# Table of Contents

# Reconstruction of the Beam Spot

Complete:
Detailed Review status

## Contact

- Francisco Yumiceva

## Introduction

The beam spot is the luminous region produced by the collisions of proton beams. The position and spread of the beam spot needs to be measured precisely because of the following reasons:

- Important input for physics e.g. b-tagging, lifetime, etc.
- For patter recognition (input for HLT and offline reconstruction).
- Extraction of the tracking impact parameter resolution.
- Quick check of global alignment.
- Provide feedback to accelerator groups.
- Beam monitor.

The algorithm used to calculated the transverse beam position is the so called **d0-phi algorithm**. This is a robust and fast $^2$ fit which just needs 1000 good tracks to reach micron precision. Using a Log-Likelihood fit, the beam width can be extracted using 20000 good tracks. A more detail description can be found in the CMS Note 2007/021.

## Access to the beam spot data

The beam spot data is stored in RECO, AOD, and PAT datasets. This collection can be accessed from an analyzer in CMSSW as follows:

```
#include "DataFormats/BeamSpot/interface/BeamSpot.h"

// within the analyze()
reco::BeamSpot beamSpot;
edm::Handle<reco::BeamSpot> beamSpotHandle;
iEvent.getByLabel("offlineBeamSpot", beamSpotHandle);

if ( beamSpotHandle.isValid() )
{
    beamSpot = *beamSpotHandle;

} else
{
    edm::LogInfo("MyAnalyzer")
      << "No beam spot available from EventSetup \n";
}

double x0 = beamSpot.x0();
double y0 = beamSpot.y0();
double z0 = beamSpot.z0();
double sigmaz = beamSpot.sigmaZ();
double dxdz = beamSpot.dxdz();
double BeamWidthX = beamSpot.BeamWidthX();
double BeamWidthY = beamSpot.BeamWidthY();

// print the beam spot object
```

```
// cout << beamSpot << endl;

// look at BeamSpot.h in DataFormats package
// for a complete list of member functions for
// the beam spot object
```

The full list of member functions available in the beam spot object can be found in the reference manual.

# Using the beam spot to correct the track impact parameter

The default impact parameter parameter of tracks are give with respect to the center of the Tracker (0,0,0). The impact parameter should be correct by the beam line offset before is used for analysis. The correction can be done either with respect to the primary vertex or with respect to the beam spot. The former can be done by using the tracking tools in the package IPTools. The later is done as follows:

```
// please see above for definition of beamSpot object
// reco::BeamSpot beamSpot;
// reco::TrackRef Track;

math::XYZPoint point(beamSpot.x0(),beamSpot.y0(), beamSpot.z0());
double d0 = -1.* Track->dxy(point);
// circular transverse beam width in MC. In data widthX ~ widthY, let's wait an see
double d0sigma = sqrt( Track->d0Error() * Track->d0Error() + 0.5* beamSpot.BeamWidthX()*beamSpot.
```

# Status of the Beam spot in MC samples

## Simulation of the beam spot

The generated particles have the production vertex at the origin (0,0,0) of the coordinate system. Therefore, to simulate a realistic luminous region the vertices of the generated particles are smeared by a Gaussian distribution in Z, and by the accelerator beta function for the transverse plane. In the case of the presence of beams with a crossing angle, the momentum of the particles are also corrected by a Lorentz boost.

The vertex smearing (and Lorentz boost) should be included in the simulation sequence by calling one of the vertex smearing modules available in "Configuration/StandardSequences", for example:

```
process.load("Configuration.StandardSequences.VtxSmearedEarly10TeVCollision_cff")
```

The following configuration files are available for several beam conditions:

- VtxSmearedEarly900GeVCollision_cff.py
- VtxSmearedEarly10TeVCollision_cff.py
- VtxSmearedBetafuncEarlyCollision_cff.py
- VtxSmearedBetafuncNominalCollision_cff.py

The values of the parameters for each beam condition can be found in the file "IOMC/EventVertexGenerators/python/VtxSmearedParameters_cfi.py". In addition to the vertex smearing by the beta function, there are options to smear the vertices by a flat or Gaussian distributions, see the "IOMC/EventVertexGenerators" package for more details.

## Reconstruction of the beam spot

In order to provide a beam spot data in MC samples, we first generate and reconstruct a QCD sample with a given beam conditions. Then, the beam spot analysis code is run to fit the data and extract the beam spot position, spread, and errors. Finally, the beam spot data is loaded into the conditions database so when the

reconstruction is run, the beam spot data is copied from the database to the event and it is stored in RECO as an EDM collection.

# (need to fix this) Get beam position from the event

Since **170pre4**, the beam spot data is available in the event. Skip to step 3 to learn how to access the beam spot data in this case.

In releases **< 170pre4** for example in **168**, you can get the beam spot from the event after the following steps:

1. Get recent version of beam spot producer:

```
addpkg RecoVertex/BeamSpotProducer
```

2. Enable beam spot Producer. In your analysis configuration cfg file by adding the following line

```
include "RecoVertex/BeamSpotProducer/data/BeamSpotSimpleGaussian.cff  // for official samples
// or use one of the following for the test samples (see above):
// BeamSpotEarlyCollision.cff, BeamSpotNominalCollision.cff, BeamSpotNominalCollision1.cff,
// BeamSpotNominalCollision2.cff, BeamSpotNominalCollision3.cff, BeamSpotNominalCollision4.cff
```

If you have your own beam spot in a sqlite file or in the DB, you can modify the cff file above to read your beam spot with the respective tag name.

3. In your analyzer include the beam spot data format and get the data:

```
#include "DataFormats/BeamSpot/interface/BeamSpot.h"

reco::BeamSpot vertexBeamSpot;
edm::Handle<reco::BeamSpot> recoBeamSpotHandle;
iEvent.getByType(recoBeamSpotHandle);
vertexBeamSpot = *recoBeamSpotHandle;
```

4. Access the reco::BeamSpot member functions, see the reference manual

5. An example to read the Beam Spot data from the DB and save it as EDCollection in a root file is given in the following cfg file:

```
RecoVertex/BeamSpotProducer/readFromProducer.cfg
```

# Get beam position from the database

Packages needed for releases 1.3.X and 1.4.X:

```
cvs co -r V00-00-00 CondCore/BeamSpotPlugins
cvs co -r V00-00-01 CondFormats/BeamSpotObjects
cvs co -r V02-01-01 CondFormats/DataRecord
cvs co -r V00-00-05 RecoVertex/BeamSpotProducer
cvs co -r V00-00-03 DataFormats/BeamSpot
```

Packages needed for releases 1.5.0pre2:

```
cvs co -r V00-00-05 RecoVertex/BeamSpotProducer
cvs co -r V00-00-04 DataFormats/BeamSpot
```

Currently, the beam spot information in the DB can be accessed through a sqlite file. The files needed to run this example can be downloaded here: BeamSpot.db and conddbcatalog.xml

Add the following includes and lines in your code to access the data:

```
#include "FWCore/Framework/interface/ESHandle.h"
#include "FWCore/Framework/interface/EventSetup.h"
#include "FWCore/Framework/interface/IOVSyncValue.h"
#include "CondFormats/DataRecord/interface/BeamSpotObjectsRcd.h"
#include "CondFormats/BeamSpotObjects/interface/BeamSpotObjects.h"

void
BeamSpotFromDB::analyze(const edm::Event& iEvent, const edm::EventSetup& iSetup)
{
        edm::ESHandle< BeamSpotObjects > beamhandle;
        iSetup.get<BeamSpotObjectsRcd>().get(beamhandle);
        const BeamSpotObjects *mybeamspot = beamhandle.product();

        std::cout << *mybeamspot << std::endl;
}
```

Look at the BeamSpotObjects class for more information about how to get all the parameters.

In your configuration file, add the following lines:

```
include "CondCore/DBCommon/data/CondDBCommon.cfi"
  replace CondDBCommon.connect = "sqlite_file:BeamSpot.db"
  replace CondDBCommon.catalog = "file:conddbcatalog.xml"
  es_source = PoolDBESSource {
    using CondDBCommon
    VPSet toGet = {
          { string record = "BeamSpotObjectsRcd" string tag = "SimpleGaussian" }
    }
  }
```

# Determination of beam position using an EDAnalyzer

In this section, we describe how to calculate the beam spot parameters using a set of good tracks. To reach a precision in the order of microns, it is necessary to loop over several events in a sample to collect at least 1000 good tracks.

1. Install CMSSW release 1_6_8. Please see  Setting up your Computing Environment

2. Checkout packages:

```
addpkg RecoVertex/BeamSpotProducer
```

4. Run over some events. In this example, we run over CSA07 QCD events:

```
cd RecoVertex/BeamSpotProducer/test
edit analysis_d0_phi.cfg:
[-> in PoolSource add a dataset, for example ->] '/store/mc/2007/7/11/CSA07-QCD_Pt_80_120-2027/00
[-> comment out the input file ->] #include "RecoVertex/BeamSpotProducer/test/NominalCollision4.c
now run cmsRun:
cmsRun analyze_d0_phi.cfg
```

The relevant output is the following:

```
calculating beam spot...
```

```
 we will use 7390 good tracks out of 28994
 number of tracks used: 0
 number of tracks used: 7390
 number of tracks used: 7390
 number of tracks used: 7390
 number of tracks used: 7390
 number of tracks used: 7390
 number of tracks used: 7390
 number of tracks used: 7384
 number of tracks used: 7373
 number of tracks used: 7359
 number of tracks used: 7337
 number of tracks used: 7301
 number of tracks used: 7244
 number of tracks used: 7189
 number of tracks used: 7129
 number of tracks used: 7013
 number of tracks used: 6787
 number of tracks used: 6382
 number of tracks used: 5665
 number of tracks used: 4615
 total number of iterations = 19
BSFitter: default fit does not extract beam width, assigning a width of zero.
 DEFAULT:
------------------------------------------------------
            Calculated Beam Spot

   X0 = 0.000137297 +/- 8.54651e-05 [cm]
   Y0 = 1.45161e-05 +/- 8.72648e-05 [cm]
   Z0 = 0.178271 +/- 0.762762 [cm]
 Sigma Z0 = 4.69784 +/- 0.577968 [cm]
 dxdz = 3.60997e-06 +/- 1.61885e-05 [radians]
 dydz = -5.41738e-05 +/- 1.6445e-05 [radians]
 Beam Width = 0 +/- 0 [cm]
------------------------------------------------------
```

This sample was generated at (0,0,0) with no crossing-angles, a RMS bunch length in Z of 5.3 cm and the average transverse beam width is of 15 microns. The beam width can be extracted using the log likelihood fit implemented in this package which requires more tracks and a tighten track quality criteria.

5. Track quality cuts can be modified via the framework, see the configuration file

```
data/d0_phi_analyzer.cff
```

# Use different beam conditions during reconstruction

Below is a recipe to change the beam conditions before reconstruction:

1. Create a text file with a new beam position. This can be done by hand or running the Beam Spot analyzer over a private sample. For example, the beam spot analyzer create a text file with the results of the fit:

```
[cmslpc06] test > more EarlyCollision.txt
X 0.0323201
Y -0.000137265
Z 0.110321
sigmaZ 3.83198
dxdz 5.58847e-06
dydz -1.97724e-05
BeamWidthX 45.9e-4
BeamWidthY 45.9e-4
Cov(0,j) 7.68449e-10 1.61892e-11 0 0 0 0 0
Cov(1,j) 1.61892e-11 7.75067e-10 0 0 0 0 0
```

```
Cov(2,j) 0 0 0.137499 0 0 0 0
Cov(3,j) 0 0 0 0.0972715 0 0 0
Cov(4,j) 0 0 0 0 5.43189e-11 1.16835e-12 0
Cov(5,j) 0 0 0 0 1.16835e-12 5.41043e-11 0
Cov(6,j) 0 0 0 0 0 0 4e-08
emittanceX 7.03e-08
emittanceY 7.03e-08
betaStar 300.0
```

You can modify the file by hand if you just want to change some of the values.

2. Create a sqlite file to be loaded during reconstruction:

```
addpkg RecoVertex/BeamSpotProducer
```

In this package, edit the file called "BeamSpotProducer/test/write2DB.py" to create a sqlite file. The new conditions that you will produce need to have a tag name. Change the following line to the name of the new condition, for example:

```
tag = cms.string('Early10TeVCollision_3p8cm_v4_mc_IDEAL')
```

The tag name is used in step 3. The filename of the output file is given by this line:

```
process.CondDBCommon.connect = "sqlite_file:EarlyCollision.db"
```

Now run cmsRun to create the sqlite file "EarlyCollision.db"

```
cmsRun write2DB.py
```

"cmsRun readDB.py" can be used to check that "EarlyCollision.db" contains the desired parameters.

3. Load new conditions. In your reconstruction configuration file, you need to overwrite the global tag with the following lines:

```
process.BeamSpotDBSource = cms.ESSource("PoolDBESSource",
       process.CondDBSetup,
       toGet = cms.VPSet(cms.PSet(
       record = cms.string('BeamSpotObjectsRcd'),
       tag = cms.string('Early10TeVCollision_OVERWRITE'))),
       connect = cms.string('sqlite_file:EarlyCollision.db')
)
process.es_prefer_beamspot = cms.ESPrefer("PoolDBESSource","BeamSpotDBSource")
```

Notice that you need to copy "EarlyCollision.db" to the same directory where the configuration file lives.

4. Validation. Check that the new beam spot data is available in the beam spot collection "offlineBeamSpot". This step confirms that the default beam parameters have been overwritten. Collection name for example: "Events/recoBeamSpot_offlineBeamSpot_RECO/recoBeamSpot_offlineBeamSpot_RECO.obj"

# Online Beam Spot

This section describes how to use Online DQM for Beam Spot

1) Check out the codes:

```
cmsrel CMSSW_3_2_5
cd CMSSW_3_2_5/src
cmsenv
```

```
cmscvsroot CMSSW
cvs co -r V00-00-02      DQM/BeamMonitor
cvs co -r V00-02-30      RecoVertex/BeamSpotProducer
scramv1 b
cd DQM/BeamMonitor/test
```

2) Start DQM Gui Server:

- To run with DQM gui. You can install DQMGui server on a local machine first by following the instruction under the section of "Specific details" in this page: DQMTest
- Follow the instruction under the section of "Starting the GUI server"
- Remember to stop GUI server and collector when you are done with the test. See "Stopping the GUI server and the collector"

3) Edit DQM_RECO_cfg.py (or DQM_RAW_cfg), change <DQM Gui Server Host> to the machine that runs gui service

```
process.DQM.collectorHost = "<DQM Gui Server Host>"
process.DQM.collectorPort = 9190
```

4) Run:

```
cmsRun DQM_RECO_cfg.py
```

5) Open web browser to check histograms:

http://<DQM Gui Server Host>:8888/dqm/devtest

Note: Example shown here runs on local files (RECO). For testing with storage manager within .CMS network at P5 please use this cfg beam_dqm_sourceclient-file_cfg.py and modify the block of process.source to to files (.dat) you want to run.

# Upload new beam spot conditions

Show Hide
We are using the drop box procedure to upload beam positions to condDB. Two files need to be put in the drop box: one text template with details of the tag, and another one is a sqlite file with the payload. There is a cronjob that checks if new sqlite files are in the dropbox every 30 minutes.

Steps to upload a new payload: (1) Prepare the sqlite file and template file. Setup analyze_d0_phi_cfg.py to write the results of the fit in a text file BeamFit.txt. Run cmsRun with this python file. Edit BeamFit.txt to include correctly the beam widths, beam width error, betastar, emittances. Setup write2DB.py to read BeamFit.txt and include the new tag name. Run cmsRun write2DB.py to get a sqlite file. Check the sqlite.db file with read2DB.py, for this you also need to setup read2DB with the appropriate file name and tag. Now you have the sqlite file to be placed in the drop box. (2) Prepare payload: login to lxplus, then to cmsusrX. You need to setup the template file. Look at this directory for examples: ~yumiceva/export. Once you have the template file ready then run the shell script renameFiles.sh to format the sqlite and template files for the drop box. (3) Move formatted sqlite and template file to the beam spot drop box: /nfshome0/popcondev/BeamSpotJob/

To check if the payloads have been uploaded to DB, you can run the following:

from online: cmscond_list_iov -c oracle://cms_orcon_prod/CMS_COND_31X_BEAMSPOT -P /nfshome0/popcondev/conddb -t from offline: cmscond_list_iov -c oracle://cms_orcoff_prod/CMS_COND_31X_BEAMSPOT -P /afs/cern.ch/cms/DB/conddb -t

# Action items

- Beam spot monitoring: * Implement a fast DQM monitoring after HTL which run the full tracking reconstruction by lumi blocks and calculate the beam position. * Implement a DQM monitoring in the in AlCaReco skims. * Displays for the CMS control room of the beam position by the Tracker and by the LHC BPMs.
- Online beam spot: * Can we run the d0-phi method in the HLT? * Can we use the primary vertex finding in the HLT to estimate the beam position?
- Feedback the beam spot position to the LHC.
- Offline beam spot:
    - We are assuming that the beam width is the same in X and Y, we could change the code to take into account the possibility of different beam widths in X and Y.
    - Compare d0-phi method against the primary vertex finding method.

# Review status

| Reviewer/Editor and Date (copy from screen) | Comments |
|---|---|
| KatiLassilaPerini - 13 Jan 2009 | moved from wb to swguide |
| FranciscoYumiceva - 10 Feb 2007 | first version of page |

Responsible: FranciscoYumiceva
Last reviewed by: FranciscoYumiceva - 1 Feb 2008

---

This topic: CMSPublic > SWGuideFindingBeamSpot
Topic revision: r47 - 2011-10-10 - AndreHolzner