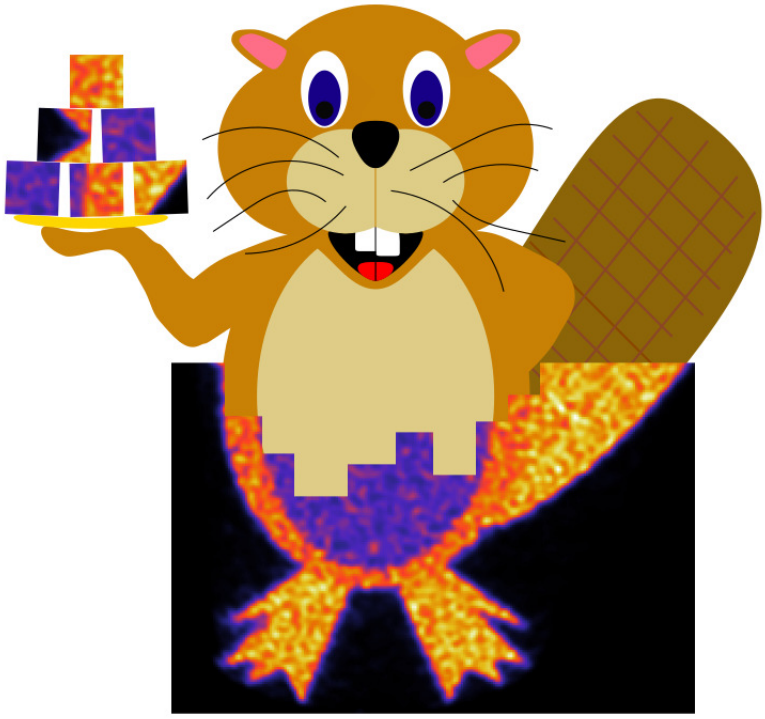
Customizable and Advanced Software for Tomographic Reconstruction (CASToR)

Version 1.1

July 18, 2017



Contents

Summary

This document is the main manual of the CASToR software platform for tomographic image reconstruction. It describes the general features of the platform and the scope of its applications. The input datafile formats and the description of the scanner systems are also detailed in this document. For developers, specific advanced manuals about different parts of the platform are available on the CASToR website.

The manual contains the following sections:

- Section 1 presents the features implemented in the platform.
- Section 2 introduces CASToR architecture and details some implementation issues.
- Section 3 describes how to install CASToR for Unix and Windows systems.
- Section 4 contains some basic steps to start with CASToR.
- Section 5 describes PET and SPECT systems definition in CASToR and their integration to the platform.
- Input PET and SPECT raw data and related metadata file formats used by CASToR are specified in section 6.
- Section 7 describes the Interfile image format used by CASToR.
- Section 8 presents several toolkits related to CASToR.

1 CASToR features

The current version of CASToR includes the following features:

- Iterative reconstruction algorithms.
- PET and SPECT (parallel-hole and convergent-beam collimators) geometries.
- Reconstruction of multi-frames and respiratory/cardiac gated acquisitions.
- Projector plug-in class system, including original [1] and accelerated [2] versions of the Siddon and Joseph [3] projectors.
- Optimization algorithm plug-in class system, including MLEM [4], NEGML [5], AML (AB-EMML) [6] and Lanwdeber [7] optimization algorithms.
- Plug-in class systems for image convolution and spatial regularization, and general image processing.
- Two-level parallel CPU implementation using MPI (multi-computers) and openMP (multi-threads) libraries.
- Interfile I/O format for images.
- Utility tool to convert GATE [8] Monte Carlo simulated data in ROOT format [9] into the CASToR datafile format.

Several advanced manuals dedicated to specific parts of the code as well as a Doxygen documentation are also available for developers (see http://www.castor-project.org/documentation).

2 CASToR architecture

The CASToR platform is an open-source software for tomographic image reconstruction, written in C++11. The key points of CASToR are generic programming, flexibility, and modularity. The code is segmented in several main components including data files management, image processing, optimization algorithms, projectors. To make the code generic and modular, specific classes implementing specific features in a component, inherit from an abstract class. Most of the code is implemented in the abstract classes so that the development of specific classes in the most comprehensive way requires the minimum amount of code. Following these guidelines, rather than implementing the different file formats specific to the scanner manufacturers, CASToR uses unified file formats for the scanner descriptions (see section 5), and for the input data (see section 6). This allows a unique implementation of the core of the reconstruction algorithms, without having to distinguish between a set of particular cases. The distinctions between the modalities (PET or SPECT), the types of data (list-mode or histogram) and the scanner manufacturers do not appear in the core algorithm. They are specified in the input data files and in the scanner description. This results in a few blocks of code specific to a scanner model or to a data type while keeping the rest of the code entirely generic. This avoids the duplication of parts of the code and any new adds-on to the code will hence be available for multiple setups and configurations. Based on this code design is that all correction terms required for quantitative reconstruction have to be estimated outside CASToR and must be supplied in the input datafile. This includes the normalization factors, the scattered events distribution, the random coincidences distribution in PET, and the attenuation coefficients in emission tomography. Finally, generic programing being often a source of inefficiency, special care has been taken to optimize as much as possible the code efficiency. The software has two levels of parallelism: a high-level splitting of the data using MPI (Message Passing Interface) and a low-level multi-threading of the loops over the events using the OpenMP (Open Multi-Processing) API.
Being generic requires a clear definition of all the concepts involved in tomographic reconstruction. For clarification, the definition of some of the terms and concepts used in CASToR are given below.

- An *algorithm* is the mathematical method used to solve a problem, like an inverse problem in tomographic image reconstruction. An *algorithm* can be analytical, iterative (in the sense of optimization), or of any other kind (SVD, posterior estimation, *etc*). Currently, the CASToR implementation considers only iterative optimization algorithms.

- In emission or transmission tomography, the data can be recorded in *list-mode* or *histogram* formats. In CASToR, whatever the data format, a *datafile* is defined as being a collection of *events*. There are currently four different types of events implemented in CASToR: PET histogram event, PET list-mode event, SPECT histogram event, and SPECT list-mode event. A list-mode event corresponds to the detection of one single photon or one coincidence. A histogram event corresponds to the data acquired during a given time frame in a given detection element, eventually for a given synchronization gate. An example of PET histogram event is the number of detected coincidences for a given pair of crystals in coincidence or for a given projection bin.

- A *datafile* is linked to a given *scanner* geometry. A *scanner* in CASToR can be described either by the geometry of a physical scanner (*e.g.* a collection of individual crystals arranged in a ring geometry for a cylindrical PET system) or by a generic *sinogram* geometry. In CASToR, the word *sinogram* is not used for a *datafile* format, but for the geometrical organization of the data in a discretized four-dimension projection space. Each *event* attached to a *sinogram* geometry corresponds to a specific projection line expressed by the standard 4-D sinogram coordinates: a radial position, an azimuthal angle, an axial position, and co-polar angle. On the other hand, each *event* attached to a physical PET scanner geometry corresponds to a specific pair of crystals in coincidence, or a group of crystal pairs in case of angular mashing or axial compression. As a consequence, both a PET histogram and a PET list-mode *event* can be attached to a *sinogram* geometry or a physical scanner geometry. Currently, CASToR

implements only physical PET and SPECT (for parallel hole and convergent-beam collimators) scanners geometries. The implementation of the *sinogram* scanner geometry is planned for future releases.

- Tomographic image reconstruction consists in the inversion of the data model linking the data space to the image space. Aside from the additive background in the detected events (scattered events and random coincidences in PET), this model is based on the so-called *projector*. In CASToR, the task of the *projector* is only to compute the system matrix elements corresponding to a given *event*, that is one row of the system matrix. The *projector* does not perform the forward or backward projections; they are computed inside the *algorithm*.

There are also general concepts that apply only to specific types of algorithms. The following subsections provide information for these different types of algorithms.

2.1 Iterative optimization algorithms

- Iterative optimization algorithms (like ML-EM) are based on an objective function (*Maximum Likelihood*, ML) and a iterative mechanism used to minimize or maximize this function(*Expectation Maximization*, EM). The objective function is made of a data fidelity term and potentially a penalty term. In CASToR, the iterative optimization algorithm is generic: both the objective function and the iterative optimization are pulled into what we call an *optimizer*. Particular *optimizers* can deal with *penalties*, although they are not yet implemented. For details about how the *optimizer* is implemented in CASToR, look at the *CASToR_HowTo_optimizer_and_penalty* specific documentation (see http://castor-project.org/documentation).

- For each iteration, and for each sub-iteration for optimization algorithms using ordered subsets of the data, a parallel loop over the *events* is performed using OpenMP. For each *event*, the following operations are performed:

  1. The *projector* is called to compute the corresponding system matrix elements, that is one row of the system matrix. If unmatched projectors are used, this operation is performed for both forward and backward projectors.
  2. The *optimizer* is called to perform the *data update step*. The following steps are performed:
     (a) Forward projection of the current image estimate for the given row of the system matrix.
     (b) Add to the content of the projection any provided estimation of the additive background (scattered events and random coincidences).
     (c) Perform operations specific to the *optimizer* to compute the correction term in the data space.
     (d) Backward projection of the correction term in a back-projected correction image for the given row of the system matrix.

  Based on this strategy, when matched forward and backward projectors are used, the *projector* is only called once. As it is the most costly operation in tomographic reconstruction, significant acceleration factors can be therefore easily achieved. Note that the *optimizer* performs the forward and backward projections taking all system matrix multiplicative terms into account: calibration factor, decay factor, decay branching ratio, frame duration, normalization and attenuation factors. For SPECT, given an attenuation map and one row of the system matrix, a double loop is performed to take the attenuation into account.

- For optimization algorithms using ordered subsets, in each subset the main loop is performed over all N_{Events} events of the datafile, with an increment equal to the number of subsets N_s, starting at the event index i_{event} equal to the index i_s of the current subset:

$$\text{for}(i_{\text{event}} = i_s; i_{\text{event}} < N_{\text{Events}}; i_{\text{event}} += N_s).$$

Note that the number of subsets can be different for each iteration. The same loop is used whatever the data type (list-mode or histogram) and modality. Thus, when using subsets in the *optimizer*, it is the user's responsibility to order the histogram events in an appropriate manner. To avoid potential artifacts in the reconstructed image, we strongly discourage to write the histogram events with the inner loop over the radial or axial coordinate. We recommend to randomly mix the order to the histogram events before writing them in the *datafile* or, at least, to write the events with the inner loop over the azimuthal coordinate.

- Iterative optimization algorithms require the computation of a sensitivity image, requiring a backward projection step over all possible (empty or not) detection elements. When reconstructing histogram data, the sensitivity image is computed on-the-fly by the *optimizer* for each subset. As the *projector* is called only once, the cost of the backward projection associated to the sensitivity image computation is minimal. It also allows to have a different number of subsets for each iteration. Note that for some specific algorithms (*e.g.* NEGML [5]), the sensitivity images depend on the current image estimate. When reconstructing list-mode data, the sensitivity image is precomputed before launching the iterations. The loop over all possible detection elements is based either only on the characteristics of the scanner geometry description (thus ignoring normalization factors) or on a user's provided normalization datafile including the normalization factors (see section 6).

2.2 Analytical algorithms

Analytical algorithms are not yet implemented into CASToR.

2.3 Other possible types of algorithms

Other types of algorithms exist, but are not yet implemented into CASToR.

3 CASToR installation

CASToR source code is available on the CASToR website at castor-project.org, once you complete the short registration form (castor-project.org/form/castor-registration-form). If you have suscribed before the 1st version release and did not receive any email for some reason, just fill in the form once again to get access to the download link. Do not hesitate to subscribe to the mailing-list in order to receive information about new versions of the platform and follow the CASToR community. In addition, if you are interested in contributing as a developer in the CASToR platform and have filled in the required fields in the subscription form, you will be contacted soon with details on how to contribute.

3.1 Computation and data precision

In CASToR, the precision of floating point numbers can be customized. This is implemented through the use of macro definitions interpreted by the pre-processor of the compiler, defining the type of different variables throughout the code, as follows:

- FLTNB: General computation precision (image matrices and all floating point operations).
- FLTNBDATA: Precision of the floating point variables read or written into CASToR datafiles.
- FLTNBLUT: Precision of the floating point variables read or written into CASToR scanner LUT elements.

The types of these macros are defined in the *include/management/gVariables.hh* and are all set to *float* simple precision by default. They can be changed to *double* or *long double* precision as desired, assuming that the operating system and the compiler are both able to take these types into account (see https://en.wikipedia.org/wiki/Long_double). When any of these definitions is changed, the code needs to be entirely recompiled. Note also that the memory requirements of the CASToR program will double with the use of *double* precision compared to *float* precision.

3.2 CASToR configuration folder

The CASToR programs may use a bunch of configuration files, for scanners, default module's configurations, isotopes database, etc. In the CASToR source tree, there is a folder named *config*, which contains all these configuration files. For instance, a datafile is linked to a scanner description; this scanner description, either based on a geometrical file or a LUT file (see section 5), is located in the *config/scanner* sub-folder. There is also one sub-folder per type of modules (*e.g.* projector, convolver, etc). And finally, there is a *config/misc* folder containing miscellaneous configuration files, such as isotopes' database with their associated half-life and branching ratio.

When running CASToR, the configuration folder is localized by the *CASTOR_CONFIG* environment variable. If the code was compiled using the CMake tools (see sub-sections 3.3.2 and 3.4.1), this variable has been hard-linked to the executable during the compilation. If the code was compiled using the provided Makefile (see sub-section 3.3.1), this variable is read dynamically from the *CASTOR_CONFIG* environment variable; so this variable must exist during run-time and point to the absolute path of the *config* directory. If the code is cross-compiled from a 64-bits Unix system for Windows systems (see sub-section 3.4.2), then the environment variable is also hard-linked during compilation. If the environment variable does not exist during the compilation, a compilation error will occur on purpose, specifying that you must set it. Note that the value of this environment variable for cross-compilation should contain the path of the configuration folder under the Windows system that will be used for execution. This path can use "/" instead of "\" but must not be provided within "double quotes". Finally, whatever the value of the *CASTOR_CONFIG* environment variable and whether it was hard-linked during the compilation or not, the path to the configuration folder can be also set by using the *-conf* command-line option and providing an alternative path.

3.3 Unix systems

There are two alternatives to compile the code: using the provided Makefile or the CMake tools. If you do not link CASToR to external libraries, we recommend using the standard Makefile as it is quite straightforward.

3.3.1 Compile CASToR on Unix systems using the provided Makefile

1. Get the CASToR source code
2. Configure the compilation using the following environment variables considered during the compilation (export CASTOR_XXX =):
   - CASTOR_CONFIG: *Set to the absolute path of the CASToR configuration directory (i.e. the config directory inside the source tree)*
   - CASTOR_MPI: *Set to 1 to build CASToR with MPI (required for multiprocessing / parallel computing at the datafile level)*
   - CASTOR_OMP: *Set to 1 to build CASToR with OpenMP (required for multithreading / parallel computing at the event level)*
   - CASTOR_ROOT: *Set to 1 to build CASToR with ROOT library (for datafile conversion using root. In case of issues with compilation using ROOT, check the root-config -libs and root-config -cflags variables are correctly initialized.)*
   - CASTOR_SIMD: *Set to 1 to build CASToR with SIMD optimization options from the compiler (CASToR does not include any specific vector programming)*
   - CASTOR_VERBOSE: *Set to 1 to output maximum information during reconstruction, mainly used for debugging*
   - CASTOR_DEBUG: *Set to 1 to activate many additional checks inside primitive functions of the code, mainly used for debugging as it can slow down the execution*
3. Run *make* to compile the CASToR code (you can use the '-j X' option of Makefile to use X threads during the compilation). It will create binaries for the main reconstruction program (castor-recon) and for each utility (section 8) into the *bin* directory.
4. (optional) Add CASToR to your environment using one of the following command lines:

   - bash or zsh:

   ```
   export PATH=$PATH:/path/to/CASToR/bin
   ```

   - tcsh

   ```
   setenv PATH ${PATH}:$/path/to/CASToR/bin
   ```

3.3.2 Compile CASToR on Unix systems using CMake

1. Install CMake: www.cmake.org
2. Get the CASToR source code
3. Configure the compilation using the following CMake variables:
   - CASToR_CONFIG: *Define the absolute path of the CASToR configuration directory (i.e. the config directory inside the source tree)*
   - CASToR_64bits: *Build CASToR on 64 bits architecture*
   - CASToR_ELASTIX: *Build CASToR with Elastix (Deformation class using Elastix will be provided in further release)*
   - CASToR_MPI: *Build CASToR with MPI (required for multiprocessing / parallel computing at the datafile level)*

- CASToR_OMP: *Build CASToR with OpenMP (required for multithreading / parallel computing at the event level)*
- CASToR_ROOT: *Build CASToR with ROOT library (for datafile conversion using root)*
- CASToR_SIMD: *Build CASToR with SIMD optimization options from the compiler (CASToR does not include any specific vector programming)*
- CASToR_VERBOSE: *Output maximum information during reconstruction, mainly used for debugging*
- CASTOR_DEBUG: *Activate many additional checks inside primitive functions of the code, mainly used for debugging as it can slow down the execution*

4. Run CMake for CASToR. It will create a makefile for your compiler.

5. Run *make* to compile the CASToR code (you can use the '-j X' option of Makefile to use X threads during the compilation). It will generate binaries for the main reconstruction program (*castor-recon*) and for each utility (see section 8) into the *bin* directory.

6. (Facultative) Add CASToR to your environment using one of the following command lines:

   - bash or zsh:

   ```
   export PATH=$PATH:/path/to/CASToR/bin
   ```

   - tcsh

   ```
   setenv PATH ${PATH}:$/path/to/CASToR/bin
   ```

3.4 Windows systems

Whatever the compilation method used, when executing CASToR on Windows systems, there may be some limits about the use of memory. In particular, due to the current implementation of the datafile read operations, some errors can appear above a certain amount of information read from the file. A workarouns is to limit the size of the portion of the datafile that is loaded in memory by using the option *-load*. The parameter of this option is the maximum percentage of the datafile that can be loaded in memory at each moment.

3.4.1 Compile CASToR on Windows systems with Visual Studio

The section provides several instructions on a way to install CASToR on windows using MSVC. It has been successfully installed and assessed using MSVC community 14.0.

The installation requires the use of CMake.

1. If you use windows console, be sure that the path of visual studio and the directory which will contain the CASToR executable is correctly set in the windows environment.

2. In the same console, open cmake (`cmake-gui`). Locate the source code and the directory where you want to build the binaries. Then configure the compilation using the following CMake variables:

   - CASToR_CONFIG: *Define the absolute path of the CASToR configuration directory (i.e. the config directory inside the source tree)*
   - CASToR_64bits: *Build CASToR on 64 bits architecture*
   - CASToR_ELASTIX: *Build CASToR with Elastix (Deformation class using Elastix will be provided in further release)*
   - CASToR_MPI: *Build CASToR with MPI (required for multiprocessing / parallel computing at the datafile level)*
   - CASToR_OMP: *Build CASToR with OpenMP (required for multithreading / parallel computing at the event level)*

- CASToR_ROOT: *Build CASToR with ROOT library (for datafile conversion using root)*
- CASToR_SIMD: *Build CASToR with SIMD optimization options from the compiler (CASToR does not include any specific vector programming)*
- CASToR_VERBOSE: *Output maximum information during reconstruction, mainly used for debugging*
- CASTOR_DEBUG: *Activate many additional checks inside primitive functions of the code, mainly used for debugging as it can slow down the execution*

3. Configure and generate. CMake will create a makefile for your compiler.

4. Build the code using Visual Studio (make sure you are in release mode), nmake or any other tool to generate binaries for the main reconstruction program (*castor-recon*) and for each utility (section 8).

3.4.2 Cross-compile CASToR on Unix systems for Windows systems with MinGW

If you do not possess Visual Studio on your windows system, a cross-compilation can be done under Unix systems. To do so, the MinGW-w64 project provides cross-compilers that can be run under 64-bits Unix systems to build executables for Windows systems, both 32-bits and 64-bits (see https://www.mingw-w64.org/). Under 64-bits Unix systems, the provided Makefile can be used with the CASTOR_MINGW environment variable set to either 32 or 64 in order to create binaries suitable for Windows 32-bits or 64-bits systems respectively. For the moment this was validated from an Ubuntu 14.04 system for a Windows 7 system (both 32- and 64-bits). It should work in principle for other systems.

4 Getting started

4.1 The *castor-recon* executable

The current implementation of the `castor-recon` program simply uses command-line options. In other words, it has no graphical interface yet and must be called through a command-line interpreter. Launching `castor-recon` without arguments will display only the main usage options. To get help about all other specific options, thematic help options can be used; they are displayed in the main usage options. As there are too many options, we will not give a detailed list here. We recommend you have a look at the benchmarks which are described in section 4.2.

As a simple example, let us say that we want to reconstruct a datafile named *my_data.cdh*. We use an iterative MLEM optimization algorithm using 10 iterations of 16 subsets. The projector used is the one proposed by Joseph[3]. A 3D gaussian of 4 mm transaxial FWHM and 4.5 mm axial FWHM including 3.5 sigmas in the kernel models a spatially uniform image-based PSF. Reconstructed images of $128 \times 128 \times 128$ voxels of $3 \times 3 \times 3$ mm^3 are saved into the output folder *my_images*. The command line would be the following:

```
castor-recon  -df my_data.cdh  -opti MLEM  -it 10:16
              -proj joseph  -conv gaussian,4.,4.5,3.5::psf
              -dim 128,128,128  -vox 3.,3.,3.  -dout my_images
```

4.2 Benchmarks

A good way to start experimenting with CASToR is to go through the different benchmarks provided on the web-site (`http://www.castor-project.org/benchmarks`). The benchmarks contain built-in CASToR datafiles from real acquisitions associated with scanners provided in the *config/scanner* directory from the CASToR source tree. In each benchmark, a script (both for Unix and Windows systems) is provided where all options are explained with details. It can be run to reconstruct an image from the provided data. A reference reconstructed image is also provided along with a program that can be used to check the consistency of this image compared to the one reconstructed from your system. In case you are interested in developing within CASToR, it can also be used to check the validity of any modification to the source code, providing such modifications were made to parts of the code executed in the context of each benchmark.

4.3 Using your own datasets

The reconstruction of simulated or acquired data requires *(i)* the integration of geometrical information regarding the scanner system and *(ii)* the conversion of the data to the CASToR datafile format. Section 5 provides guidance regarding how to integrate any supported system geometry in CASToR. Section 6 provides descriptions about CASToR datafile formats. Section 8 contains the description of several tools dedicated to dataset conversion:

- GATE geometry conversion script and root data converter (8.2).
- Code sample demonstrating how to convert any dataset to the CASToR format (8.4).

Finally, note that some data conversion binaries able to convert data from several commercial scanner systems can be found on the CASToR website (section 8.1.

5 Scanner system integration

This section provides guidance about the integration of the scanner geometry in the CASToR platform. The current implementation supports PET and convergent-beam SPECT (no pinhole collimator) geometry models. The geometrical information is recovered from geometry files, located in the scanner repository directory '${CASTOR_CONFIG}/scanner', where the environment variable ${CASTOR_CONFIG} must be defined beforehand (see Section 3).

The reconstruction program will look into this directory to find the appropriate system, according to the scanner name included in the data header file. The addition of a new scanner system requires writing a scanner description file and saving it in this directory; there is no need to recompile CASToR. Due to the various types of conceivable scanner architectures, two different methods are proposed to generate new scanner geometries:

- **.geom file** : For systems with generic architectures, the characteristics of the scanner can be described using a generic ASCII file. This file contains mandatory and optional fields about the geometry of the scanner. Based on this geometry ASCII file, at run-time, a Look-Up-Table (LUT) is generated that contains the 3D Cartesian coordinates and the orientation unit vector for all elementary detection units of the scanner (typically, crystals for PET), and for each projection for SPECT.

- **.lut file** : For systems featuring less generic geometries, the user has the possibility to precompute the LUT containing the Cartesian coordinates and the orientation unit vector for all elementary detection units of the scanner; currently only available for PET scanners. This binary LUT file is accompanied with an ASCII header file containing mandatory and optional fields about the scanner system.

For a given scanner, if both a geometric file and a LUT file are found in the *config/scanner* folder, the geometric one is used by default.

All detection elements are assumed to be of rectangular shape. The orientation unit vector provides the direction of the detection element depth. Positions and lengths are provided in millimeters, angles in degrees.

For the incorporation of geometries modeled using the GATE simulation platform, a separate utility can be used to convert a generic geometry defined in GATE macro files (.mac) to the CASToR geometry file format (.geom). Please see section 8.2 for more information.

5.1 PET systems

5.1.1 Scanner integration using a generic geometry file (.geom)

The .geom file is an ASCII file containing information about the structural design of a cylindrical PET scanner. It is based on the Cylindrical PET system of the GATE Monte-Carlo simulation platform[8] and breaks down into 5 nested levels (see Figure 1):

- layer;
- rsector (rotational sector);
- module;
- submodule;
- crystal.

Each element is of rectangular shape. The rotational sectors are arranged in a ring geometry, and all crystals within a rsector have the same orientation. The number of rotational sectors, modules, submodules, and crystals can vary from layer to layer. The elementary detection unit is the crystal. A look-up-table (LUT) is generated at run-time from this decomposition, attributing to each crystal

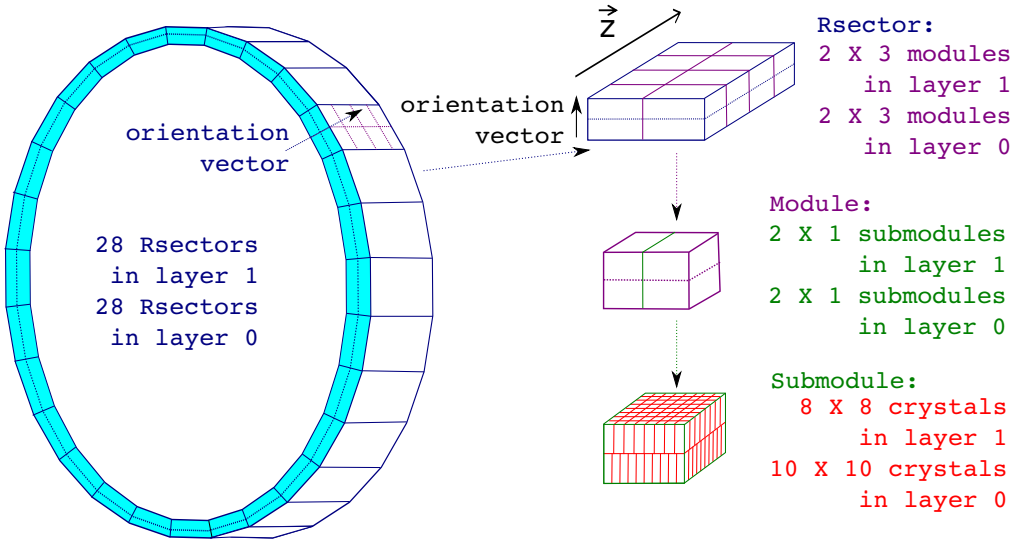


Figure 1: Illustration of the levels of decomposition of the PET scanner geometry, for a two-layer system.

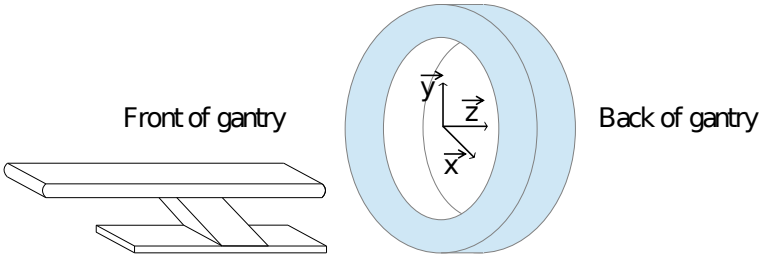


Figure 2: Left-handed Cartesian coordinate system $(\mathbf{x}, \mathbf{y}, \mathbf{z})$.

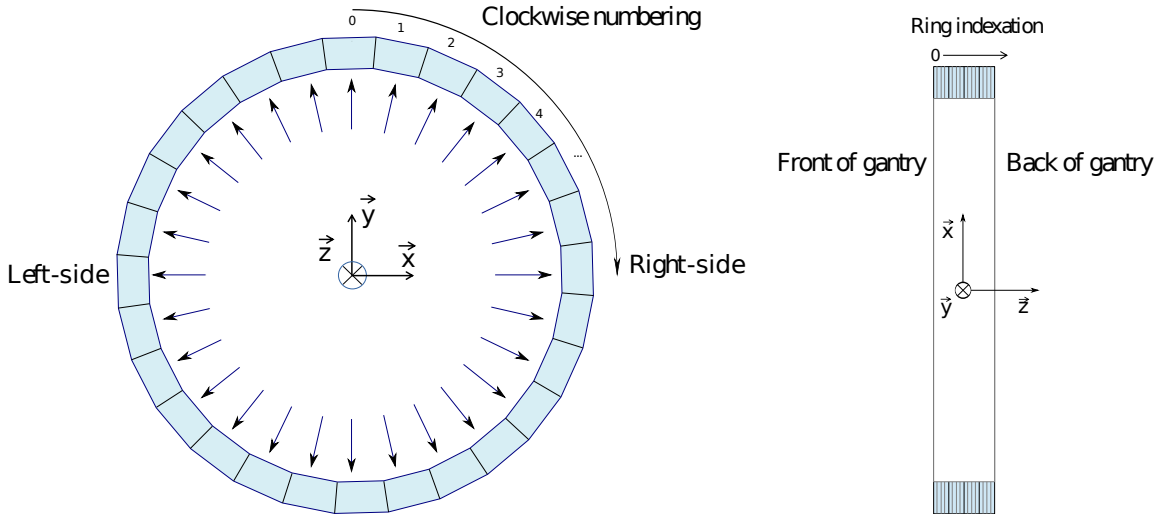
a unique identification number (ID) and computing the 3D Cartesian coordinates of its center and its orientation unit vector.

CASToR conventions about scanner elements indexation and orientations are illustrated in Figures 2 and 3. The elements numbering starts at zero. The origin of the left-handed Cartesian coordinate system $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is centered axially and transversally on the scanner. By default, CASToR will position the center of the first rsector directly above the scanner isocenter, then arrange the following rsectors by spanning them transversally over 360 degrees in the clockwise direction (as viewed from the front of the gantry) around the isocenter (see Figure 3(a)). The angular position of the first rsector relative to the $\mathbf{y}$ axis (θ_0, 0° by default) and the angular span (Δ_θ, 360° span by default) are customizable (see Figure 4). For a system of N rsectors, the angular position θ_i relative to the $\mathbf{y}$ axis of rsector i is then given by

$$\theta_i = i \times \frac{\Delta_\theta}{N} + \theta_0. \tag{1}$$

Crystal indexation A unique identification number is assigned to each crystal of the PET scanner. All the events in the data file will be identified by these crystal numbers. Crystals are first ordered transversally along a crystal ring, following a clockwise direction, and then axially across the crystal rings, from the front to the back of the gantry (see Figure 3). The first crystal is located in the first rsector and, when $\theta_0 = 0°$, possess the smallest value on the $\mathbf{x}$ axis. If the scanner contains several layers, all the crystals of the first layers are indexed before switching to those of the next layer. For example, for a scanner containing two layers, the index of the first crystal of the second layer will be equal to the total number of crystals in the first layer. Equation 2 provides the absolute crystal indexation *idCrystal* for an one-layer system according to the variables below:

- *nbRsectors*: number of rsectors



(a) Transverse indexation (b) Axial indexation

Figure 3: Transverse and axial indexations for PET scanner elements. Numbering starts at zero. The orientation unit vector of each rsector is also displayed; its axial coordinate is null. (a) View from the front. (b) View from the bottom.

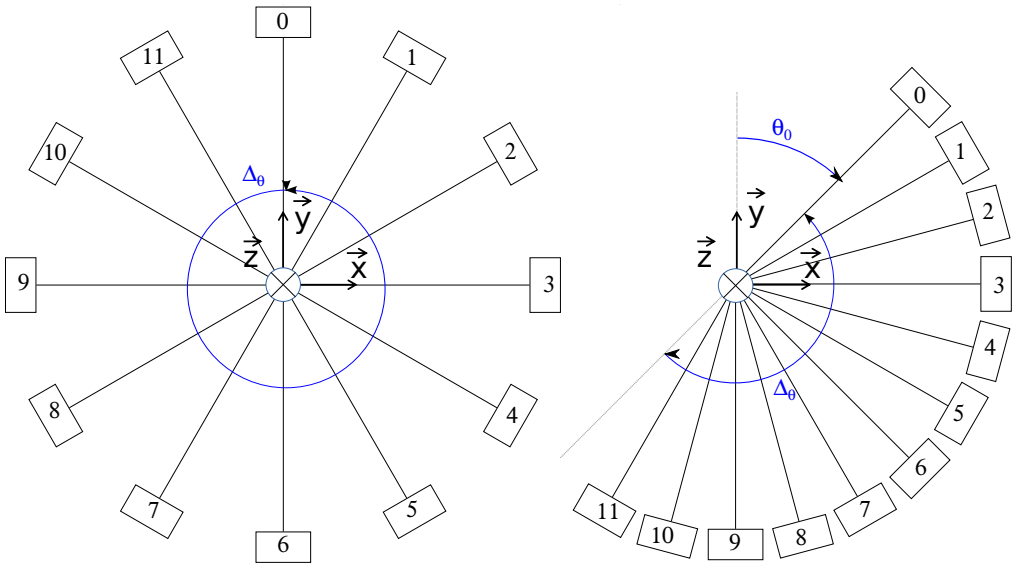


Figure 4: Angular spanning for a PET scanner consisting of 12 rsectors. Left: $\theta_0 = 0°$, $\Delta_\theta = 360°$. Right: $\theta_0 = 45°$, $\Delta_\theta = 180°$.

- $nbAxialModules$ / $nbTransaxialModules$: number of axial/transaxial modules in a rsector
- $nbAxialSubmodules$ / $nbTransaxialSubodules$: number of axial/transaxial submodules in a module
- $nbAxialCrystals$ / $nbTransaxialCrystals$: number of axial/transaxial crystals in a submodule
- $idRsector$: rsector id
- $idAxialModule$ / $idTransaxialModule$: axial/transaxial module id in a rsector
- $idAxialSubmodule$ / $idTransaxialSubmodule$: axial/transaxial submodule id in a module
- $idAxialCrystal$ / $idTransaxialCrystal$: axial/transaxial crystal id in a submodule

$$\begin{aligned} idCrystal =& idRing \times nbCrytalsPerRing \\ &+ idTransaxialCrystal \\ &+ idTransaxialSubmodule \times nbTransaxialCrystals \\ &+ idTransaxialModule \times nbTransaxialSubodules \times nbTransaxialCrystals \\ &+ idRsector \times nbTransaxialModules \times nbTransaxialSubodules \\ &\times nbTransaxialCrystals, \end{aligned} \tag{2}$$

where the total number of crystals in a ring, $nbCrytalsPerRing$, is given by

$$\begin{aligned} nbCrytalsPerRing =& nbRsectors \times nbTransaxialModules \\ &\times nbTransaxialSubodules \times nbTransaxialCrystals \end{aligned} \tag{3}$$

and the ring index $idRing$ is given by

$$\begin{aligned} idRing =& idAxialCrystal + idAxialSubmodules \times nbAxialCrystals \\ &+ idAxialModule \times nbAxialSubmodules \times nbAxialCrystals. \end{aligned} \tag{4}$$

Geom file structure The name of the .geom file should correspond to the scanner name, followed by the file extension geom. The file shall be located in the scanner repository directory '${CASTOR_CONFIG}/scanner', where examples of .geom files for various PET scanner systems can be found. The description of each mandatory and optional field in the .geom file is provided below. The fields are case sensitive. The separator (between the field and the value) is the colon sign ":".

Mandatory fields

- **modality**: PET.
- **scanner name**: no restriction. Scanner names are usually referenced using the following template: `Modality_Manufacturer_Model`, (*e.g.*: `PET_Siemens_mMR`).
- **description**: any string containing additional information about the scanner.
- **number of elements**: total number of crystals.
- **number of layers**: number of layers.
- **voxels number transaxial**: default number of voxels in the **x** and **y** directions for the reconstructed images (only used if the image matrix dimensions are not provided by the user in the command-line options (*-dim*)).

- **voxels number axial**: default number of voxels in the **z** direction for the reconstructed images (only used if the image matrix dimensions are not provided by the user in the command-line options (*-dim*)).
- **field of view transaxial**: default reconstructed transverse FOV in mm (only used if the FOV or voxel dimensions are not provided by the user in the command-line options (*-fov* or *-vox*)).
- **field of view axial**: default reconstructed axial FOV in mm (only used if the FOV or voxel dimensions are not provided by the user in the command-line options (*-fov* or *-vox*)).

The following variables depend on the number of layers. In the event that the scanner contains several layers, each variable must list the values corresponding to each layer, separated by commas (*e.g.*: `field: val_layer1, val_layer2`):

- **scanner radius**: for each layer, radial distance in mm between the scanner isocenter and the front face of the rsectors (see Figure 5(a)).
- **number of rsectors**: for each layer, number of rotational sectors.
- **number of crystals transaxial**: for each layer, number of transverse crystals within a submodule.
- **number of crystals axial**: for each layer, number of axial crystals within a submodule.
- **crystals size depth**: for each layer, depth size in mm of the crystal along the orientation vector.

Optional fields
The following optional variables also depend on the number of layers.

- **crystals size trans**: for each layer, transverse size in mm of the crystal.
- **crystals size axial**: for each layer, axial size in mm of the crystal.
- **rsectors first angle**: for each layer, transaxial angular position in degree of the center of the first rsector (default is 0°, see Figure 4).
- **rsectors angular span**: for each layer, the transverse angular span in degree of the rsectors (default is 360°, see Figure 4).
- **number of modules transaxial**: for each layer, number of transverse modules within a rsector (default is 1).
- **number of modules axial**: for each layer, number of axial modules within a rsector (default is 1).
- **module gap transaxial**: for each layer, transverse gap in mm between two consecutive modules within a rsector (default is 0.0 mm).
- **module gap axial**: for each layer, axial gap in mm between two consecutive modules within a rsector (default is 0.0 mm).
- **number of submodules transaxial**: for each layer, number of transverse submodules within a module (default is 1).
- **number of submodules axial**: for each layer, number of axial submodules within a module (default is 1).
- **submodule gap transaxial**: for each layer, transverse gap in mm between two consecutive submodules within a module (default is 0.0 mm).

- **submodule gap axial**: for each layer, axial gap in mm between two consecutive submodules within a module (default is 0.0 mm).
- **crystal gap transaxial**: for each layer, transverse gap in mm between two consecutive crystals within a submodule (default is 0.0 mm).
- **crystal gap axial**: for each layer, axial gap in mm between two consecutive crystals within a submodule (default is 0.0 mm).
- **mean depth of interaction**: for each layer, from the crystal front surface, mean depth of interaction in mm along the orientation vector (default is crystal depth size divided by 2).

The following optional variables do not depend on the number of layers.

- **min angle difference**: minimum transverse angle difference in degree between two crystals to define a valid line of response. This parameter is principally used for the list-mode sensitivity image generation, in order to model any hardware restriction on the lines of response (default is 0°, see Figure 5(a)).
- **rsectors nbZShift**: number of distinct axial shift values for the rsector, similar to the GATE [8] *Zshift* definition (default is 0, see Figure 5(b)).
- **rsectors ZShift**: axial shift values in mm, separated by commas. The number of values must be equal to the value provided in **rsectors ZShift**. The axial shift value for rsector i is then given by $ZShift[i \pmod{nbZShift}]$.

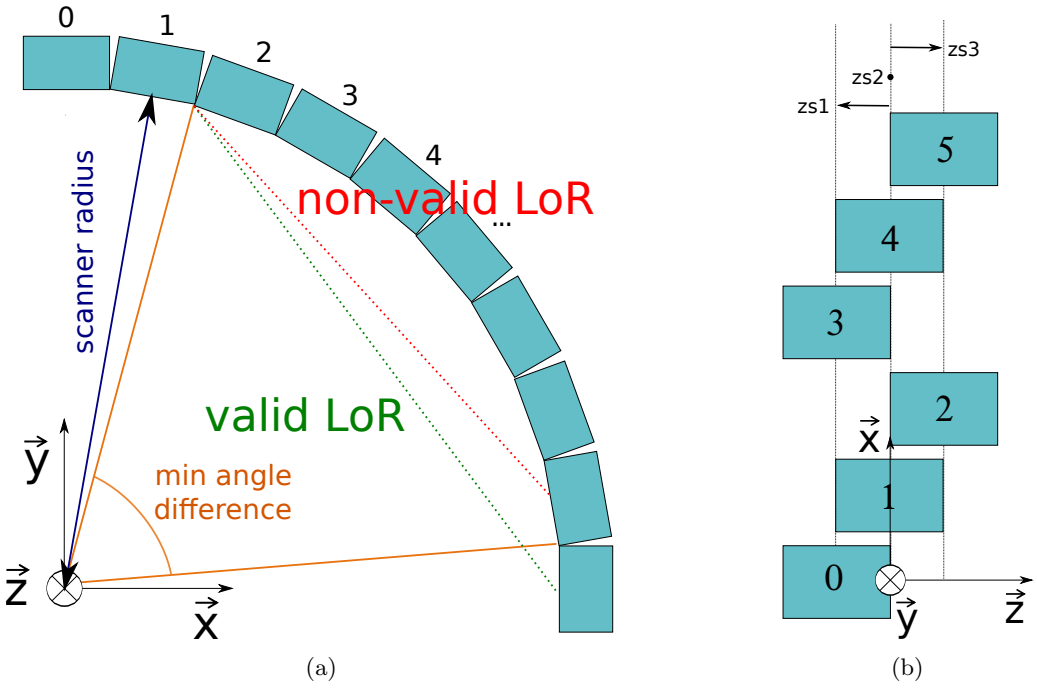


Figure 5: Illustration of several transverse (a) and axial (b) geometric variables related to the PET scanner configuration files.
(a)***scanner radius*** is defined as the radial distance between the scanner isocenter and the front face of the rsector. The ***min angle difference*** represents the required minimal angular difference between two detectors to generate a line of response.
(b) System with 3 different axial shifts $zs1, zs2, zs3$ (***rsectors nbZShift*** = 3 ; ***rsectors ZShift*** = -10,0,+10 mm).

The decomposition of the scanner geometry used in the .geom files should be sufficient to model the geometry of most known PET scanners. This is also the most user-friendly method to integrate

a scanner as the reconstruction algorithm will generate itself the LUT. The user still has to perform the conversion from the original data indexation to the CASToR indexation, by applying Equation 2. The *castor-datafileConversionEx* script could be used in order to facilitate this operation (see section 8.4). Another way to integrate a PET scanner is by directly providing a LUT integrating the Cartesian coordinates and the orientation unit vector of all the scanner detection elements, as described in section 5.1.2.

5.1.2 Scanner integration using a pre-computed LUT file (.lut)

This method is advised for scanners with unconventional design and for users who would like to generate a LUT with their own crystal indexation. The scanner LUT is composed of two files: an ASCII header file (with the .hscan file extension) containing metadata about the scanner specifications and a binary data file (with the .lut file extension) containing the LUT. Both files must have the same basename, given by the scanner name, and be located in the scanner repository directory '${CASTOR_CONFIG}/scanner'.

Binary LUT file The binary data in the .lut file must contain a number of *words* equal to the number of elementary detection units (typically, crystals for PET scanners). The content of each word is described in Table 1. It defines the coordinates and the orientation of the elementary detection unit along its depth in the left-handed Cartesian coordinate system described in Figure 2.

It is possible to group the elementary detection units into *layers*: all elementary detection units within a given layer have the same depth size. If the system consists of only one layer, then all the elementary detection units have the same depth size. It is also possible to group the elementary detection units within a layer into *rings*. The number of elementary detection units and the number of rings can vary between layers, but for a given layer, the number of elementary detection units per ring is constant. The ring value of a given detection unit is only used to limit the co-polar aperture (maximum ring difference) during a list-mode reconstruction.

For a system consisting of `NLayer` layers and `NRing[iLayer]` rings and `NDet[iLayer]` elementary detection units per layer `iLayer`, the writing of the elementary detection unit words in the LUT file shall satisfy the following order:

```
for(iLayer=0;iLayer<NLayer;iLayer++)
 for(iRing=0;iRing<NRing[iLayer];iLayer++)
  for(iDet=0;iDet<NDet[iLayer]/NRing[iLayer];iDet++) {
   fwrite(&Posx[iLayer][iRing][iDet],sizeof(FLTNBLUT),1,fPtr);
   fwrite(&Posy[iLayer][iRing][iDet],sizeof(FLTNBLUT),1,fPtr);
   fwrite(&Posz[iLayer][iRing][iDet],sizeof(FLTNBLUT),1,fPtr);
   fwrite(&OrVx[iLayer][iRing][iDet],sizeof(FLTNBLUT),1,fPtr);
   fwrite(&OrVy[iLayer][iRing][iDet],sizeof(FLTNBLUT),1,fPtr);
   fwrite(&OrVz[iLayer][iRing][iDet],sizeof(FLTNBLUT),1,fPtr);
}
```

Table 1: The content of one word, corresponding to one elementary detection unit, in the binary LUT file. Distances are given in mm. The default data type is single-precision floating-point. The data type can be changed by editing the variable 'FLTNBLUT' in the 'gVariables.hh' file (see Section 3.1). The orientation unit vector shall satisfy $OrVx^2 + OrVy^2 + OrVz^2 = 1$.

Tag	***Description***
Posx	Position of the center of the elementary detection unit on the X-axis
Posy	Position of the center of the elementary detection unit on the Y-axis
Posz	Position of the center of the elementary detection unit on the Z-axis
OrVx	Orientation unit vector of the elementary detection unit on the X-axis
OrVy	Orientation unit vector of the elementary detection unit on the Y-axis
OrVz	Orientation unit vector of the elementary detection unit on the Z-axis

Header LUT file The content of the ASCII LUT header file is described below. It includes mandatory and optional fields. The keywords are case sensitive. The separator (between the field and the value) is the colon sign ":".

Mandatory fields

- **modality**: PET.
- **scanner name**: no restriction. Scanner names are usually referenced using the following template: `Modality_Manufacturer_Model`, (*e.g.*: `PET_Siemens_mMR`).
- **description**: any string containing additional information about the scanner.
- **number of elements**: total number of elementary detection units (typically, crystals in PET).
- **number of layers**: number of layers.
- **voxels number transaxial**: default number of voxels in the **x** and **y** directions for the reconstructed images (only used if the image matrix dimensions are not provided by the user in the command-line options (*-dim*)).
- **voxels number axial**: default number of voxels in the **z** direction for the reconstructed images (only used if the image matrix dimensions are not provided by the user in the command-line options (*-dim*)).
- **field of view transaxial**: default reconstructed transverse FOV in mm (only used if the FOV or voxel dimensions are not provided by the user in the command-line options (*-fov* or *-vox*)).
- **field of view axial**: default reconstructed axial FOV in mm (only used if the FOV or voxel dimensions are not provided by the user in the command-line options (*-fov* or *-vox*)).

The following variables depend on the number of layers. In the event that the scanner contains several layers, each variable must list the values corresponding to each layer, separated by commas (*e.g.*: `field: val_layer1, val_layer2`):

- **number of rings in scanner**: for each layer, number of axial rows of elementary detection units (typically, crystal rings in PET).
- **number of crystals in layer**: for each layer, number of elementary detection units (typically, crystals in PET).
- **crystals size depth**: for each layer, depth size in mm of the elementary detection units along the orientation vector (typically, crystals in PET).

Optional fields

- **mean depth of interaction**: for each layer, from the elementary detection unit front surface, mean depth of interaction in mm along the orientation vector (default is elementary detection unit depth size divided by 2).
- **min angle difference**: minimum transverse angle difference in degree between two crystals to define a valid line of response. This parameter is principally used for the list-mode sensitivity image generation, in order to model any hardware restriction on the lines of response (default is 0°, see Figure 5(a)). There is a unique value for all layers.

5.2 SPECT systems using parallel/convergent collimators

The most important part of SPECT reconstruction consists in the modelling of the collimator which will affect the solid angle providing the probability of a photon originating from the scanned object to reach the gamma camera. Due to the numerous types and geometries of SPECT parallel and convergent collimators, CASToR does not rely on the full modeling of the collimator in the projector. Instead, the collimator is characterized by a polynomial equation whose coefficients are defined in the SPECT gamma camera configuration file. The order and parameters of the polynomial equations allow to specify the collimator type and features. The couple of SPECT data indices, defined as a projection angle associated to a crystal/pixel pair, are used to compute a focal point. Cartesian positions of this focal point and the crystal/pixel of the gamma camera allows to outline a projection ray/volume.

Although CASToR uses polynomial equations by default to characterize the different types of collimator, one can implement his own model in the SPECT geometry class to estimate the focal point of a gamma camera pixel/projection angle couple, or even model the entire geometry of the collimator in a projector class.

5.2.1 Scanner integration using a generic file (.geom)

The .geom ASCII file for SPECT scanners contains information about the structural design of the SPECT gamma camera head(s), and the focal model and parameters which characterize the collimator.

The description of each mandatory and optional field in the geom file is provided below. The keywords are case sensitive. The separator (between the field and the value) is the colon sign ":".

General fields :

- **modality**: `SPECT_CONVERGENT`
- **scanner name**: no restrictions, but it is adviced to follow the following template: `Modality_Constructor_Model` (*e.g.* `SPECT_Siemens_INTEVO`).
- **number of detector heads**: total number of heads in the SPECT system
- **trans number of pixels**: total number of transaxial pixels (crystals) in the gamma camera head(s) (1 for monolythic cameras).
- **trans pixel size**: Transaxial size of each pixel/crystal in mm
- **trans gap size**: Transaxial gap between pixels/crystals in mm
- **axial number of pixels**: total number of axial pixels (crystals) in the gamma camera head(s) (1 for monolythic cameras).
- **axial pixel size**: Axial size of each pixel/crystal in mm
- **axial gap size**: Axial gap between pixels/crystals in mm
- **detector depth**: Crystal depth (mm). This parameter is currently not used in CASToR reconstruction, but is included here for future developements. The following variables depend on the number of detector heads. If the SPECT system contains several heads, each variable must list the values corresponding to each head, separated by commas.
- **scanner radius**: default global distance in mm between the scanner's center of rotation (COR) and the head's collimator. If this distance is dependent on the projection angle, this information must be provided in the datafile header and will overwrite this value.

The following variables are related to the collimator configuration. If the SPECT system contains several heads, the set of parameters specific to each head must be provided and preceded of the head they belong to (*head1*, *head2*, etc):

- **trans focal model**: string corresponding to the transaxial focal model. Must be *constant*, *polynomial*, *slanthole*, or *custom*
- **trans number of coef model**: Number of parameters for the transaxial model. Must be 1,2 or 3 for *polynomial* (see eq. 5 below), 1 for *slanthole* (see eq. 6), or any number for *custom*. This parameter is discarded for the *constant* model.
- **trans parameters**: A number of parameters corresponding to the previous field, separated by commas. For the polynomial model, coefficients must be given in ascending degree order (c,b,a, as defined in eq. 5 below)).
- **axial focal model**: string corresponding to the axial focal model. Must be *constant*, *polynomial*, *slanthole*, or *custom*.
- **axial number of coef model**: Number of parameters for the transaxial model. Must be 1,2 or 3 for *polynomial* (see eq. 5 below), 1 for *slanthole* (see eq. 6), or any number for *custom*. This parameter is discarded for the *constant* model.
- **axial parameters**: A number of parameters corresponding to the previous field, separated by commas. For the polynomial model, coefficients must be given in ascending degree order (c,b,a, as defined in eq. 5 below)).

5.2.2 Collimator focal model defined by the (.geom) file

The current implementation of SPECT parallel and convergent collimators in CASToR relies on several models, which either apply to the transaxial or axial directions. Different models could be selected for the transaxial/axial direction in order to model collimators such as a fan-beam for example. These models are listed below:

constant: The *constant* model (Fig. 6) defines a parallel hole collimator. It does not require any parameters as the focal point axial/transaxial geometric position will be computed as the symetrical opposite to the SPECT pixel relatively to the axial/transaxial plane parallel to the detector surface and containing origin. A parallel collimator requires constant models for both axial/transaxial directions while a fan-beam collimator requires a constant model for one of these directions.

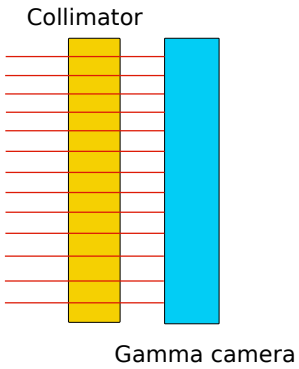


Figure 6: Constant model for SPECT collimator

polynomial: The *polynomial* model allows to characterize convergent and multi-convergent collimators using polynomial equations of different order to compute the focal point(s). The polynomial expression is interpreted in mm (the default distance unit used in CASToR). The axial/transaxial

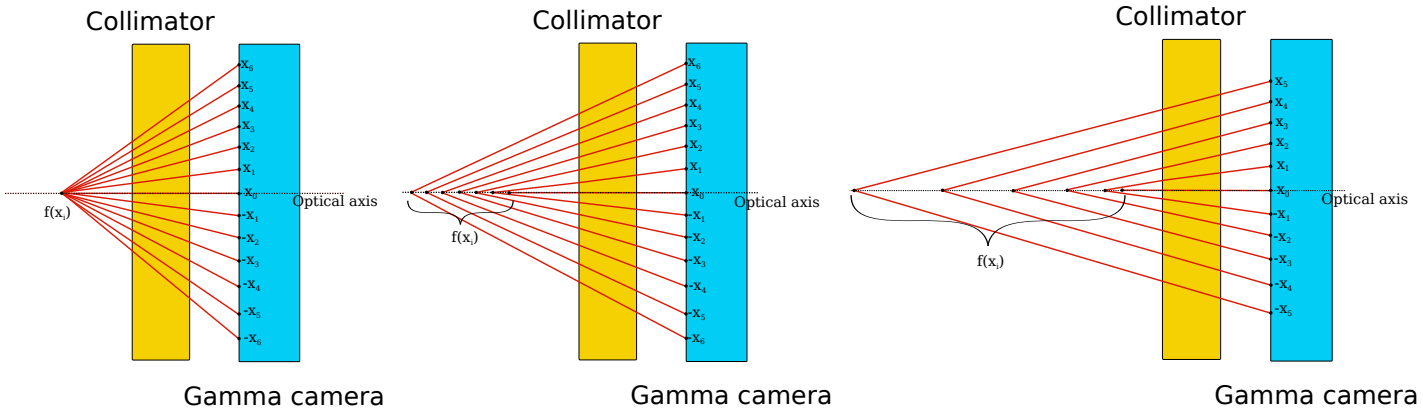
focal length corresponding to a SPECT pixel in the axial/transaxial plane is located on the optical axis (line orthogonal to and centered on the detector surface), and whose position is computed as:

$$f(x) = ax^2 + b|x| + c \tag{5}$$

where, x corresponds to the axial/transaxial distance between the SPECT pixel position, and its orthogonal projection on the optical axis (Fig. 7).

A polynomial function with one coefficient (degree 0) results in a mono-convergent (or mono-divergent if a negative coefficient is provided) collimator on a specific direction, as each crystal of the gamma camera for a given projection angle will have a similar focal point (Fig. 7(a)) on that direction. A fan-beam collimator can be modeled as a collimator with a constant model on one of the transaxial/axial direction, and a degree 0 polynomial model on the other direction. A mono-convergent cone-beam collimator corresponds to a polynomial model for each direction.

A polynomial function with two or three coefficients (degree 1 or 2) results in a multi-convergent (or multi-divergent if a negative coefficient is provided) collimator on a specific direction, as each crystal of the gamma camera for a given projection angle will have a different focal point on that direction (Figs. 7(b) and 7(c)).



(a) Polynomial, degree 0 (b) Polynomial, degree 1 (c) Polynomial, degree 2

Figure 7: Polynomial models for SPECT collimators using either one (a), two (b) or three (c) coefficients. x_i represent the distances between each pixel i reference position and the optical axis (either in the axial or transaxial plane)

slanthole: The *slanthole* model (Fig. 8) allows to characterize slanthole collimators using polynomial equations of degree 1 to compute the focal point(s). The slanthole model is a specific case of parallel hole collimators where all the collimator septas are rotated according to a given angle. The focal position is computed as

$$f(x) = ax \tag{6}$$

where, a corresponds to the slanthole unique coefficient defining the angle of the collimator, while *sign(a)* defines the angle orientation.

custom: The *custom* model provides the opportunity for the user to implement his own collimator model characterized by his own equations (such as hyperbolic or any other kind of equation). In the current CASToR version, the custom model must be implemented in the *iScannerSPECTConv::ComputeFocalPositions()* function in the *src/scanner/iScannerSPECTConv.cc* file, in the conditionnal statements related to the "custom" model, for both axial and transaxial focal position estimations. The current implementation of the custom model returns an error by default.

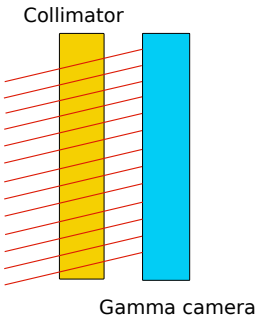


Figure 8: Slanthole models for SPECT slanthole collimators

5.2.3 Acquisition-specific geometry information

The SPECT geometry will use several acquisition-specific information from the datafile header in order to generate the geometry (see section 6.2). The keywords are case sensitive. The separator (between the field and the value) is the colon sign ":".

Number of projections: The total number of projections (for all heads if the system contains more than one gamma camera) will be read from the datafile header.

Projection angles: The Cartesian positions of each pixel/crystal of the gamma camera and their corresponding focal points will be computed for each projection angles. The projection angles must therefore be provided in the datafile header. If the SPECT system contains several heads, the angles for the first head must be entered first, followed by the angles of the second head, etc.

Number of bins: The transaxial and axial number of bins must be provided in the datafile header. The pixel sizes will be computed from the number of bins and the gamma camera size. If this information is not provided (as for a pixellated detector), the related fields from the SPECT configuration file (*transaxial/axial number of pixels*) will be used instead. Pixels contained in each axial row must be subsequently indexed, starting from the pixels with the smallest x and z Cartesian positions (considering the head is located directly above the center of rotation), as presented in Fig. 9.

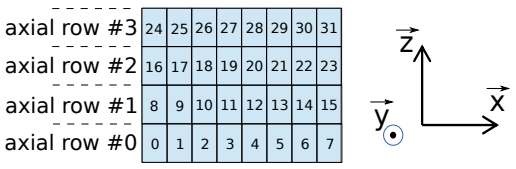


Figure 9: Pixel indexation in a SPECT gamma camera with 4 axial rows of 8 transaxial pixels.

(Global) Distance camera surface to COR: The distance between the center of rotation and the detector surface will be read from the datafile before computing the focal points and crystal's geometric Cartesian locations. If this distance depends on the projection angle, the distance corresponding to each projection angle must be provided after the field ***Distance camera surface to COR***. If this distance is similar for each projection angle, one value must be provided by the field ***Global distance camera surface to COR***.

If none of these are provided in the datafile, the values given from the SPECT configuration file (***scanner radius*** tag) will be used instead.

6 CASToR input datafile format

As explained in section 2, CASToR uses an unified description for input data to be reconstructed. This section describes the format of the datafile for the different data types (PET, SPECT, etc) and data mode (histogram, list-mode and normalization). Each datafile consists in two separate files: an ascii header and a binary *datafile*. The header contains all mandatory information about the acquisition, the system, the data type, as well as information about the optional correction factors embeded into the datafile. The binary *datafile* contains the raw data from the acquisition (or the simulation), described by a series of *events* gathering several mandatory and optional fields. These fields depend on the modality and acquisition types.

There are no predefined file extension for CASTOR datafiles. The ascii header file is provided to the reconstruction program, and this header file contains the name of the associated binary datafile. However, we commonly use the *.cdh* and *.cdf* extensions for header and binary file respectively, standing for *CASToR data header/file*.

6.1 PET datafile format

By PET data, we mean PET data linked to a PET geometry as described in section 5.1. In this context, some PET scanners include intrinsic compression of the data channels, either axially (*e.g.* span) or over the azimuthal angles (*e.g.* mashing). Raw data may have been also saved using compression factors. Note here that we do not talk about computer compression (*e.g.* zip, rar or any proprietary compression algorithm). To deal with this situation in a generic way (*i.e.* without any manufacturer's specific methodology), CASToR uses an optional field in the ascii header file specifying the maximum number of data channels contributing to a given event. If this number is higher than 1, for each event in the binary datafile, the actual number of data channels contributing to this event is given, followed by the list of all these data channels. This mechanism holds for any mode of data.

PET datafiles can contain histogram or list-mode data with associated corrections. An additional datafile mode can be used to describe the whole set of operational data channels with their associated normalization factors. When performing a list-mode reconstruction, the normalization datafile can be used to compute the sensitivity image. The loop will then be performed over the normalization *events* (one per data channel) contained in the normalization datafile, taking the normalization factors into account. If not provided, a loop over the scanner elements is performed, assuming a uniform efficiency and no compression. For the sensitivity computation, an attenuation image containing attenuation coefficients (in cm^{-1}) can be provided to correct for attenuation (providing the attenuation correction factors are also present in the datafile to be reconstructed). When using a normalization datafile to compute this sensitivity image, the attenuation correction factors can also be embeded so that they are taken into account during the computation (again providing the attenuation correction factors are also present in the datafile to be reconstructed).

6.1.1 ASCII header

The content of the ascii header file is described below. It includes mandatory and optional fields. All keywords are case sensitive and the separator (between the field and the value) is the colon sign ":".

Mandatory fields

- **Scanner name**: The name of the PET scanner (corresponding to a scanner in the *config/scanner* folder, without the file extension)
- **Data filename**: The absolute or relative path to the binary datafile
- **Number of events**: The number of CASToR events in the binary datafile
- **Data mode**: Can be "list-mode", "histogram" or "normalization"

- **Data type**: PET
- **Start time (s)**: The relative start time of the acquisition in seconds (ignored for normalization mode)
- **Duration (s)**: The duration of the acquisition in seconds (ignored for normalization mode)

Optional fields

- **Maximum number of lines per event**: In case of compression, this is the maximum number of lines or crystal pairs that can contribute to an event (default: 1)
- **Maximum ring difference**: This is used only for sensitivity image computation for list-mode data and when no normalization file is provided (default: the maximum for the PET scanner in use)
- **Calibration factor**: The calibration factor associated to the current scanner settings (default: 1.)
- **Attenuation correction flag**: Specify if the binary datafile contains attenuation correction factors (default: 0 = no ; 1 = yes)
- **Normalization correction flag**: Specify if the binary datafile contains normalization correction factors (default: 0 = no ; 1 = yes)
- **Scatter correction flag**: Specify if the binary datafile contains scatter correction factors (default: 0 = no ; 1 = yes)
- **Random correction flag**: Specify if the binary datafile contains random correction factors (default: 0 = no ; 1 = yes)
- **Isotope**: Isotope used during the acquisition (see the *config/misc/isotopes_pet.txt* file for the complete list), it is used for decay and branching ratio corrections (no correction if not specified)

6.1.2 Binary datafile

In order to have a generic implementation of the high-level parallelism using MPI, all events in a given datafile must have the same encoding size in bytes. This implies that in the case of compression (*i.e.* when the *Maximum number of lines per event* field is higher than 1), events including a number of lines less than this maximum number have to contain garbage bytes on the disk. This is illustrated in the tables below.

Tables 2, 3 and 4 describe the binary content of an event from a histogram, list-mode and normalization binary datafile, respectively. Note that little endian encoding only is supported. Whether an optional data field is present or not, the order of the fields provided in these tables must be observed. The meaning of the type *FLTNBDATA* is explained in section 3. The crystal IDs are linked to the PET geometry description used for the scanner associated to the datafile (see section 5.1).

Table 2: Mandatory/Optional fields of a PET histogram event

	Symbol	Description	Type	Mandatory
1	t	Time in ms	uint32_t	yes
2	a	Attenuation correction factor	FLTNBDATA	no
3	r	Un-normalized random intensity rate of the corresponding histogram bin (count/s)	FLTNBDATA	no
4	n	Normalization factor of the corresponding histogram bin	FLTNBDATA	no
5	p	Amount of data in the bin	FLTNBDATA	yes
6	s	Un-normalized scatter intensity rate of the corresponding histogram bin (count/s)	FLTNBDATA	no
7	k	Number of contributing crystal pairs	uint16_t	if *Maximum number of lines* > 1
	For [k]			
8.1	c1	Crystal ID1	uint32_t	yes
8.2	c2	Crystal ID2	uint32_t	yes
	For [r]	r = Maximum number of lines per event minus k		
9.1	0	Garbage	32bits	yes
9.2	0	Garbage	32bits	yes

Table 3: Mandatory/Optional fields of a PET list-mode event

	Symbol	Description	Type	Mandatory
1	t	Time in ms	uint32_t	yes
2	a	Attenuation correction factor	FLTNBDATA	no
3	s	Un-normalized scatter intensity rate of the corresponding event (count/s)	FLTNBDATA	no
4	r	Un-normalized random intensity rate of the corresponding event (count/s)	FLTNBDATA	no
5	n	Normalization factor of the corresponding event	FLTNBDATA	no
6	k	Number of contributing crystal pairs	uint16_t	if *Maximum number of lines* > 1
	For [k]			
7.1	c1	Crystal ID1	uint32_t	yes
7.2	c2	Crystal ID2	uint32_t	yes
	For [r]	r = Maximum number of lines per event minus k		
8.1	0	Garbage	32bits	yes
8.2	0	Garbage	32bits	yes

Table 4: Mandatory/Optional fields of a PET normalization event

	Symbol	Description	Type	Mandatory
1	a	Attenuation correction factor	FLTNBDATA	no
2	n	Normalization factor of the corresponding data channel	FLTNBDATA	no
3	k	Number of contributing crystal pairs	uint16_t	if *Maximum number of lines* > 1
	For [k]			
4.1	c1	Crystal ID1	uint32_t	yes
4.2	c2	Crystal ID2	uint32_t	yes
	For [r]	r = Maximum number of lines per event minus k		
5.1	0	Garbage	32bits	yes
5.2	0	Garbage	32bits	yes

6.2 SPECT parallel/convergent datafile format

By parallel/convergent SPECT data, we mean SPECT data linked to a parallel/convergent SPECT geometry as described in section 5.2. SPECT datafiles can hold histogram or list-mode data with associated corrections (the latter is not yet supported).

6.2.1 ASCII header

The content of the ascii header file is described below. It includes mandatory and optional fields. All keywords are case sensitive and the separator (between the field and the value) is the colon sign ":".

Mandatory fields

- **Scanner name**: The name of the SPECT parallel/convergent scanner (corresponding to a scanner in the *config/scanner* folder, without the file extension)
- **Data filename**: The absolute or relative path to the binary datafile
- **Number of events**: The number of CASToR events in the binary datafile
- **Data mode**: Can be "list-mode" or "histogram"
- **Data type**: SPECT
- **Start time (s)**: The relative start time of the acquisition in seconds
- **Duration (s)**: The duration of the acquisition in seconds
- **Number of projections**: The total number of projections in the acquisition (considering all the heads)
- **Projection angles**: Angles in degrees for each projection, separated by commas. For a multiheaded SPECT system, the angles of the 1^{st} head must be provided first, followed by the angles of the 2^{nd} head, etc..

  The list of angles corresponding to the system displayed in Fig. 10, must be:
  $ang_{h1,0}$, $ang_{h1,1}$, $ang_{h1,2}$, $ang_{h1,3}$, $ang_{h1,4}$, $ang_{h1,5}$, $ang_{h1,6}$, $ang_{h1,7}$, $ang_{h1,8}$, $ang_{h1,9}$, $ang_{h2,0}$, $ang_{h2,1}$, $ang_{h2,2}$, $ang_{h2,3}$, $ang_{h2,4}$, $ang_{h2,5}$, $ang_{h2,6}$, $ang_{h2,7}$, $ang_{h2,8}$, $ang_{h2,9}$

Optional fields

- **Number of bins**: The transaxial and axial number of bins, separated by a comma. (default: if number of bins are not provided in the datafile header, the transaxial/axial number of pixels as defined in the .geom scanner configuration file are used instead)
- **Head rotation direction**: Rotation direction of the head(s). Must be either clockwise (*CW*, defaut) or counter-clockwise *(CCW)*.
- **Calibration factor**: The calibration factor associated to the current scanner settings (default: 1.)
- **Normalization correction flag**: Specify if the binary datafile contains normalization correction factors (default: 0 = no ; 1 = yes)
- **Scatter correction flag**: Specify if the binary datafile contains scatter correction factors (default: 0 = no ; 1 = yes)
- **Isotope**: Isotope used during the acquisition (currently not taken into account)

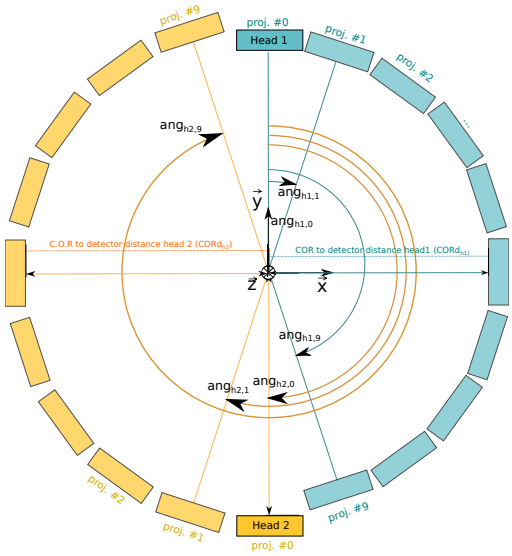


Figure 10: Projections angles conventions for a multi-headed SPECT system, in the regular (clockwise) orientation.

The two following fields are optional but if used, then only one must appear. If none is provided, then the global distance from camera surface to the center of rotation is taken from the SPECT system geometric description file, and assumed constant for all projections.

- **Global distance camera surface to COR**: Global distance in mm between the detector surface and the center of rotation (this distance must be constant for all projections)
- **Distance camera surface to COR**: Distance in mm between the detector surface and the center of rotation, for each projection angle (separated by commas). For a multiheaded SPECT system, the distances for each projection angle of the first head must be provided first, followed by the distances of each projection angles of the 2^{nd} head, etc..

  This field expect a number of entries equal to the total number of projection angles (for each head), even if the distances are constant. For example, the list of COR to detector distances for the system displayed in Fig. 10 must be:
  $CORd_{h1}$, $CORd_{h1}$, $CORd_{h1}$, $CORd_{h1}$, $CORd_{h1}$, $CORd_{h1}$, $CORd_{h1}$, $CORd_{h1}$, $CORd_{h1}$, $CORd_{h1}$, $CORd_{h2}$, $CORd_{h2}$, $CORd_{h2}$, $CORd_{h2}$, $CORd_{h2}$, $CORd_{h2}$, $CORd_{h2}$, $CORd_{h2}$, $CORd_{h2}$, $CORd_{h2}$

6.2.2 Binary datafile

Tables 5 and 6 describe the binary content of an event from a histogram and list-mode binary datafile, respectively. Note that little endian encoding only is supported. Whether an optional data field is present or not, the order of the fields provided in these tables must be observed. The meaning of the type *FLTNBDATA* is explained in section 3. The projection and bin IDs are linked to the SPECT parallel/convergent geometry description used for the scanner associated to the datafile (see section 5.2).

Table 5: Mandatory/Optional fields of a parallel/convergent SPECT histogram event

	Symbol	**Description**	**Type**	**Mandatory**
1	t	Time in ms	uint32_t	yes
2	p	Amount of data in the bin	FLTNBDATA	yes
3	s	Un-normalized scatter intensity rate of the corresponding histogram bin (count/s)	FLTNBDATA	no
4	n	Normalization factor of the corresponding sinogram bin	FLTNBDATA	no
5	v	Projection ID (angular view)	uint32_t	yes
6	c	Bin ID	uint32_t	yes

Table 6: Mandatory/Optional fields of a parallel/convergent SPECT list-mode event

	Symbol	**Description**	**Type**	**Mandatory**
1	t	Time in ms	uint32_t	yes
2	s	Un-normalized scatter intensity rate of the corresponding event (count/s)	FLTNBDATA	no
3	n	Normalization factor of the corresponding event	FLTNBDATA	no
4	v	Projection ID (angular view)	uint32_t	yes
5	c	Bin ID	uint32_t	yes

7 Image file format

CASToR uses the Interfile format for image reading and writing. The format consists in a binary image file associated with an ASCII header (text file) containing metadata about the image (image location, dimensions, voxel size, etc). The header being easy to read and edit without specific tools, this format is very flexible and can handle different type of images, given that the adequate keys are provided.

7.1 Supported keys

A limited number of keys are actually required to read/write an image in order to simplify the conversion from another image format to Interfile, and conversely. An Interfile key is based on key/value pairs such as *key := value*.

The header must be located in the same directory as the image file. The minimal interfile header must provide the image path, the image dimensions, the voxel size and the image encoding, as in this example:

Listing 1: Interfile header mandatory keys for reading

```
1  !name of data file := img_name.img
2  !total number of images := 47
3  imagedata byte order := LITTLEENDIAN
4  number of dimensions := 3
5  !matrix size [1] := 256
6  !matrix size [2] := 256
7  !matrix size [3] := 50
8  !number format := short float
9  !number of bytes per pixel := 4
10 scaling factor (mm/pixel) [1] := 4
11 scaling factor (mm/pixel) [2] := 4
12 scaling factor (mm/pixel) [3] := 4
13 image duration (sec) := 1
```

Below are listed the supported keys in the current version:

- "name of data file" : name of the image binary file
- "imagedata byte order" : endianness (must be LITTLEENDIAN or BIGENDIAN)
- "data starting block" : Data offset in block of 2048 bytes
- "data offset in bytes" : Data offset using a provided number of bytes
- "number format" : Type of each pixel/voxel data (must be 'long float', 'short float', 'signed integer', 'unsigned integer')
- "number of bytes per pixel" : Number of bytes for each pixel/voxel of the image data
- "matrix size [1]" : Width (number of voxels) (x-axis dimension)
- "matrix size [2]" : Height (number of voxels) (y-axis dimension)
- "matrix size [3]" : Slices (number of voxels) (z-axis dimension)
- "matrix size [4]" : number of (dynamic) time frames
- "scaling factor (mm/pixel) [1]" : Width voxel size
- "scaling factor (mm/pixel) [2]" : Height voxel size
- "scaling factor (mm/pixel) [3]" : Axial voxel size

- "slice thickness (pixels)" : slice thickness, assumed to be provided in mm regardless of the name. 'scaling factor (mm/pixel) [3]' has the priority over this field. If it is not provided, the axial voxel size is computed as (scaling factor (mm/pixel) [1] + scaling factor (mm/pixel) [2]) / 2.) * slice thickness (pixels)
- "number of time frames" : number of (dynamic) time frames. Priority over "matrix size [4]"
- "number of respiratory gates" : (CASToR key) number of respiratory gated images
- "number of cardiac gates" : (CASToR key) number of cardiac gated images
- "rescale slope" : calibration factor
- "quantification units" : calibration factor (cumulative to "rescale slope"). If this field provids unit (eg : kbq/cc), it is simply ignored
- "rescale intercept" : additive calibration value
- "study duration (sec)" : acquisition duration in seconds
- "image duration (sec)" : image duration in seconds
- Image positionning of offset related keys such as "origin (mm) [x]', 'offset [x]', 'first pixel offset (mm) [x]' are not supported in the current version, which assumes that all images are centered in the field of view.
- Image orientation related keys such as 'slice orientation', 'patient rotation', 'patient orientation' are ignored in the current version, as such information is most of the time missing and/or misleading.

7.2 Dynamic images

Dynamic images can be provided or written on disk through two formats: a metaheader associated to a series of Interfile images, and a single file containing all the frames/gated images. Default output option for dynamic images is the metaheader.

7.2.1 Metaheader

The metaheader is an ASCII file gathering main information about the dynamic images, the actual number of images, and the path to each one of them. Here is an example:

Listing 2: Interfile Metaheader mandatory keys for reading

```
number of time frames := 10
number of respiratory gates := 1
number of cardiac gates := 1
! total number of datasets:= 10
%data set [1]:={0,img_frm0.hdr,UNKNOWN}
%data set [2]:={0,img_frm1.hdr,UNKNOWN}
%data set [3]:={0,img_frm2.hdr,UNKNOWN}
%data set [4]:={0,img_frm3.hdr,UNKNOWN}
%data set [5]:={0,img_frm4.hdr,UNKNOWN}
%data set [6]:={0,img_frm5.hdr,UNKNOWN}
%data set [7]:={0,img_frm6.hdr,UNKNOWN}
%data set [8]:={0,img_frm7.hdr,UNKNOWN}
%data set [9]:={0,img_frm8.hdr,UNKNOWN}
%data set [10]:={0,img_frm9.hdr,UNKNOWN}
```

The metaheader must contain the following mandatory keys:

- "total number of datasets" : number of associated files

- "data set [xxx]" : path to each interfile image header associated with the metaheader, where xxx represents a number starting from 1 (not 0). It must be an array key whose second element contains the path. The 1st and 3rd elements, respectively 0 and UNKNOWN in the example) are ignored. The number of "data set [xxx]" must correspond to the value provided by the 'total number of datasets' keys
- "number of time frames" : Number of dynamic frames (not required if the data does not contain any)
- "number of respiratory gates" : (CASToR key) Number of respiratory gated images (not required if the data does not contain any)
- "number of cardiac gates" : (CASToR key) Number of cardiac gated images (not required if the data does not contain any)

The header of the associated image files must be similar to standard Interfile header file as described by the listing 2.

7.2.2 Merged dynamic interfile

The binary image file must contain all the frames and/or gated images concatenated into a single binary file. The associated header must contain all information for each frame/gated image included in the binary image file, as in the example below (listing 3). No information about the data offset position of each image is required.

By default, dynamic images will be written on disk as a metaheader associated with a separate interfile for each image. The command line option "-omd" must be used to enable the output of dynamic images in one single file.

Listing 3: Interfile header of a dynamic image

```
!name of data file := img_name.img
imagedata byte order := LITTLEENDIAN
number of time frames := 10
number of respiratory gates := 1
number of cardiac gates := 1

// Frame 1
imagedata byte order := LITTLEENDIAN
!matrix size [1] := 256
!matrix size [2] := 256
!matrix size [3] := 50
!number format := short float
!number of bytes per pixel := 4
scaling factor (mm/pixel) [1] := 4
scaling factor (mm/pixel) [2] := 4
scaling factor (mm/pixel) [3] := 4
image duration (sec) := 10
image start time (sec) := 0
// Frame 2
imagedata byte order := LITTLEENDIAN
!matrix size [1] := 256
!matrix size [2] := 256
!matrix size [3] := 50
!number format := short float
!number of bytes per pixel := 4
scaling factor (mm/pixel) [1] := 4
scaling factor (mm/pixel) [2] := 4
scaling factor (mm/pixel) [3] := 4
image duration (sec) := 10
image start time (sec) := 10
```

```
31
32  //(...)
33
34  // Frame 10
35  imagedata byte order := LITTLEENDIAN
36  !matrix size [1] := 256
37  !matrix size [2] := 256
38  !matrix size [3] := 50
39  !number format := short float
40  !number of bytes per pixel := 4
41  scaling factor (mm/pixel) [1] := 4
42  scaling factor (mm/pixel) [2] := 4
43  scaling factor (mm/pixel) [3] := 4
44  image duration (sec) := 10
45  image start time (sec) := 90
```

8 CASToR utilities

This part presents various utilities related to CASToR data processing, generation or conversion. Apart from the binary datafile converters, the tools executables are located in the same directory than the main reconstruction program, while the source can be found in the toolkits/ directory from the main CASToR repository. Compilation with CMake requires the two CASToR_BUILD_GATE_UTILITIES and CASToR_BUILD_SAMPLE_UTILITIES options to be enabled in order to generate the executables. The toolkits currently include:

- Datafile converter for Siemens Biograph 6 Truepoint system (Unix/Windows binary). Subsection 8.1.
- GATE conversion utilities, to be used to generate a CASToR generic geometry file (geom) from a GATE macro file containing a *cylindricalPET* or *ecat* geometry and a CASToR datafile from a root file generated by GATE for these geometries. Subsection 8.2.
- *castor-PetScannerLutEx*, an example of a program which generates a geometric file containing a PET Look-Up-Table with information about the scanner elements, readable by CASToR. Subsection 8.3.
- *castor-datafileConversionEx*, an example of a program providing guidances on how to convert a datafile to the CASToR format. Subsection 8.4.

8.1 Binaries for manufacturer datafile conversion

Several executables able to convert manufacturer's datasets to a CASToR datafile may be provided on the CASToR website (see http://www.castor-project.org/converters) . For now, only a converter for the Siemens Biograph 6 Truepoint is available), as binary file (*i.e.* no sources available), accompanied with a small manual.

Note that all these converters and associated materials are not supported nor validated by any manufacturer. Note also that the CASToR software associated or not with these converters is not approved for any medical use. The CASToR developers and collaboration members cannot be responsible for any consequences resulting from the use of any tool distributed on the CASToR website.

8.2 GATE utilities

The GATE utilities use the macro files of a GATE simulation to generate the scanner geometry (as a CASToR .geom file 5.1.1), and CASToR datafile (6.1). These utilities currently have several limitations regarding the nature of the systems which can be converted:

- Supported GATE systems include *cylindricalPET*, *ecat*, and *SPECThead* systems.
- For PET systems, it is assumed that the first hierarchic level element (i.e rsector for *cylindricalPET* and block for *ecat*) use a ring repeater, while the other elements use either cubic or linear repeaters.
- For SPECT systems, the specificities of the collimator aren't implemented.
- Supported GATE output format is root.
- Command positionning the first hierarchic level element (i.e: */gate/rsector/placement/setTranslation*) must have a single non-zero component.
- The script will look into the provided macro file, and the .mac file called from the given macro file, for GATE commands characterizing the system. Lines starting with a comment character (#) are ignored.

There are currently two utilities: *castor-GATERootToCastor* and *castor-GATEMacToGeom*.

The *castor-GATERootToCastor* program could be used to convert a ROOT list-mode datafile generated with GATE[8] to a CASToR datafile.

Usage:

```
castor-GATErootToCastor(.exe) -i   path/to/ifile.root
                                   (or)
                              -il  path/to/ifile.txt
                              -o   path/to/out/file
                              -m   path/to/macrofile.mac
                              -s   scanner_alias
```

Main options:

- **-i** *path/to/file.root* : give an input root datafile to be converted.
- **-il** *path/to/file.txt* : give an input text file containing a list of root files to be converted. The path to each datafile to be converted must be entered on a newline. They will be concatenated into one single CASToR datafile.
- **-m** *path/to/macrofile.mac* : give the input GATE macro file used for the GATE simulation.
- **-o** *path/to/out/file* : give the output file base name (no default).
- **-s** *scanner_alias* : provide the name of the scanner used to acquire the original data. It must correspond to a .geom or .hscan file in the *config/scanner* repository.

Optional settings:

- **-t** : only the true coincidences will be converted.
- **-oh** : Specify if the output datafile must be written in histogram format (default : list-mode).
- **-atn** *path/to/atn/image* : For histogram output, provide an attenuation image (cm-1) related to the acquisition. Analytic projection will be performed during the data conversion in order to estimate attenuation correction factors (acf) for each histogram event.
- **-k** : For list-mode output, write the kind of coincidence (true/scatter/rdm/...) in the output datafile (disabled by default).
- **-ist** *isotope_alias* : provide alias of the isotope used during the simulation. Supported PET isotopes and their parameters are listed in config/misc/isotopes_pet.txt file. New isotopes could be added in this file.
- **-isrc** *path/to/img:dims* : provide name and dimensions (separated by a colon) of an image generated with the GATE sourceID values (emission positions of each detected event). The option must start with the path to the output image which will be generated. Dimensions and voxel sizes (in mm) of the image must be provided using commas as separators, as follows: *path/to/image:dimX,dimY,dimZ,voxSizeX,voxSizeY,voxSizeZ*.
- **-geo** : Generate a CASToR geom file from the provided GATE macro file(s). A geom file with the *scanner_alias* (as defined with the -s option) basename will be generated in the scanner repository (default location : *config/scanner*).
- **-sp_bins** nbinsT,nbinsA : Option specific to simulation using SPECThead systems, with root output. Give transaxial and axial number of bins for projections, separated by a comma. Pixel sizes will be computed from the crystal surface and the transaxial/axial number of bins. *scanner_alias* (as defined with the -s option) basename will be generated in the scanner repository (default location : *config/scanner*).

- **-ot** list: Provide a series of time offsets in seconds to apply before each input file (in the case of converting several datafiles of a dynamic acquisition, timestamps of events may be resetted for each files. This variable allows to manually increment the time between each datafile(s) if required. The number of time offsets must be equal to the number of input files, provided by *-i* or *-if* options. 'list' is a list of time offsets, separated by commas ','.

Table 7 lists currently supported GATE macro commands. Non-supported commands include some geometric operations as well as informations regarding the source (isotope alias) which must be set manually. For SPECT systems, the conversion does not handle the collimator configuration.

Table 7: Supported GATE commands

Command	Note
/gate/.../placement/setTranslation	... = *rsector, block, head, layers* alias
/gate/.../ring/setRepeatNumber	... = *rsector, block, head* alias
/gate/.../cubicArray/setRepeatNumberX	... = *module, submodule, crystal, layer, block, pixel* alias
/gate/.../cubicArray/setRepeatNumberY	... = *module, submodule, crystal, layer, block, pixel* alias
/gate/.../cubicArray/setRepeatNumberZ	... = *module, submodule, crystal, layer, block, pixel* alias
/gate/.../ring/setRepeatNumber	... = *block, head* alias
/gate/.../linear/setRepeatNumber	... = *block* alias
/gate/.../ring/setFirstAngle	... = *rsector, block, head* alias
/gate/.../geometry/setXLength	... = *module, submodule, crystal, layers, block, pixel* alias
/gate/.../geometry/setYLength	... = *module, submodule, crystal, layers, block, pixel* alias
/gate/.../geometry/setZLength	... = *module, submodule, crystal, layers, block, pixel* alias
/gate/.../cubicArray/setRepeatVector	... = *module, submodule, crystal, layers, block, pixel* alias
/gate/application/setTimeStart	
/gate/application/setTimeStop	
CylindricalPET & ECAT systems	
/gate/systems/cylindricalPET/rsector/attach	
/gate/systems/cylindricalPET/module/attach	
/gate/systems/cylindricalPET/submodule/attach	
/gate/systems/cylindricalPET/crystal/attach	
/gate/systems/cylindricalPET/.../attach	... = *layer* alias (0,1,2,...)
/gate/systems/ecat/block/attach	
/gate/systems/ecat/crystal/attach	
/gate/application/addSlice	
/gate/digitizer/Coincidences/minSectorDifference	
/gate/.../cubicArray/setAngularSpan	... = *rsector, block* alias
/gate/.../ring/setModuloNumber	... = *rsector, block* alias
/gate/.../ring/setZShift	... = *rsector, block* alias
SPECThead system	
/gate/systems/SPECThead/base/attach	
/gate/systems/SPECThead/crystal/attach	
/gate/systems/SPECThead/pixel/attach	
/gate/application/setTimeSlice	
/gate/application/addSlice	
/gate/.../ring/setAngularPitch	... = *head* alias
/gate/.../moves/insert	... = *head* alias
/gate/.../setSpeed	... = *head* alias
/gate/.../setPoint2	... = *head* alias
/gate/.../ring/setAngularPitch	... = *head* alias

The *castor-GATEMacToGeom* is a very basic utility to generate a geom file from a GATE[8] mac file defining a *cylindricalPET*, *ecat*, or *SPECThead* geometry. The script assumes that at least the *rsector* and *crystal* levels are defined for a cylindricalPET, *block* and *crystal* for an ecat system, and *crystal* for a SPECThead system. Note that the geom file creation can also be performed using the *castor-GATERootToCastor* script with the *-geo* option (the geometry file will be created in the scanner repository before the datafile conversion).

Usage:

```
castor-GATEMacToGeom(.exe) -m path/to/mac/file
                           -o scanner_alias
```

where *-m* and *-o* indicate the path to the GATE macro file and the basename of the output *.geom* file that will be generated in the scanner repository (default location: *config/scanner/*) respectively.

8.3 castor-PetScannerLutEx

This program is a C++ template utility whose aim is to help developers in generating a PET user-made geometry Look-Up-Table (LUT) readable by CASToR, as described in section 5.1.2. The file presents the generation of a LUT for a GATE model of the GE Discovery RX system. The code could be edited to generate any kind of PET geometry.

Usage: `castor-PetScannerLutEx(.exe) -alias system_alias`

The script's single argument *-alias* recovers the base name to be given to the LUT files. These files (an ASCII header with the .hscan extension, and the binary file containing the LUT with the .lut extension) will be written on disk in the scanner repository (default location: *config/scanner*).

8.4 castor-datafileConversionEx

The purpose of the *castor-datafileConversionEx* is to provide guidance for the conversion process of the datafile from the original manufacturer/simulator indexation to the CASToR indexation. It does not perform any conversion by itself and must be adjusted to the conversion of any system dataset. It implements the required calls to CASToR Datafile and Event objects and functions in order to write a dataset in the CASToR format. It also provides directions regarding the incorporation of correction factors for each event as defined in the CASToR datafile (section 6.1 for PET).

As an example, the script performs the conversion for a GATE simulated PET system (it expects root datafiles, therefore requires compilation with the ROOT library, as described in section 3).

Usage:

```
castor-datafileConversionEx(.exe) -ih path/to/histo/datafile
                                      (or)
                                  -il path/to/listm/datafile
                                  -o  path/to/out/file
                                  -s  scanner_alias
```

Main options:

- **-ih** *path/to/histo/datafile* : give an input histogram datafile to convert
- **-il** *path/to/listm/datafile* : give an input list-mode datafile to convert
- **-o** *path/to/cstr/file.cdh* : give the path to the output file will be created inside this folder (no default)

- **-s** *scanner_alias* : provide the name of the scanner used for to acquire the original data. It must correspond to a .geom or .hscan file in the config/scanner repository.

Optional settings:

- **-nc** *path/to/norm_factors_file* : provide a file containing normalization correction factors
- **-sc** *path/to/scat_factors_file* : provide a file containing scatter correction factors
- **-rc** *path/to/rdm_factors_file* : provide a file containing random correction factors
- **-ac** *path/to/atn_factors_file* : provide a file containing attenuation correction factors
- **-cf** *calibration factor* : provide a calibration factor in FLTNB type (float or double, see section 3.1).
- **-ist** *isotope_alias* : provide alias of the isotope used in the input datafile. Supported PET isotopes and their parameters are listed in config/misc/isotopes_pet. New isotopes could be added in the same file.

References

[1] R. L. Siddon, "Fast calculation of the exact radiological path for a three-dimensional CT array," *Medical Physics*, vol. 12, no. 2, pp. 252–255, 1985.

[2] F. Jacobs, E. Sundermann, B. De Sutter, M. Christiaens, and I. Lemahieu, "A fast algorithm to calculate the exact radiological path through a pixel or voxel space," *CIT. JOURNAL OF COMPUTING AND INFORMATION TECHNOLOGY*, vol. 6, no. 1, pp. 89–94, 1998.

[3] P. M. Joseph, "An Improved Algorithm for Reprojecting Rays through Pixel Images," *IEEE Transactions on Medical Imaging*, vol. 1, no. 3, pp. 192–196, Nov 1982.

[4] L. A. Shepp and Y. Vardi, "Maximum Likelihood Reconstruction for Emission Tomography," *IEEE Transactions on Medical Imaging*, vol. 1, no. 2, pp. 113–122, Oct 1982.

[5] K. V. Slambrouck, S. Stute, C. Comtat, M. Sibomana, F. H. P. van Velden, R. Boellaard, and J. Nuyts, "Bias Reduction for Low-Statistics PET: Maximum Likelihood Reconstruction With a Modified Poisson Distribution," *IEEE Transactions on Medical Imaging*, vol. 34, no. 1, pp. 126–136, Jan 2015.

[6] C. Byrne, "Iterative algorithms for deblurring and deconvolution with constraints," *Inverse Problems*, vol. 14, no. 6, pp. 1455–1467, 1998.

[7] L. Landweber, "An Iteration Formula for Fredholm Integral Equations of the First Kind," *American Journal of Mathematics*, vol. 73, pp. 615–624, 1951.

[8] S. Jan, G. Santin, D. Strul, S. Staelens, K. Assi, D. Autret, S. Avner, R. Barbier, M. Bardis, P. M. Bloomfield, D. Brasse, V. Breton, P. Bruyndonckx, I. Buvat, A. F. Chatziioannou, Y. Choi, Y. H. Chung, C. Comtat, D. Donnarieix, L. Ferrer, S. J. Glick, C. J. Groiselle, D. Guez, P.-F. Honore, S. Kerhoas-Cavata, A. S. Kirov, V. Kohli, M. Koole, M. Krieguer, D. J. van der Laan, F. Lamare, G. Largeron, C. Lartizien, D. Lazaro, M. C. Maas, L. Maigne, F. Mayet, F. Melot, C. Merheb, E. Pennacchio, J. Perez, U. Pietrzyk, F. R. Rannou, M. Rey, D. R. Schaart, C. R. Schmidtlein, L. Simon, T. Y. Song, J.-M. Vieira, D. Visvikis, R. V. de Walle, E. Wiers, and C. Morel, "GATE: a simulation toolkit for PET and SPECT," *Physics in Medicine and Biology*, vol. 49, no. 19, pp. 4543–4561, 2004.

[9] R. Brun and F. Rademakers, "ROOT - An Object Oriented Data Analysis Framework," *Nucl. Inst. & Meth. in Phys. Res. A*, vol. 389, pp. 81–86, 1997.