matrix trace.

static std::vector<std::complex<double>> **DFTShifted**(std::vector<std::complex<double>> data, int n)
Compute DFT shifted.

static std::vector<std::complex<double>> **DFT**(std::vector<std::complex<double>> data, int n)
Compute DFT.

static std::vector<std::complex<double>> **IDFT**(std::vector<std::complex<double>> data, int n)
Compute IDFT.

static inline int **EtaAuxiliaryFcn**(int s, int N)
Eta function definition.

static std::vector<double> **GetContourCurvature**(std::vector<std::complex<double>> data, double sigma)
Compute contout curvature.

static inline long int **factorial**(int n)
Compute factorial.

static inline double **Compute2DGausIntegral**(double A, double sigmaX, double sigmaY)
Compute 2d gaussian integral.

static inline double **Compute2DGausEllipseIntegral**(double A, double Bmaj, double Bmin)
Compute 2d gaussian ellipse integral.

static inline double **ComputeEllipseArea**(TEllipse *ellipse)
Compute ellipse area from ROOT TEllipse.

static inline double **ComputeEllipseArea**(double bmaj, double bmin)
Compute ellipse area.

static int **ComputeEllipseOverlapArea**(double &overlapArea, double &err, int &rtn, TEllipse *ellipse1, TEllipse *ellipse2, int method = 1, *Contour* *overlapContour = 0)
Compute overlap area between two ellipses.

static double **ComputePolygonArea**(polygon_2d &poly)
Compute polygon area.

static int **ComputeContourArea**(double &area, *Contour* *contour)
Compute contour area.

static int **ComputeContourOverlapArea**(double &overlapArea, int &overlapFlag, *Contour* *contour, *Contour* *contour2, *Contour* *overlapContour = 0)
Compute contour overlap area.

static int **ComputePolygonOverlapArea**(double &overlapArea, polygon_2d &overlap_poly, polygon_2d &poly, polygon_2d &poly2)
Compute overlap area between two polygons (in BOOST format)

static int **Contour2Polygon**(polygon_2d &poly, *Contour* *contour)
Convert contour to polygon (in BOOST format)

static int **Ellipse2Polygon**(polygon_2d &poly, double Cx, double Cy, double R1, double R2, double theta, int n = 20)
Convert an ellipse to polygon (in BOOST format)

static int **Ellipse2Polygon**(polygon_2d &poly, TEllipse *ellipse, int n)
Convert a TEllipse to polygon (in BOOST format)

static double **ComputeEllipseEccentricity**(TEllipse *ellipse)
Compute ellipse eccentricity from ROOT TEllipse.

static double **ComputeEllipseEccentricity**(double bmaj, double bmin)
Compute ellipse eccentricity from (bmaj, bmin)

template<typename **T**>
static inline *T* **Mod**(*T* x, *T* y)
Modulus operator.

static inline double **GetAngleInRange**(double theta, double theta_limit)
Limit angle in range [-theta_limit,theta_limit].

static inline double **EvalGaus2D**(double X, double Y, double A, double X0, double Y0, double sigmaX, double sigmaY, double theta)
Evaluate 2d elliptical gaussian at given (x,y) (NB: theta in radians)

static bool **IsPointInsidePolygon**(double x, double y, const std::vector<TVector2> &polygon)
Check if 2D point with coordinates (x,y) is inside a polygon.

static inline bool **IsPointInsidePolygon**(const TVector2 &point, const std::vector<TVector2> &polygon)
Check if 2D point with coordinates (x,y) is inside a polygon.

static void **ComputeRotatedCoords**(double &xrot, double &yrot, double x, double y, double cx, double cy, double theta)
Compute rotated coordinates (x,y)

static inline double **GetEuclideanDist**(double x1, double y1, double x2, double y2)
Compute Euclidean distance between two points.

static double **SynchrotronSelfAbsSED**(double *x, double *pars)
SynchrotronSelfAbsSED model.

static double **SynchrotronExtFreeFreeAbsSED**(double *x, double *pars)
SynchrotronExtFreeFreeAbsSED model.

static double **SynchrotronIntFreeFreeAbsSED**(double *x, double *pars)
SynchrotronIntFreeFreeAbsSED model.

static double **FreeFreeSED**(double *x, double *pars)
FreeFreeSED model.

class `Caesar::StatsUtils` : public TObject

Public Functions

StatsUtils()
Class constructor: initialize structures.

virtual **~StatsUtils**()
Class destructor: free allocated memory.

Public Static Functions

template<typename **T**>
static inline BoxStats<*T*> **ComputeBoxStats**(std::vector<*T*> &vec, bool sorted = false)
Compute box plot stats (min/max, median, Q1, Q3)

template<typename **T**>
static inline *T* **GetMedianFast**(std::vector<*T*> &vec, bool useParallelVersion = false, bool alreadySorted = false)
Compute median & 1st & 3rd quantiles.

Compute median using nth_element (should run in O(n)) Compute median using nth_element (should run in O(n))

template<typename **T**>
static inline *T* **GetMADFast**(std::vector<*T*> const &vec, *T* median, bool useParallelVersion = false)
Compute MAD using nth_element (should run in O(n))

template<typename **T**>
static inline *T* **GetMedian**(std::vector<*T*> &vec, bool isSorted = false)
Compute median using vector sorting (should run in O(nlog n))

template<typename **T**>

static inline *T* **GetRangeMedian**(std::vector<*T*> &vec, int start, int end, bool isSorted = false)
Compute median using vector sorting (should run in O(nlog n))

template<typename **T**>
static inline *T* **GetMAD**(std::vector<*T*> const &vec, *T* median)
Compute MAD using vector sorting (should run in O(nlog n))

template<typename **T**>
static inline void **ComputeMeanAndRMS**(*T* &mean, *T* &stddev, std::vector<*T*> const &vec)
Compute mean & std dev.

template<typename **T**>
static inline int **UpdateClippedStats**(ClippedStats<*T*> &stats_clipped, std::vector<*T*> &vec_clipped, std::vector<*T*> const &vec, ClippedStats<*T*> &stats, double clipsig = 3, bool useParallelVersion = false)
Compute clipped estimators.

template<typename **T**>
static inline int **GetClippedEstimators**(ClippedStats<*T*> &stats, std::vector<*T*> const &vec, *T* median, *T* mean, *T* stddev, double clipsig = 3, int maxiter = 5, double tol = 0.1, bool useParallelVersion = false)
Compute clipped stat estimators.

static double **GetMahalanobisDistance**(TMatrixD x, TMatrixD mean, TMatrixD Sigma, bool isInverted = false)
Compute Mahalanobis distance of input matrix.

static int **ComputePageRank**(std::vector<double> &ranks, TMatrixD &M, double d = 0.85, double tol = 1.e-4)
Compute page rank of input matrix.

template<typename **T**>
static inline *T* **ComputeNormMean**(*T* mean, *T* DataMin, *T* DataMax, *T* NormMin, *T* NormMax)
Compute norm mean after linear transformation from [DataMin,DataMax] to [NormMin,NormMax].

template<typename **T**>
static inline *T* **ComputeNormDiff**(*T* diff, *T* DataMin, *T* DataMax, *T* NormMin, *T* NormMax)
Compute norm difference between two values after linear transformation from [DataMin,DataMax] to [NormMin,NormMax].

template<typename **T**>
static inline void **ComputeNormMeanAndRMS**(*T* &mean_transf, *T* &rms_transf, *T* mean, *T* rms, *T* DataMin, *T* DataMax, *T* NormMin, *T* NormMax)
Get mean/rms after linear transformation from [DataMin,DataMax] to [NormMin,NormMax].

template<typename **T**>
static inline *T* **GetNormValue**(*T* x, *T* xmin, *T* xmax, *T* NormMin, *T* NormMax)
Get value normalized in a range.

template<typename **T**>
static inline void **NormalizeVector**(std::vector<*T*> &data, *T* NormMin = 0, *T* NormMax = 1)
Normalize a vector in a range.

static inline double **GetEllipseAxisSigmaContourScaleFactor**(double nsigma)
Multiply gaussian 2D ellipse axis by this factor to get the nsigma ellipse contour.

static inline double **GetEllipseAxisCLContourScaleFactor**(double CL)
Multiply gaussian 2D ellipse axis by this factor to get the CL ellipse contour.

static inline int **GetEllipseParsFromCovMatrix**(double &sigmaX_rot, double &sigmaY_rot, double &theta, double sigmaX, double sigmaY, double covXY)
Get 2D gaussian confidence ellipse.

static inline TEllipse *GetFitCLEllipse(double x0, double y0, double sigmaX, double sigmaY, double covXY, double CL)
Get 2D gaussian confidence ellipse.

static inline double GetGaus2DThetaPar(double sigmaX, double sigmaY, double covXY)
Get elliptical 2D gaussian.

static inline TEllipse *GetFitEllipse(double x0, double y0, double sigmaX, double sigmaY, double covXY, bool useFWHM = false)
Get ellipse corresponding to 2D gaussian.

template<typename T>
static inline int RemoveOutliers(std::vector<*T*> &data, double iqr_factor = 1.5, bool sorted = false)
Remove outliers from vector by iteratively removing data points outside from 1.5*IQR.

template<typename T>
static inline int ComputeStatsBootstrapError(std::map<std::string, *T*> &statsErr, const std::vector<*T*> &data, int nSamples = 100)
Extract a number of random samples from data vector with given sample size, with/without repetitions and uniform weights.

static int GenerateFitParsAroundCovMatrix(std::vector<std::vector<double>> &fitPars_rand, const std::vector<double> &fitPars, TMatrixD &fitCovMatrix, int nsamples = 1)
Generate fit parameters randomized around fit covariance matrix.

static int GenerateFitParsAroundCovMatrix(std::vector<std::vector<double>> &fitPars_rand, const std::vector<double> &fitPars, const std::vector<std::vector<double>> &fitCovMatrix, int nsamples = 1)
Generate fit parameters randomized around fit covariance matrix.

class `Caesar::Graph` : public TObject

Public Functions

Graph()
Standard constructor.

Graph(size_t nvertex)
Constructor with nvertex initializer.

Graph(TMatrixD const &A)
Construct from adjacency matrix.

virtual ~Graph()
Standard destructor.

inline int GetNVertexes()
Get number of vertex.

inline void AddVertex()
Add vertex.

int AddEdge(int v, int w)
Add edge.

TMatrixD *GetAdjacencyMatrix()
Get adjacency matrix.

int **GetConnectedComponents**(std::vector<std::vector<int>> &connected_items)
Get connected components in the aundirected graph.

class `Caesar::ZernikeMoments`

Public Functions

ZernikeMoments()
Class constructor.

~ZernikeMoments()
Class destructor.

Public Static Functions

static std::vector<double> **GetZernike2D_Direct**(*Image* *img, double order, double radius)
Compute zernike 2D moments with direct method.

static std::vector<double> **GetZernike2D**(*Image* *img, double order, double rad)
Compute zernike 2D moments.

static std::vector<double> **GetZernike2DOld**(*Image* *img, double D, double R)
Compute zernike 2D moments (TO BE DEPRECATED)

static std::vector<double> **mb_Znl**(double *X, double *Y, double *P, int size, double D, double m10_m00, double m01_m00, double R, double psum)
Define Zernike moment generating function.

15.5 System Utilities

These classes define system utility methods. Defined in `src/utils/SysUtils.h`

class `Caesar::SysUtils` : public TObject

Public Functions

SysUtils()
Class constructor: initialize structures.

virtual **~SysUtils**()
Class destructor: free allocated memory.

Public Static Functions

static bool **CheckFile**(std::string path, *FileInfo* &info, bool match_extension = false, std::string extension = "")
Check if a file exists in filesystem.

static bool **CheckDir**(std::string path)
Check if directory exists in filesystem.

static inline timespec **TimeDiff**(timespec start, timespec end)
Compute the difference between two timestamps.

static inline timespec **TimeSum**(timespec time1, timespec time2)
Compute the sum of two timestamps.

static inline std::string **GetISOTimestamp**()
Get current timestamp in ISO 8601 format string.

static inline double **TimeToSec**(timespec time)
Convert a timestamp to seconds.

static inline double **TimeToNSec**(timespec time)
Convert a timestamp to nanoseconds.

static int **GetFITSImageSize**(const std::string &filename, long int &Nx, long int &Ny)
Get the size of a FITS image.

static int **GetNCores**()
Get the number of cores in system.

static void **SetOMPThreads**(int nthreads)
Set the number of threads to be used by OpenMP (if enabled at build)

static int **GetOMPThreads**()
Get the number of threads currently used by OpenMP (return 0 if OMP is disabled at build)

static int **GetOMPMaxThreads**()
Get the maximum number of threads that can be used by OpenMP (return 0 if OMP is disabled at build)

static int **GetOMPCores**()
Get the number of cores currently used by OpenMP (return 0 if OMP is disabled at build)

static int **GetOMPThreadId**()
Get thread id (return 0 if OMP is disabled at build)

static bool **IsMPIInitialized**()
Is MPI run initialized (return 0 if MPI is disabled at build)

static bool **IsMPIFinalized**()
Is MPI run finalized (return true if MPI is disabled at build)

static int **GetProcMemoryInfo**(*ProcMemInfo* &info)
Get process used virtual memory.

static std::string **GetHost**()
Get hostname.

static int **GetProcId**()
Get processor id.

static std::string **GetAsciiLogo**()
Get ascii art logo.

static void **PrintAsciiLogo**()
Print ascii art logo.

class **FileInfo** : public TObject

class Caesar::**ProcMemInfo** : public TObject

Public Functions

inline **ProcMemInfo**()
ProcMemInfo constructor.

inline virtual **~ProcMemInfo**()
ProcMemInfo destructor.

inline std::string **GetPrintable**()
Get process mem info in printable string form.

inline void **PrintInfo**()
Print process mem info.

CHAPTER

SIXTEEN

CITING AND CITATIONS

When using CAESAR for a publication, **please cite the following article** in you paper:

S. Riggi et al., "Automated detection of extended sources in radio maps: progress from the SCORPIO survey", MNRAS (2016) doi: 10.1093/mnras/stw982, arXiv:1605.01852

or consider including me (S. Riggi, INAF - Osservatorio Astrofisico di Catania, Via S. Sofia 78, I-95123, Catania, Italy) as a co-author on your publications.

16.1 More References

CHAPTER

SEVENTEEN

FREQUENTLY ASKED QUESTIONS (FAQS)

Below some of the questions you might have in mind when using CAESAR:

1. In which language is implemented CAESAR?

CAESAR is implemented in C++.

2. Does CAESAR support python?

I know astronomers love python…CAESAR is developed in C++ and for this version you need to use it as it is. I used C++ for both personal preference, to re-use existing code and for performance reason. It is however possible to use CAESAR classes in python through the PyROOT interface. This feature is at present experimental and to be tested. Please read the *Usage* documentation section for further details.

3. In which operating systems CAESAR is supported?

CAESAR is specifically developed for Linux OS. It was used on Ubuntu and RedHat/Centos distributions. In principle it should be usable also in MacOS as all external dependencies (ROOT, OpenCV, boost, …) are supported in MacOS. For this you will need to manually modify the CAESAR Makefile. No support for Windows OS is available and never will be. If you are in trouble with the installation and you want to get started in short time, we provide a Singularity container with all software and dependencies already installed, ready to be used. Please read the *Container* documentation section for further details.

4. Does CAESAR support distributed or parallel processing?

Yes, CAESAR source finding supports two level of parallelism since 2018. Images can be partitioned in adjacent/overlapping tiles and source findind can be carried out on different computing processors using MPI library. Source finding tasks per each tile can be splitted among different threads using OpenMP library. Alternative programming models will be explored in the future. Scripts are provided for workload submission on a batch system (PBS/SLURM).

5. What data input formats can be handled?

CAESAR currently works on 2D images in different formats:

- FITS
- ROOT native Caesar::Image
- Standard image formats (png, jpeg, etc)
- OpenCV Mat
- VTK (to be tested)

Other formats are planned to be added:

- CASA images
- HDF5

Cubes are not supported.

6. What data outputs are delivered to user?

CAESAR returns the following outputs to user at the end of processing:

- **A ROOT file containing the following information:**
  - Computed maps (bkg, rms, significance, residual, etc) stored as Caesar::Image objects
  - Detected sources stored as a ROOT TTree of Caesar::Source objects
  - Run configuration options stored as a ROOT TTree
- **Two DS9 file regions containing:**
  - Detected sources (also called islands in other finders)
  - Gaussian components fitted to detected sources
- **Two ascii catalog files containing parameters relative to:**
  - Detected sources (also called islands in other finders)
  - Gaussian components fitted to detected sources
- Computed maps (bkg, rms, significance, residual, etc) in FITS format

CHAPTER

EIGHTEEN

LICENSE

CAESAR is distributed for research use only and licensed under the GNU General Public License v3.0 License:

```
License and Disclaimer

Copyright 2016 Simone Riggi

Caesar is free software: you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License,
or (at your option) any later version.
Caesar is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details. You should
have received a copy of the GNU General Public License along with
this program. If not, see <http://www.gnu.org/licenses/>.
```

CHAPTER

NINETEEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

C

D

E

F

I

L

W