CMS Internal Note

The content of this note is intended for CMS internal use and distribution only

3rd October 2007

The Tracker Geometry Validation Procedure

Riccardo Ranieri

CERN, Geneva, Switzerland

Abstract

This is the description of the Tracker geometry validation procedure which is followed whenever a new CMSSW software prerelease or release is announced and published.
The Tracker geometry validation consists of two parts: the check of the active volume geometric center positioning, local axes orientations and detector identity number detid, which define the detector in CMSSW, and the comparison of the Tracker material budget in terms of radiation length and nuclear absorption length.
There have been developed many software tools to write, debug and analyse the Tracker geometry simulation. The usage of these tools is explained with some examples.

1 Why do we need a Tracker geometry validation?

The silicon Tracker [1] of the CMS experiment [2] at LHC [3] is the largest all silicon tracking detector ever built. It consist of about 200 m^2 of silicon microstrip detectors and 1 m^2 of silicon pixel detectors. A total 15 148 microstrip and 1 440 pixel modules are assembled in several substructures for a total of about 10 million and 66 million readout channels, respectively. To reach the best tracking performance a detailed description of the positioning of the detectors is demanded.
The CMS simulation software is based on Object Oriented technology using the toolkit GEANT4 [4] and CMS has chosen to provide the geometry description using the Detector Description Language (DDL) [5]. The algorithms build the volume tree as required by GEANT4 and position them appropriately by providing the correct translation and rotation matrices. Each parametrisation within the DDL has a unique name, associated to a C++ class. The shared library built with the C++ classes is loaded on demand. Each class is often associated with a list of parameters which needs to be supplied with xml (eXtensible Markup Language) files. The C++ class has essentially two methods, one to initialise itself by loading the parameters given in the xml file and the other to execute the parametrization. During the process of execution, DDL solids, logical or position parts can be created. The design of the geometry description of the tracker follows some rules to speed-up the parsing of the xml files and the initialization of the GEANT4 library:

- break up the content of a sub-detector into a number of sub-components going to the required granularity (example: Tracker Outer Barrel (TOB) Fig. 1 → a layer Fig. 2 → rods grouping modules Fig. 3 → microstrip modules Fig. 4);
- try to avoid duplication in defining solids or logical parts;
- use of constant names and simple expressions in the xml files to understand the meaning of a set of numbers;
- use average materials in the passive part unless there are localisations of some dense material (like aluminium blocks, cooling pieces) which are then separately described in detail;
- use algorithms in positioning objects if a suitable correlation is found (example: in a microstrip module, the position of electronics, frame and silicon sensor are heavily correlated; wheel-inserts in the outer microstrip end-caps carbon fibre frame are correlated with the positioning of the modules in the petal of the wheel).

These considerations make the geometrical description better readable, more easily maintainable and reusable.

A careful implementation of the engineering drawings is requested. In addition, the large number of channels implies a substantial passive material to readout, power and cool the electronics. This causes multiple scattering, nuclear interactions, electron bremsstrahlung and photon conversions, that in turn demand an accurate evaluation of the passive material budget. Each single part, either sensitive or passive, is weighed and carefully simulated. The accuracy of the material budget estimate and its distribution is checked using engineering drawings, photographs and weighting large assembled parts.
The simulation of the Tracker geometry is included in the CMS software framework `CMSSW` [6], the packages `Geometry/TrackerCommonData` and `Geometry/CMSCommonData` contain the xml files and the classes to define the Tracker geometry which is used to simulate and reconstruct pp collisions. The definition and documentation of the passive material definitions is stored in `Geometry/TrackerCommonData/data/Materials`.
The development and maintaining of the Tracker geometry software is as demanding as the implementation of the material and volume definitions in GEANT4. To help the software developers in the task of reviewing the Tracker material budget and updating the module positioning, software tools and have ben developed within the `CMSSW` framework.
The description of the tools available to write, test and check the Tracker geometry is the subject of the Sec. 2 and 3, regarding the active volumes properties and the definition of the composite materials respectively. The software tools developed are used to check in an automated way the Tracker geometry when a new `CMSSW` prerelease or release version is published. The Tracker geometry validation procedure is described in Sec. 4. In a phase of software development, the validation is useful to keep the geometry updates under control, then in a second phase of software maintenance the validation will serve to guarantee a fully compatible Tracker geometry in the passage between a `CMSSW` version to the following one.

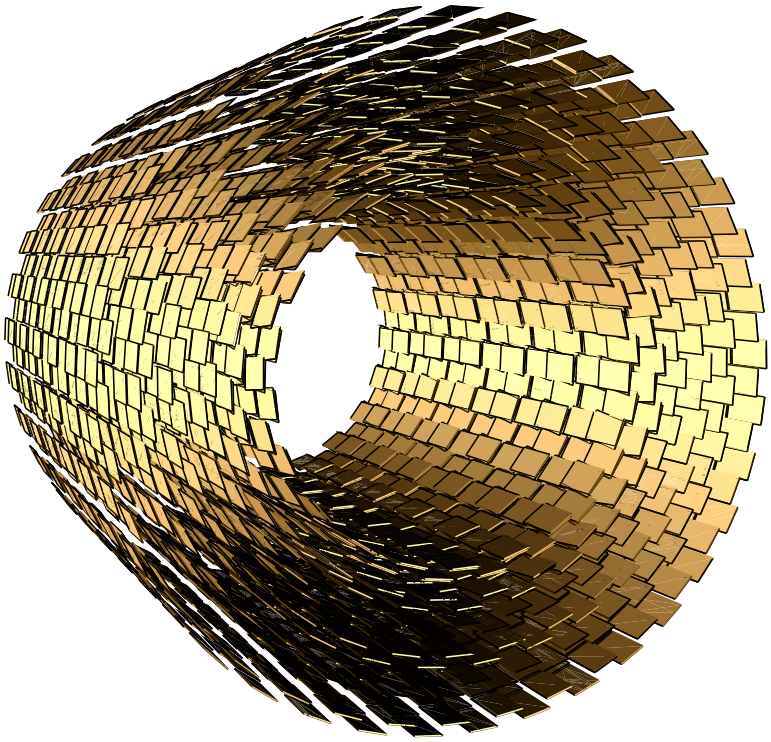


Figure 1: The Tracker Outer Barrel (TOB) active volumes.

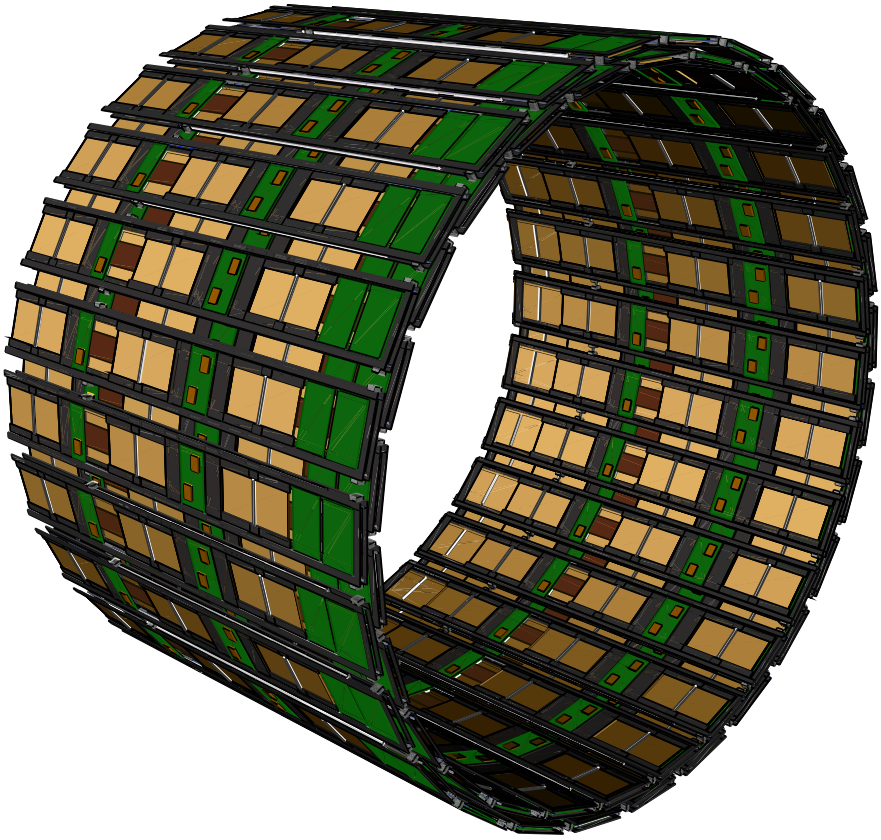


Figure 2: The third layer of the TOB.

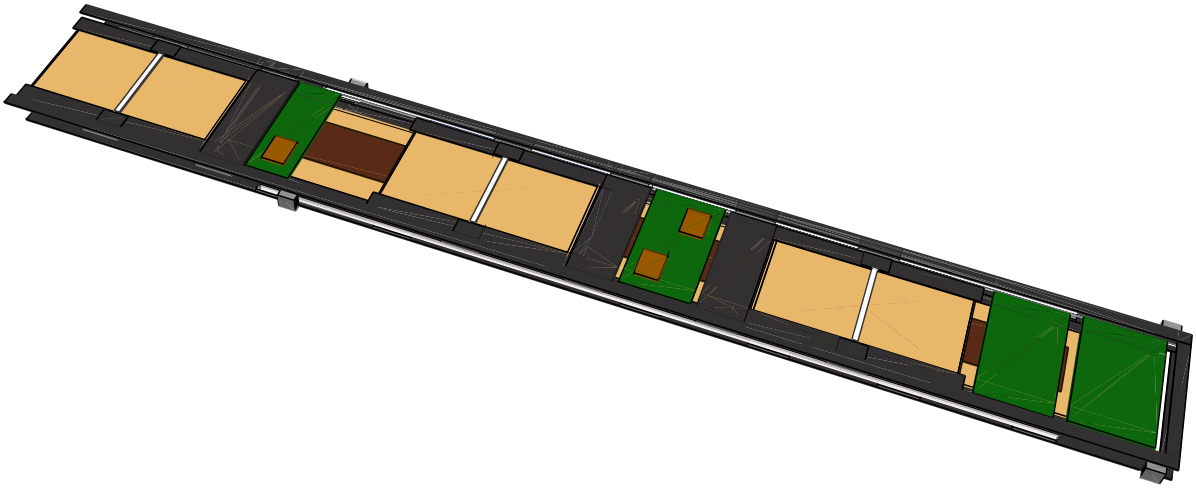


Figure 3: A TOB rod.

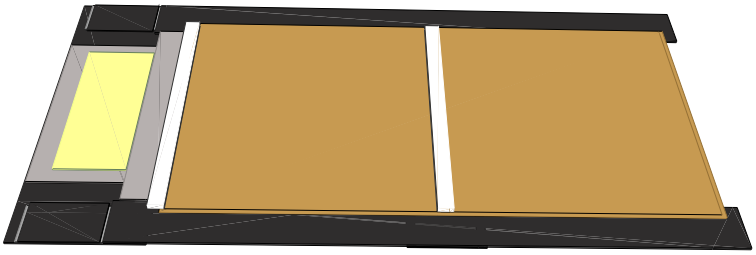


Figure 4: A silicon microstrip detector of the TOB.

2 Check of the active volume positioning

Once all the active volumes positioning and orientation have been frozen and a detid procedure is fixed, the geometry has to be checked each time a new CMSSW prerelease is published. To do it, some classes have been written to perform checks on the geometrical properties of the Tracker active volumes:

- positioning and orientations;
- detectors detid.

2.1 Active volumes positioning and orientations

The detector active volumes are defined as the GEANT4 volumes in which the simulated charged particles crossing the Tracker detector release the energy, whose amount is used to simulate the detector response and the front-end output signals.
The active volumes, being GEANT4 volumes, are defined by:

- the shape
- the material
- the coordinates of the geometrical center O of the volume (position)
- the components of the right-handed local reference frame (x, y, z) centered in O (rotation matrix)

The local reference frame is defined as following [7]:

z **axis** perpendicular to the sensitive surface, pointing outwards from the backplane to the front surface.

y **axis** along the longer side of the active volume surface (for the microstrip modules the convention for the direction of the y axis is such that the hybrid is on $y < 0$ side).

x **axis** defined to guarantee the right-handedness of the local (x, y, z) reference frame (for the microstrip modules the strip number is given by the strip local x coordinate and the strip pitch).

The position and orientation of the active volumes are compared with the engineers' projects to match the drawings of the Tracker detector. For the final version of the Tracker geometry, all these numbers will be frozen and will not change in the future. The global CMS coordinate system [8] is used throughout the text to express the coordinates or the component of the vectors: the x axis is horizontal and points approximately towards the center of the LHC, the y axis is approximately vertical and the z axis is along the beams.
The list of all the Tracker active volumes with their properties is computed by the C++ class `ModuleInfo.cc` published in the `test` directory of the package `Geometry/TrackerGeometryBuilder`. To create the output file `ModuleInfo.log` is enough to run

```
cmsRun trackerModuleInfo.cfg
```

from `Geometry/TrackerGeometryBuilder/test`.
The structure of the output file appears as the following:

```
******** raw Id = 369158508 (00010110000000001110100101101100)
        nav type = (0,0,0,3,3,1,42,1,3,0,0,)
TIB+   Layer 3 ext     string 22        module 3 TIBActiveRphi2
son of tibmodule2:TIBModule2 -425.252219        88.393991       527.901000
      volume 2.298 cm3       density 2.330 g/cm3     weight 0.005354 kg
thickness 320 um        active area 71.81 cm2
      Active Area Center
       O = (-425.2522,88.3940,527.9010)
              polar radius 434.3420    phi [deg] 168.2576      phi [rad] 2.9366
      Active Area Rotation Matrix
       z = n = (-0.9989,0.0478,-0.0000)
       [Rec] = (-0.9989,0.0478,-0.0000)
       x = t = (0.0478,0.9989,0.0000)
       [Rec] = (0.0478,0.9989,0.0000)
       y = k = (0.0000,0.0000,-1.0000)
       [Rec] = (0.0000,0.0000,-1.0000)
```

In the first line the detid (`raw Id`) is written both in decimal and binary base. For details about the detid definition see next paragraph 2.2. The `nav type` is the GEANT4 path one should follow in the hierarchy of the physical volumes to identify the volume itself (in this case the active volume). From the example taken from the `CMSSW_1_5_1` version of the Tracker geometry, the `nav type` of this active volume corresponds to:

0 First and only copy of the CMS mother volume OCMS (read the numbers as in a C++ code: 0 is the first, 1 the second, etc. etc.).

0 First and only daughter of the CMS mother volume CMSE.

0 First daughter of the CMSE volume: Tracker.

3 Fourth daughter of Tracker: TIB.

3 Fourth daughter of TIB: TIBLayer2 (corresponding to the third layer).

1 Second daughter of TIBLayer2: TIBLayer2Up (the external side of a TIB layer cylinder).

42 Forty-third daughter of TIBLayer2Up: TIBString2Up1 (the twenty-second string of the external side of the TIB Layer 3).

1 Second daughter of TIBString2Up1: TIBString2UpPls1 (the $z > 0$ part of the TIB layer 3 cylinder, commonly referred to as TIB+).

3 Fourth daughter of TIBString2UpPls1: TIBModule2 (the third module among the daughters of the string).

0 First daughter of TIBModule2: TIBWaferRphi2 (the silicon sensor).

0 First daughter of TIBWaferRphi2: TIBActiveRphi2 (the active volume of the silicon sensor, excluding the inactive wafer edges, the interface between the two sensors of the thick silicon modules and, from CMSSW_1_7_0, the backplane).

Then a brief description of the active volume is printed followed by the x, y, z coordinates (in mm) and some characteristics (volume, density, weight and thickness of the silicon active volume).
The coordinates of the geometrical center O of the active volume volume are then repeated (in mm) together with the values of the polar radius $R_{xy}= \sqrt{x^2 + y^2}$ and the azimuth angle ϕ in degrees and radians.
At the end the three versors defining the rotation matrix are listed. They are obtained in two different ways: as rotation matrix of the GeometricDet deep components of the IdealGeometryRecord (simulation-oriented) and as versors of the surface of the GeomDet corresponding to the GeometricDet detid and taken from the TrackerDigiGeometryRecord (reconstruction-oriented and labelled as [Rec]). Needless to say that the two expression of the three components of each of the three local axes must correspond independently from the way their values were accessed.

2.2 Detectors Numbering

The "detector identity number", referred to as detid, is a 32-bit integer value, filled according to fixed rules for the different Tracker sub-detectors. It defines univocally each CMS detector component.
The rules to construct the detid for the microstrip Tracker sub-detectors are defined in [9]. The Tracker numbering code is built within the package Geometry/TrackerNumberingBuilder for all the Tracker sub-detectors. A C++ class ModuleNumbering.cc is in the test directory to check if the microstrip numbering scheme is properly built. It can be run from Geometry/TrackerNumberingBuilder/test typing:

```
cmsRun trackerModuleNumbering.cfg
```

This class performs the complementary job with respect to the numbering scheme algorithm. First it constructs the detid's according to the rules, then for each detid it searches for the corresponding detector. After that, it loops on the detid's and checks if the detectors are properly ordered. A text file ModuleNumbering.log is produced with the description and the report of all the checks performed with the percentage of successes and failures.
The tail of the output file after a successfull running of the Tracker microstrip numbering scheme appears like the following:

```
-------------------------------------
Module Numbering Check Summary
-------------------------------------
Number of checks:   18636
             OK:    18636 (100.00%)
         ERRORS:        0 (0.00%)
-------------------------------------
```

3 Material Budget

3.1 Composite Material definition

All the Tracker materials are defined in some xml files [5] collected in the Geometry/TrackerCommonData package and in the file Geometry/CMSCommonData/data/materials.xml. Most of the Tracker materials are mixtures of pure or mixed materials.
The mixtures are defined with a fortran program mixture.f published in CMSSW in Geometry/TrackerCommonData/data/Materials [10]. The input files with predefined materials (pure elements or mixtures), called pure_materials.input and mixed_materials.input, contain the list of the materials with their properties (coherently with the GEANT4 materials defined in the xml files). The format of the two files is: material name, atomic weight, atomic number, density (g/cm^3), radiation length X_0 (cm) and

nuclear absorption length λ_0 (cm).
The mixtures are defined in several input files, as for example `TOB_module.in`. The input file contains comments and documentation of the materials defined for the different volumes of the Tracker together with the lines used by `mixture.f` to calculate the density and composition of a mixture.
To define a new mixture, the syntax to be adopted is the following:

- start new mixtures with a # in the first column;
- start the components with a * in the first column;
- do not start any comment with # or *.

To declare a mixture: small comment on the mixture, name of the mixture (coherently with the name assigned in the xml files), the volume (in cm^3) of the shape filled with the mixture, shape area (in cm^2, not really needed). To declare the items in the compound: progressive item number, small comment, material (one of the materials defined in `pure_materials.input` or `mixed_materials.input`), volume (in cm^3), multiplicity, category. The category can be one of the following: SUP for support, SEN for sensitive volumes, CAB for cables, COL for cooling, ELE for electronics (this distinction is used to attribute the fraction of radiation length and nuclear absorption length to the different categories).
This is an example of a material defined in `TOB_module.in`, called TOB_sid_rail2:

```
.....................................................................
 #    "TOB side rail passive rphi"  "TOB_sid_rail2"       2.856    35.7
 * 1  "CF leg"                      "Carbon fibre str."   2.29136  1     SUP
 * 2  "Insulator Kapton"            "T_Kapton"            0.287    1     ELE
 * 3  "Insulator Copper"            "Copper"              0.0046   1     ELE
.....................................................................
=====================================================================
```

It is the material of one of the carbon fibre legs of the TOB modules frame, whose dimensions are $(1.7 \times 21 \times 0.08)$ cm^3.
The properties of the compound are calculated by `mixture.f`. The information needed to define the new material in `CMSSW` are the normalised density $\hat{\rho}$ and the weight fraction of each item of the compound. The normalised density is defined as:

$$\hat{\rho} = \sum_i \rho_i \frac{V_i}{(\sum_i V_i)} \tag{1}$$

where ρ_i and V_i are the density and the volume of the i-th item of the compound and the sums are performed over all the items.
The `mixture.f` program calculates also X_0 and λ_0 of the material together with the total weight of the volume.
To run it, compile the fortran

```
make mixture
```

and run with an input file (without the extension `.in`)

```
./mixture TOB_module
```

The following output files are produced:

`tob_module.tex` a LaTeX file with the definitions of all the compounds contained in the `tob_module.in` input file;

`tob_module.x0` the file with the list of each compound with the fraction of the radiation length X_0 due to the different material categories defined above (SUP, SEN, CAB, COL, ELE);

`tob_module.l0` the file with the list of each compound with the fraction of the nuclear absorption length λ_0 due to the different material categories defined above (SUP, SEN, CAB, COL, ELE);

tob_module.titles the list of the compounds showing their density, the number of components with the volume fraction;

do a script to compile the LaTeX output file.

The table of the mixture TOB_sid_rail2 defined above summarizes the properties of the compound:

TOB side rail passive rphi (Material name: TOB_sid_rail2)

	Component	Material	Volume [cm^3]	%	Weight [g]	%	Density [g/cm^3]	X_0 [cm]	%	λ_0 [cm]	%
1	CF leg	Carbon fibre str.	2.291	88.711	3.872	89.734	1.690	25.000	87.319	45.414	89.864
2	Insulator Kapton	T_Kapton	0.287	11.111	0.402	9.311	1.400	28.400	9.628	53.294	9.591
3	Insulator Copper	Copper	0.005	0.178	0.041	0.955	8.960	1.435	3.053	15.056	0.544

Mixture density [g/cm^3]	1.67072
Norm. mixture density [g/cm^3]	1.51100
Mixture Volume [cm^3]	2.58296
MC Volume [cm^3]	2.85600
MC Area [cm^2]	35.70000
Normalization factor	0.90440
Mixture X_0 [cm]	24.60784
Norm. Mixture X_0 [cm]	27.20909
Norm. Mixture X_0 (%)	0.29402
Mixture λ_0 [cm]	46.00433
Norm. Mixture λ_0 [cm]	50.86736
Norm. Mixture λ_0 (%)	0.15727
Total weight (g)	4.31541

X_0 contribution	
Support:	0.873
Sensitive:	0.000
Cables:	0.000
Cooling:	0.000
Electronics:	0.127

λ_0 contribution	
Support:	0.899
Sensitive:	0.000
Cables:	0.000
Cooling:	0.000
Electronics:	0.101

From the table the following lines are then included in the CMSSW file `Geometry/TrackerCommonData/data/tobmaterial.xml` where the material TOB_sid_rail2 is defined:

```
<CompositeMaterial name="TOB_sid_rail2"
density="1.51100*g/cm3" symbol=" " method="mixture by weight">
 <MaterialFraction fraction="0.89734">
   <rMaterial name="materials:Carbon fibre str."/>
 </MaterialFraction>
 <MaterialFraction fraction="0.09311">
   <rMaterial name="materials:T_Kapton"/>
 </MaterialFraction>
 <MaterialFraction fraction="0.00955">
   <rMaterial name="materials:Copper"/>
 </MaterialFraction>
</CompositeMaterial>
```

The density is the normalised mixture density of Eq. 1 aken from the summary table with the weight fractions.

3.2 Material Budget analysis

The CMSSW package `Validation/Geometry` contains the classes and the configuration files to run the material budget analysis.
The package provides the access to the GEANT4 steps of the trajectories of the particles crossing the volumes of the CMS geometry. A file with 100 000 single neutrinos simulated homogeneously from the nominal CMS primary vertex $(0, 0, 0)$ with $|\eta| < 5$ can be downloaded from the link `Validation/Geometry/data/download.url`.
The CMSSW configuration files are in `Validation/Geometry/test`, all of them using the *dummy physics* configuration to switch-off all the interactions with matter in GEANT4, thus allowing to follow the unperturbed trajectory of the simulated particles throughout the volumes listed in the configuration parameter SelectedVolumes.

The output file of the material budget analysis is a ROOT [11] file containing all the profiles of X_0 and λ_0 as a function of η, φ, (η, φ) and (z, R_{xy}) for the chosen volumes. Optionally a second ROOT file can be filled with the details of all the GEANT4 steps and volumes crossed by the simulated particle organized in a ROOT tree.
The Tracker analysis is specialised in the class `MaterialBudgetTrackerHistos`, the X_0 and λ_0 profiles are computed separately for each of the material categories assigned during the compound definition (SUP, SEN, CAB, COL, ELE, see Sec. 3.1), for the volumes filled with air (category: AIR) and for the other materials for which the assignment has not been made yet (category: OTH). The lists of the category fraction of X_0 and λ_0 for each material are in the files `trackerMaterials.x0` and `trackerMaterials.l0`, obtained chaining all the `.x0` and `.l0` files from the `mixture` executable after running with all the input files stored in `Geometry/TrackerCommonData/data/Materials`.
To run all the Tracker analysis and create the plots, create a directory `Images` in `Validation/Geometry/test` and launch the command

```
./go
```

or

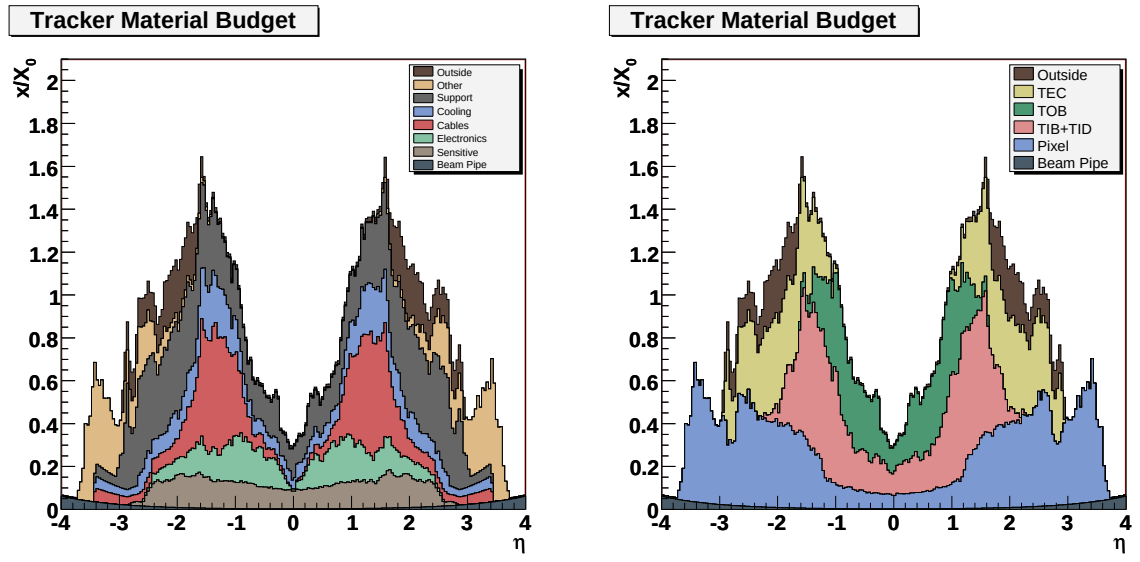
```
./go_debug
```

to store the text output of the analysis into files. The validation analysis is run with all the Tracker configuration files, corresponding to the different sub-detectors or macro-structures: pixel barrel, pixel end-cap $z > 0$, pixel end-cap $z < 0$, TIB, forward TID, backward TID, TIB-TID services, TOB, TEC, Tracker structures (support tube, thermal screen, patch panel), beam pipe, full Tracker geometry. The ROOT files are then processed by the macros `MaterialBudget.C` and `MaterialBudget_TDR.C` which produce the plots stored in `Images`. They are:

- x/X_0 profile as a function of pseudorapidity η
- x/X_0 profile as a function of azimuthal angle φ
- x/X_0 profile as a function of pseudorapidity η and azimuthal angle φ
- $1/X_0$ profile as a function of z and polar radius R_{xy}
- Sum of x/X_0 (from the origin of CMS frame) profile as a function of z and polar radius R_{xy}
- x/λ_0 profile as a function of pseudorapidity η
- x/λ_0 profile as a function of azimuthal angle φ
- x/λ_0 profile as a function of pseudorapidity η and azimuthal angle φ
- $1/\lambda_0$ profile as a function of z and polar radius R_{xy}
- Sum of x/λ_0 (from the origin of CMS frame) profile as a function of z and polar radius R_{xy}

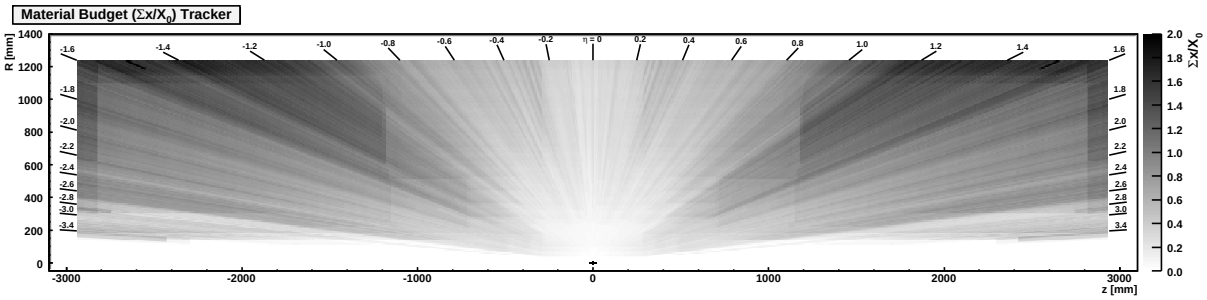
for each sub-structure, for the whole Tracker or the pixels only, the microstrip Tracker only, the inner Tracker (TIB, TID+, TID- and inner services). The profile of x/X_0 as a function of η is shown for the Tracker geometry published in `CMSSW_1_5_1` in Fig. 5 for the different material categories (a) and the different sub-detectors (b).

Fig. 6 shows the total amount of x/X_0 crossed by a particle coming from the CMS reference frame origin in the (R_{xy}, z) plane (a) and the local amount of material as a function of R_{xy} and z (b), from the geometry published in `CMSSW_1_6_0`.

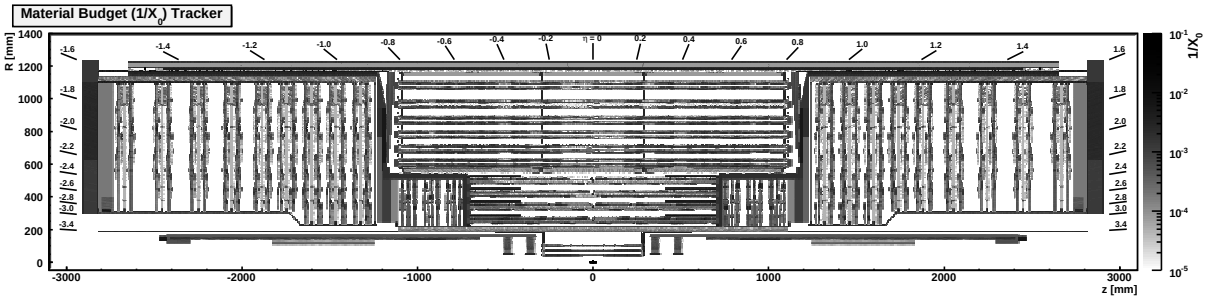


(a) (b)

Figure 5: Material Budget profile of the Tracker simulation from `CMSSW_1_5_1`: fraction of radiation length x/X_0 as a function of pseudorapidity η for the different material categories (a) or sub-detectors and structures (b).



(a)



(b)

Figure 6: Material Budget profile of the Tracker simulation from `CMSSW_1_6_0`: fraction of radiation length x/X_0 summed along the path of particles coming from the CMS reference frame origin in (R_{xy}, z) plane (a) and local amount of material $(1/\mathrm{X}_0)$ in the as a function of R_{xy} and z (b).

3.3 Volume properties

The properties of the GEANT4 volumes are accessible with the class `PrintMaterialBudgetInfo` published in the `CMSSW` package `SimG4Core/PrintGeomInfo`. The volume for which a loop on all the physical daughter volumes is set in the configuration file `run_MaterialBudgetInfo.cfg` stored in the `test/data` directory of the package. The line to be changed is the one setting the `Name` of the volume, in this example the TOBModule2, shown in Fig. 4:

```
replace g4SimHits.Watchers = {
       {   string type = ''PrintMaterialBudgetInfo''
           untracked string Name = ''TOBModule2''
       }
   }
```

Running

```
cmsRun $CMSSW_RELEASE_BASE/src/SimG4Core/PrintGeomInfo/
                              test/data/run_MaterialBudgetInfo.cfg
```

three text files are produced as output:

TOBModule2.weight A text file with the properties of all the GEANT4 volumes children of the chosen mother volume. For each physical volume the following information is reported:

- the geometric level in the GEANT4 hyerarchy;
- the name of the volume;
- the copy number of the volume;
- the name of the solid (the shape of the volume);
- the name of the material with which the solid is filled;
- the density of the material;
- the weight of the volume.

The output of the TOBModule2 volume shown in Fig. 4, which represents one of the TOB microstrip detector modules mounted on layers 3 and 4, is listed in Tab. 1.

TOBModule2_table.tex The LaTeX table obtained from the previous file (cut and pasted in this note into Tab. 1).

TOBModule2.element The list of chemical elements present in the mixture with the GEANT4 index, the total mass and the mass fraction. The total mass is computed excluding from the sum all the volumes filled only with air. The list of chemical elements for the TOBModule2 volume and his physical children is in Tab. 2.

Geom. Level	Volume Name	Copy Number	Solid Name	Material Name	Density	Mass
8	TOBModule2	6	TOBModule	Air	1.214 mg/cm^3	278.673 mg
9	TOBSiEncaps	1	TOBEncapsulant1	TOB_Sens_Interface	756.65 mg/cm^3	779.846 mg
9	TOBSiBackEncaps	1	TOBEncapsulant2	Silicone_Gel	965 mg/cm^3	912.89 mg
9	TOBPA1	1	TOBPA1	TOB_PA_rphi	2.33941 g/cm^3	2.76635 g
9	TOBPAEncaps	1	TOBEncapsulant1	TOB_Sens_Interface	756.65 mg/cm^3	779.846 mg
9	TOBSideRailL	1	TOBSideRailL	TOB_sid_rail1	1.75613 g/cm^3	5.01551 g
9	TOBSideRailR	1	TOBSideRailR	TOB_sid_rail2	1.511 g/cm^3	4.31542 g
9	TOBFrame	1	TOBFrame	TOB_frame_ele	1.536 g/cm^3	5.184 g
9	TOBHybSup	1	TOBHybSup	TOB_hybrid_supp	3.05582 g/cm^3	4.27387 g
9	TOBModCool1	2	TOBModCool1	TOB_mod_cool1	3.96724 g/cm^3	380.855 mg
9	TOBModCool1	1	TOBModCool1	TOB_mod_cool1	3.96724 g/cm^3	380.855 mg
9	TOBModCoolComp1	2	TOBModCoolComp1	TOB_mod_comp	1.37093 g/cm^3	171.092 mg
9	TOBModCoolComp1	1	TOBModCoolComp1	TOB_mod_comp	1.37093 g/cm^3	171.092 mg
9	TOBModCool2	2	TOBModCool2	TOB_mod_cool2	4.39847 g/cm^3	554.207 mg
9	TOBModCool2	1	TOBModCool2	TOB_mod_cool2	4.39847 g/cm^3	554.207 mg
9	TOBModCoolComp2	2	TOBModCoolComp2	Carbon fibre str.	1.69 g/cm^3	210.912 mg
9	TOBModCoolComp2	1	TOBModCoolComp2	Carbon fibre str.	1.69 g/cm^3	210.912 mg
9	TOBWaferRphi2	1	TOBWaferRphi	Silicon	2.33 g/cm^3	899.413 mg
10	TOBActiveRphi2	1	TOBActiveRphi	Silicon	2.33 g/cm^3	19.9786 g
11	TOBInactive	1	TOBInactive	Silicon	2.33 g/cm^3	339.038 mg
9	TOBHybrid2	1	TOBHybrid	TOB_ele34	2.60162 g/cm^3	1.95122 g

Table 1: TOBModule2 volume list.

Element	Index	Total Mass	Mass Fraction
Oxygen	1	5.75559 g	0.115504
Hydrogen	3	944.68 mg	0.018958
Carbon	4	13.3352 g	0.267613
Iron	5	278.501 mg	0.00558901
Manganese	6	1.20585 mg	2.41992e-05
Chromium	7	0.0166349 mg	3.33833e-07
Nickel	8	2.39159 mg	4.79949e-05
Aluminium	10	4.31244 g	0.0865428
Copper	13	1.89956 g	0.0381207
Silicon	15	22.818 g	0.457917
Sulfur	16	0.0502733 mg	1.00889e-06
Phosphor	17	0.0360293 mg	7.23043e-07
Lead	19	10.7974 mg	0.000216684
Tin	20	26.386 mg	0.00052952
Barium	21	53.1654 mg	0.00106693
Titanium	22	58.4253 mg	0.00117249
Silver	24	42.4833 mg	0.000852565
Bor 10	26	7.55764 mg	0.000151668
Bor 11	27	33.2536 mg	0.00066734
Antimony	29	7.64269 mg	0.000153375
Sodium	31	75.2784 mg	0.0015107
Potassium	32	92.5232 mg	0.00185677
Zinc	33	74.8409 mg	0.00150192
Total Weight	without Air	49.8301 g	Total Fraction 1

Table 2: TOBModule2 chemical elements list.

4 Tracker Geometry Validation

Any change on the positioning or orientation or detid of the silicon or pixel detectors makes the Tracker geometry defined in different CMSSW releases uncompatibles with each other. It means that data simulated with one CMSSW version cannot be reconstructed with the other version. This is summarised with the requirement of *backward compatibility* between different CMSSW releases which has to be satisfied if not clearly stated otherwise.
To ensure the *backward compatibility* and debug the Tracker geometry code, a *validation procedure* has been defined. It has to run after every change of the Tracker geometry code is published in the CMSSW software repository.
The core of the validation procedure is a set of three scripts which are collected in
`Validation/Geometry/test`:

`TrackerGeometryValidation.sh` the script to run the complete set of Tracker geometry validation tests and comparisons between two different versions of CMSSW;

`copyWWWTrackerGeometry.sh` the script to copy the most important validation output files and plots into the Tracker validation directories, available online from the link
http://cmsdoc.cern.ch/cms/performance/tracker/activities/validation/validation.html;

`copyReferenceFiles.sh` the script to copy all the output files in the Tracker directory for future reuse.

The Tracker validation geometry will run launching the command from a CMSSW environment

```
./TrackerGeometryValidation.sh CMSSW_1_4_0
```

to compare the actual version of the geometry with the one from another CMSSW version, in the example the CMSSW_1_4_0 version. In the next paragraphs a detailed description of the validation steps will be given.

4.1 Tracker geometry validation procedure

The `TrackerGeometryValidation.sh` script first prepares the working area and download the reference files chosen accordingly to the CMSSW version given as input. It creates a text file, called
`TrackerGeometryValidation.log` to collect the validation summary.
The first output lines contain the description of the job and the definition of the *new* and *old* geometry, coming from the diffrent CMSSW versions:

```
-----------------------------------------------------
  VALIDATION OF THE TRACKER GEOMETRY
    NEW = CMSSW_1_5_1 vs OLD = CMSSW_1_4_0
-----------------------------------------------------
```

The Tracker geometry validation can be run from a private CMSSW environment to debug private updates of the geometry files, in this case the *new* geometry is the one defined in the private `Geometry/TrackerCommonData` and `Geometry/CMSCommonData` packages which are, in the example, in the user's `CMSSW_1_5_1` environment.
The working area is set to the directory from where the script is launched and the directory from where the reference *old* geometry files are downloaded is defined:

```
Working area: /data/riccardo/CMSSW_1_5_1/src/Validation/Geometry/test
Reference area: /afs/cern.ch/cms/performance/tracker/activities
                     /validation/ReferenceFiles/CMSSW_1_4_0/Geometry
```

If the `Images` directory to store the plots has not been defined before, the script creates it:

```
Creating directory Images/
...done
```

The sample of simulated particles chosen to run the Tracker material budget analysis is set by the content of the `Validation/Geometry/data/download.url` introduced in Sec. 3.2, if no any other `single_neutrino.random.dat` file is found in the working directory:

```
Download the Monte Carlo source file...
...done
```

This sample contains 100 000 single neutrinos simulated homogeneously from the nominal CMS primary vertex $(0, 0, 0)$ with $|\eta| < 5$. For a smaller sample of 10 000 neutrinos simulated with the same characteristics, it is enough to copy the file
`/afs/cern.ch/cms/data/CMSSW/Validation/Geometry/data/single_neutrino.random.dat140`
in the working directory and rename it `single_neutrino.random.dat` before running the validation script.
The reference ROOT files stored into the reference directory (see above) are then copied into the local area

```
Download the reference 'old' files...
...done
```

to be used in comparing the actual version of the geometry and the chosen as a reference.
After this inizialization, the validation procedure starts, at first the material budget analysis is run for all the Tracker sub-structures and the beam pipe (see Sec. 3.2):

```
Running Tracker Structure...
...done
Running Pixel Barrel...
...done
Running Pixel Forward Plus...
...done
Running Pixel Forward Minus...
...done
Running TIB...
...done
Running TID+...
...done
Running TID-...
...done
Running Inner Tracker Services...
...done
Running TOB...
...done
Running TEC...
...done
Running Tracker...
...done
Running BeamPipe...
...done
```

The ROOT files thus produced are used to create the material budget profiles listed in Sec. 3.2 with the *new* version of the geometry:

```
Run the Tracker macro MaterialBudget.C to produce the 'new' plots...
...done
```

A new ROOT macro `TrackerMaterialBudgetComparison.C` is then run to compare the output of the material budget analysis from the *new* and *old* version of the geometry:

```
Run the Tracker macro TrackerMaterialBudgetComparison.C to compare 'old'
and 'new' plots...
...done
```

The output of this macro is a set of comparison images between material budget profiles:

- x/X_0 profile as a function of pseudorapidity η
- x/X_0 profile as a function of azimuthal angle φ
- x/X_0 profile as a function of pseudorapidity η and azimuthal angle φ
- $1/\mathrm{X}_0$ profile as a function of z and polar radius R_{xy}
- Sum of x/X_0 (from the origin of CMS frame) profile as a function of z and polar radius R_{xy}
- x/λ_0 profile as a function of pseudorapidity η
- x/λ_0 profile as a function of azimuthal angle φ
- x/λ_0 profile as a function of pseudorapidity η and azimuthal angle φ
- $1/\lambda_0$ profile as a function of z and polar radius R_{xy}
- Sum of x/λ_0 (from the origin of CMS frame) profile as a function of z and polar radius R_{xy}

The comparison is done also for the different material categories (see Sec. 3.2), superimposing the two profiles, or histogramming the ratio between the *new* and the *old* one.
The comparison between the Tracker x/X_0 profile as a function of η from the geometry published in `CMSSW_1_5_1` (*new*) and `CMSSW_1_4_0` (*old*) is shown in Fig. 7a. For each material category and the whole material budget the statistical compatibility factor between the two histograms (Kolmogorov Factor, KF) is printed with the integral of both distributions to have a rough estimate of the difference between them. In Fig. 7b the comparison is presented showing the ratio between the *new* and *old* histograms.

At the end of the material budget analysis, further checks are performed, first of all the file containing the information of the active volumes positioning and orientation, described in Sec. 2.1, is produced

```
Run the Tracker ModuleInfo analyzer to print Tracker Module
info (position/orientation)...
...done
```

and then compared with the reference one

```
Compare the ModuleInfo.log (Tracker Module position/orientation) file
with the reference one...
...done
```

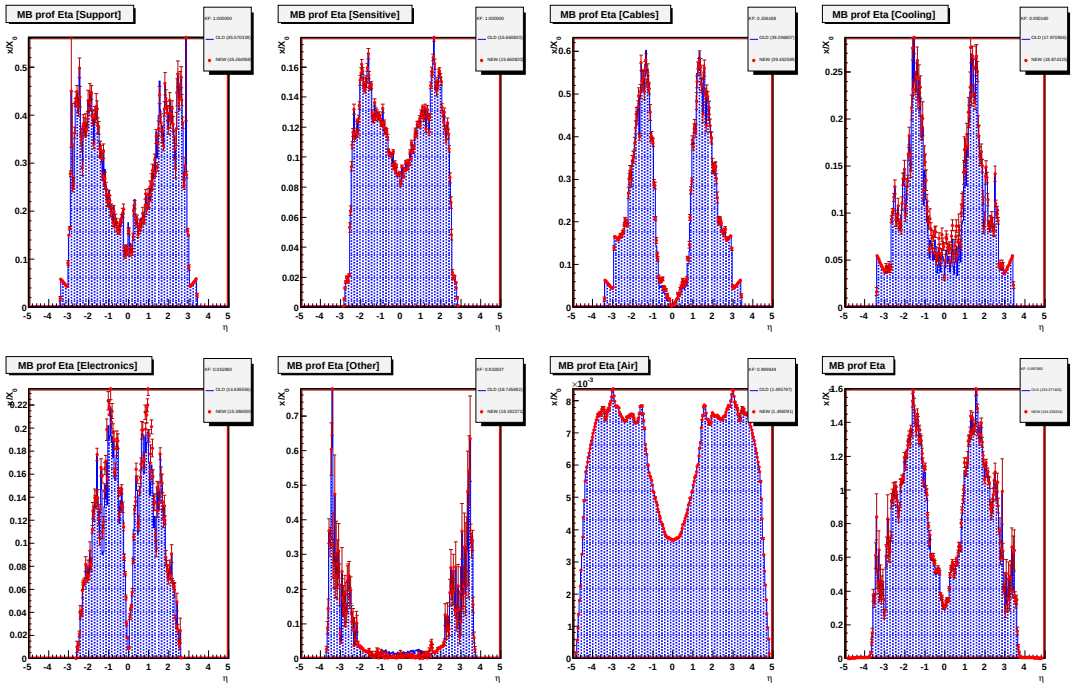
If there are differences between the two text files a warning is given:

```
WARNING: the module position/orientation is changed, check diff_info.temp
file for details
...done
```

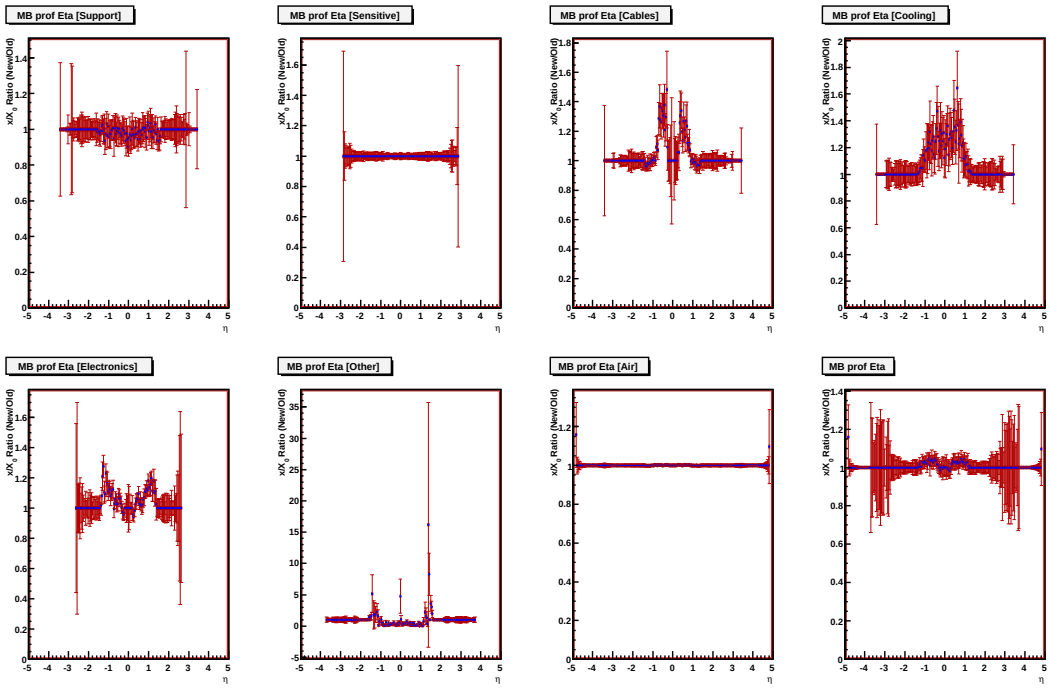
otherwise if everything is good the output is:

```
Tracker Module position/orientation OK
...done
```

The file `diff_info.temp` contains the output of the differences in the text of the two `ModuleInfo.log` files. Then the numbering scheme check, described in Sec. 2.2 is performed. The output of the numbering scheme test must appear as the following:



(a)



(b)

Figure 7: Comparison of the Material Budget profile of the Tracker simulation between `CMSSW_1_5_1` (*new*) and `CMSSW_1_4_0` (*old*): fraction of radiation length $\mathrm{x/X_0}$ (a) and ratio of the $\mathrm{x/X_0}$ between *new* and *old* version of the geometry (b) as a function of pseudorapidity η for the different material categories and the total material (bottom right plots).

```
Run the Tracker ModuleNumbering analyzer to print Tracker Numbering check...
-------------------------------------
Module Numbering Check Summary
-------------------------------------
Number of checks:   18636
             OK:    18636 (100.00%)
         ERRORS:        0 (0.00%)
-------------------------------------
...done
```

with 0% errors, because the detid definition must not change [9]. An output different from the one shown above must be justified by changes in the numbering scheme code done on purpose.
A second check is performed mixing together the numbering scheme, the GEANT4 geometry and the active volume positioning. The class `ModuleInfo.cc` produces also a text file `ModuleNumbering.dat` with the list, for each active area, of the detid, the `nav type`, the coordinates of the geometric center. This file is first compared with the reference one and if nothing has changed, the output appears as

```
Compare the ModuleNumbering.dat (Tracker Module position/orientation)
file with the reference one...
Tracker Module numbering OK
...done
```

otherwise

```
Compare the ModuleNumbering.dat (Tracker Module position/orientation)
file with the reference one...
WARNING: the module numbering is changed, check diff_num.temp file for
details
...done
```

and the differences between the actual and the reference file are stored in the `diff_num.temp` file.
The causes of differences in the `ModuleNumbering.dat` or `ModuleInfo.log` files can be several. If no errors are detected in the numbering scheme process with the previous test, hence we can assume that the detid's have not changed from the *old* to the *new* version of the geometry, the only elements that can change are:

coordinates of the active volume centers It can happen if delibaretely some active volumes have been moved to update their positioning (sizeable changes) or due to numerical approximations of the compiler (in this case small changes, less than 0.1 μm, hence negligible;)

`nav type` if the geometry is frozen this case cannot happen, otherwise if some updates have been done the `nav type` can change if some volumes have been placed somewhere in the hierarchy list within in between the GEANT4 path to reach the active volume. Taking as a reference the example shown in Sec. 2.1 if two new volumes are placed as children of the Tracker mother volume, before TIB volume is constructed, the `nav type` changes from $(0, 0, 0, 3, 3, 1, 42, 1, 3, 0, 0,)$ to $(0, 0, 0, 5, 3, 1, 42, 1, 3, 0, 0,)$. This is not an error, but only the result of Tracker geometry updates between two different releases.

To quantify the differences in the active volume coordinates and make a further check on the detid's, a ROOT macro `TrackerNumberingComparison.C` is run. It maps all the detid's and the coordinates of the active volume centers using the `nav type` as map index. This check is not meaningful if, due to Tracker geometry updates, the `nav type` of the active volumes changes. If `nav type`'s are not changed, two possible sources of errors can arise:

coordinates of the active volume centers It can happen only if the active volume coordinates have changed by more than the tolerance limit set to 0.1 μm, separately for x, y and z. This error should be ignored if the active volumes positioning has been deliberately modified in the *new* version of the Tracker geometry.

detid The active volume has a different numbering. Since detid's checks have been performed in the previous steps, it must not occur except if the numbering scheme logic has been modified.

When there are no errors, the output appears as

```
Run the TrackerNumberingComparison.C macro
Tracker Numbering Scheme OK
...done
```

otherwise an error message is printed

```
Run the TrackerNumberingComparison.C macro
ERROR: a failure in the numbering scheme, see NumberingInfo.log
...done
```

with a detailed report of the errors found stored in the file `NumberingInfo.log`.
The last test which is performed is running the GEANT4 overlap tool to search for volumes overlapping in the Tracker geometry. All the volumes are scanned to search for overlaps. The overlaps can occur when two volumes placed at the same hierarchy level (children of the same mother volume) share a portion of the space or when a child volume is not entirely contained within the mother volume. In the first case the `CMSSW` executables do not crash when loading the Tracker geometry, in the second case they do.
The output of the overlap check tool is stored in the `trackerOverlap.log` file. If no overlaps are detected no lines are written in the `TrackerGeometryValidation.log` file, otherwise the lines with some details of the detected overlap are written:

```
WARNING - G4PVPlacement::CheckOverlaps()
Overlap is detected for volume TOBRod0H
with volume TOBRod0L
at daughter local point (-55.1027,4.34835e-14,544.855)

--
```

All the Tracker overlaps have been cured, hence no further warnings must be detected by the GEANT4 overlap check tool.
This is the last check performed and the final line will then appear in the `TrackerGeometryValidation.log` file

```
TRACKER GEOMETRY VALIDATION ENDED...  LOOK AT THE RESULTS
```

4.2 Files publishing

All the images and the main results of the validation are stored in the directory `/afs/cern.ch/cms/performance/tracker/activities/validation/$RELEASE`, where `$RELEASE` is the corresponding `CMSSW` release or pre-release. To store them it is enough to run the shell script

```
./copyWWWTrackerGeometry.sh
```

which creates the Tracker validation directories, if not present, and store all the files. They are then accessible from the Tracker validation web page http://cmsdoc.cern.ch/cms/performance/tracker/activities/validation/validation.html.

4.3 Reference files

The files produced during the validation procedure are stored in the Tracker validation reference directory `/afs/cern.ch/cms/performance/tracker/activities/validation/ReferenceFiles/$RELEASE` for each `CMSSW` release or pre-release `$RELEASE`. The script to copy them is

```
./copyReferenceFiles.sh
```

which creates the reference directory, if it has not already been done, and copy all the ROOT files and text output files and the images to be used as a reference.

4.4 Simplified validation procedure

The procedure described in Sec. 4.1 checks all the Tracker sub-structures and is useful when each CMSSW release or prerelease is built. During the update of the xml files (review of all the materials and all the volumes) it is useful to privately check if the files ready to commit in the CMSSW repository are correct from the validation point-of-view. It means that a minimal set of tests must be performed before releasing the code. The full validation chain should be time consuming since tests are performed for each Tracker sub-structure and not for the few ones updated by the developer. A reduced set of tests must be although performed before releasing the code in CMSSW. They are:

GEANT4 overlap

check if the new/modified volumes have introduced overlaps in the Tracker geometry:

```
cmsRun $CMSSW_RELEASE_BASE/src/Validation/
                    CheckOverlap/test/data/runTracker.cfg
```

and look at the output of the algorithm searching for WARNINGs;

Active volumes

check if the active volumes have been moved or detid's changed

```
cmsRun $CMSSW_RELEASE_BASE/src/Geometry/
                TrackerGeometryBuilder/test/trackerModuleInfo.cfg
diff ./ModuleInfo.log /afs/cern.ch/cms/performance/tracker/activities/
             validation/ReferenceFiles/$Version/Geometry/ModuleInfo.log
```

which should return an empty output, at least in the part of the file regarding the updated subdetector, if no changes were done on purpose in the active volume positioning. If some volumes have been added or canceled, and consequently the nav type's are modified, they will appear as differences between the two files (see Sec. 4.1 for a more detailed explanation).

Only if both tests have been passed, the CMSSW developer can publish the updated files and should send an email to the geometry hypernews forum hn-cms-geometry@cern.ch with a brief description of the geometry updates.

References

[1] CMS Collaboration, *"CMS: The Tracker Project Technical Design Report"*, CERN/LHCC 98-06, CMS TDR 5, 15 April 1998,
CMS Collaboration, *"Addendum to the CMS Tracker TDR"*, CERN/LHCC 2000-016, CMS TDR 5 Addendum 1, 21 February 2000.

[2] CMS Collaboration, *"The Compact Muon Solenoid Technical Proposal"*, CERN/LHCC 94-38, LHCC/P1, 15 December 1994.

[3] The LHC Study Group, CERN/AC/95-05 (1995), *"The Large Hadron Collider Conceptual Design Report"*.

[4] Geant Collaboration, "GEANT4 - *a simulation toolkit"*, Nuclear Instruments and Methods in Physics Research A 506 (2003) 250-303;
Geant Collaboration, *"Geant4 developments and applications"*, IEEE Transactions on Nuclear Science 53 No. 1 (2006) 270-278.
The GEANT4 web page is http://geant4.web.cern.ch/geant4/.

[5] See http://cmsdoc.cern.ch/cms/software/ddd/www.

[6] The CMS software framework CMSSW web page is online at https://twiki.cern.ch/twiki/bin/view/CMS/CMSSW, the userguide is available at https://twiki.cern.ch/twiki/bin/view/CMS/WorkBook.

[7] M. Winkler, CMS-THESIS 2002-015, dissertation Technischen Universitat Wien, Fakultat fur Technische Naturwissenschaften und Informatik, Wien, Austria, *"A Comparative Study of Track Reconstruction Methods in the Context of CMS Physics"*.

[8] H. Breuker and P. Petagna, *"The CMS parameter booklet"*, CMS-TK-GN-0001 ver. 2, EDMS Id. 367936, available at https://edms.cern.ch/document/367936; P. Ingenito, CMS-DIGEP-ES-0001 ver. 3, EDMS Id. 90485, available at https://edms.cern.ch/document/90485.

[9] R. Ranieri, CMS-IN 2007/020, 8^{th} March 2007, *"The Silicon Microstrip Tracker Numbering Scheme"*.

[10] See http://cmsdoc.cern.ch/cms/Tracker/CernSW/geometry/compositemats/CompositeMats.html.

[11] R. Brun and F. Rademakers, *"ROOT - An Object Oriented Data Analysis Framework"*, Proceedings of the AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86.
See also http://root.cern.ch/;
ROOT User's Guide v5.16 is available at http://root.cern.ch/root/doc/RootDoc.html.