Inner Detector Alignment within the ATLAS Full Dress Rehearsal


Abstract

With the Full Dress Rehearsal (FDR) the ATLAS collaboration exercised the software of the complete ATLAS data taking chain, starting from the trigger sub farm output (SFO) to the physics analysis of the taken data. The data were simulated with an imperfect detector and the production of calibration constants was an essential part of the software chain. The capability to produce alignment constants for the reconstruction of the events within 24h after data taking was performed and the constants applied to the event reconstruction.
Figure 1: Integration of the ID alignment into the overall ATLAS data taking schema. This note discussed the part indicated as alignment chain.

1 Introduction

During the year 2008 several tests were performed to exercise the overall software chain to be used for the ATLAS detector [1]. The ATLAS computing model requires the readiness of calibration and alignment constants for all sub-detectors within 24h after data taking. Fig. 1 shows the integration of the Inner Detector (ID) alignment into the overall ATLAS alignment and calibration scheme during data taking. The alignment is performed using a dedicated ID calibration stream and was performed in a sequence taking the ID substructures into account. The beam spot was determined before the alignment and used as an input to the alignment procedure. After the production of new alignment constants a beam spot was calculated using express stream events and the updated alignment. The alignment constants as well as the beam spot were carefully monitored before and after the alignment. Based on the monitoring results a decision was taken whether new constants need to be uploaded to the database.

2 Data Sample and ID Calibration Stream

During the different FDR challenges (FDR1, FDR2 and its follow-ups) the ID Calibration Stream was used. The stream was not produced by the HLT trigger algorithm, i.e the Partial Event Builder (PEB) algorithm was not used to produce these samples and separate procedures were applied to produce single track samples similar to the expected calibration stream. The sample used for the FDR1 was produced by stripping of single tracks from existing simulated samples, while the sample for the FDR2 was produced
with dedicated simulated single tracks.

2.1 FDR1 Data sample

For the FDR1, the calibration stream used was based on the JF17 sample (dataset 5802) which was a dijets sample with $E_T$(jet)>17 GeV. Events which passed the trigger $trk10i_{calib}$ at Level 2 were written out. Using the IDSCAN track trigger algorithm isolated high momentum ($p_T > 10$ GeV/c) were selected. In order to increase the selection rate, the selection track $p_T$ cut was lowered from 10 to 5 GeV/c in the $trk10i_{calib}$ trigger, which approximately doubled the rate of useful tracks from the JF17 sample. The sample was biased towards a harder $p_T$ spectrum but using a lower $p_T$ cut the composition of low and high momentum tracks was roughly comparable. The bandwidth was constant while producing the calibration stream and it was possible to write just partial events (the rate was 60 Hz for tracks with hits in overlapping modules for a luminosity of $10^{31}$). Single tracks were written out to the ID calibration in a byte stream format, where the event was only partially stored.

The cosmic tracks used were converted from raw data objects (RDO) to bytestream format without partial event building.

2.2 FDR2 Data sample

The FDR2 calibration stream was composed of a single pion sample and a cosmic sample without using any trigger information. The cosmic events were simulated using the ATLAS-Comm-01-00-00 geometry, digitized with ATLAS-CSC-02-01-00 geometry (Magnetic field bmagatlas04_test1.data with initial displacements, which is identical to the misaligned ATLAS-CSC-02-00-00 geometry) and reconstructed using the conditions tag OFLCOND-CSC-00-01-00, meaning that the same realistic CSC misaligned geometry was used for simulation and reconstruction.

The single pion sample was produced with the misaligned geometry tag ATLAS-CSC-02-01-00 without displaced beam spot. The pions were generated uniformly with a momentum range from 10 to 50 GeV/c (1M events). For the generation of these pions the csc_evgen_trf.py jobOptions was used followed by csc_simul_trf.py. Only ID and no calorimeter information was used, which allows to do the simulation and digitization in one step. The final RDO file can be used directly as an input to the Reconstruction. Finally, in order to make them FDR-2 compliant, both samples were transformed to bytestream format using the InDetCalibStream package. The single pion and cosmic calibration stream sample are located at:

- /castor/cern.ch/grid/atlas/cal/atlcal/perm/id/FDR2_IDCalibStream/single_pion/
- /castor/cern.ch/grid/atlas/cal/atlcal/perm/id/FDR2_IDCalibStream/cosmics/

The size is about 10G and 100G respectively.

3 Overall Outline of the Procedure

The alignment procedure was performed almost automatically using dedicated python scripts. The full chain alignment started with a first pass beamspot determination using the calibration stream. Using the obtained beamspot position, the silicon alignment was calculated and the center of gravity of the system was calculated and subtracted from the constants. In a second step the TRT alignment was performed, using the Si constants produced in the previous step. At the end, the center of gravity was calculated again but using both systems, the Si and the TRT. Finally, with the new alignment constants, the beamspot was determined, but using the express stream as an input, which al-
allows to determine the uncertainty on the beamspot position. The final set of constants contained the final alignment constants plus the beamspot position calculated. The full chain is depicted in Figure 2. All the processes were launched from a python script that called each sub-script sequentially until the constants were obtained. The final alignment and beamspot constants were copied to a common place. The script was called InDetAlignExample_SuperScript.py and was located in the package InnerDetector_InDetExample_InDetAlignExample. The alignment was been done using a dedicated afs account (atlidali) and the jobs were send to the CERN analysis facilities (CAF) availables for the alignment, named atlascaf. At the time of the FDR 10 host with 8 cores each were available for the alignment. Each host was composed by one Intel Xeon CPU E5345 at 2.33 GHz with 8 cores and 16054 Mb of ram memory, meaning 2 GB of memory by core. The jobs were submitted using the atlas CERN batch service.

3.1 Si Alignment

The silicon alignment procedure consisted of several iterations with each iteration subdivided in two main steps: In the first step the tracks were processed and the alignment matrix filled. This step was done in parallel using the pion and cosmic data sample. In the FDR these jobs were sent to the atlas computing analysis facility (CAF). After the production of all output matrices, these matrices were combined in the second step. This combined matrix was used to solve the linear equation and the alignment constants of that iteration were obtained. This process was repeated until convergence was reached.

During each iteration the number of events to be reconstruct could be selected as well the alignment geometry level. The alignment geometry levels were divided in three levels:

- Level 1: The alignable structures are the SCT barrel, the two SCT endcaps and the Pixel detector. These structures were treated as a whole. In this level one has 4 structures leading to 24 degrees of freedom.

- Level 2: The structures to align are the layers and the endcap disks of the silicon detectors. Thus one has 4 SCT layers and 2 x 9 SCT endcap disks, and 3 pixel layers and 2 x 3 Pixel endcap disks. In total there are 31 structures with 186 degrees of freedom.
Level 3: The alignment is performed using each module as a separate alignable structure. There are 4088 SCT modules and 1744 Pixel modules leading to almost 35k degrees of freedom. In order to solve the linear system at level 3, the number of degrees of freedom needed to be reduced or only a subset of the detectors, as the barrel, was selected. Another option was to remove single degrees of freedom of all the modules. Since the FDR more alignment geometry levels have been implemented to the alignment software 1).

The alignment was performed using the python script named InDetAlignExample_IteratorForSilicon.py. The script created the directory structure of the output: the output was stored in the path defined by the python script. Inside this directory each iteration result was stored in a directory called IterN, where N reflected the iteration number. In this directory the matrices, constants and results of the iteration N were stored. Inside each iteration directory there were two subdirectories for collision data and cosmic data and in each of these sub-directories there were M subdirectories numbered from 0 to M with M reflecting the number of CPUs to be used during the iteration. In an other subdirectory, called logs, all the log files, jobOption files and scripts were saved. The python script created one jobOption for each subjob and sent it to the queues. When all the subjobs were finalized, the script called a C program (add_big.C) that added all submatrices and subvectors to obtain the final bigmatrix and bigvector. Finally, another jobOptions was created to solve the system composed by the bigmatrix and the bigvector. The output of this job obtained the alignment constants of that iteration. At the end of one iteration, the script launched the next iteration. The silicon iterator had a couple of flags, which could be changed by the user. The most important were:

- **OutputPath**: The path where the output will be stored.
- **Athena paths**: The paths of the release to be used.
- **ColEvents and CosEvents**: Number of events to be used. Zero means that this type of data will not be used and a -1 means that all the data will be used.
- **ColCPUs and CosCPUs**: number of CPUs that one want to use to run with colision and cosmic data.
- **FirstIteration**: First iteration to be run. Usually it starts with the iteration 0.
- **Iterations**: Number of total iterations to be performed.
- **RunCog**: This flag switched on or off the use of the centre of gravity calculation.

### 3.2 TRT Alignment

The TRT Alignment is run after the internal alignment of the silicon detectors and proceeds in two main stages. First the TRT Barrel [2] and two Endcaps [2] are aligned with respect to the silicon detectors, referred to as L1 alignment. The determination of the alignment parameters is an iterative process in which the tracks used for the alignment are processed several times in order for the alignment parameters to reach convergence. [2] To minimize the time spent during each iteration, the input events are partitioned, reconstructed in parallel, and combined in the end to determine the alignment parameters for the given iteration. Throughout the TRT alignment in the FDR, eighty sub-jobs were processed in parallel for each iteration. During the L1 alignment, combined silicon and TRT tracks were used for the alignment and the convergence of the alignment parameters was complete after a few iterations. Ten

---

1) https://twiki.cern.ch/twiki/bin/view/Atlas/InDetAlignGeometryLevel
iterations of the TRT alignment were done at L1 during the FDR. In the second step of the TRT alignment, known as L2 alignment, the internal TRT modules are aligned with respect to each other. In the FDR ten iterations of the TRT alignment were done at L2, using both combined and TRT-only tracks. The convergence of the alignment parameters was reached after a few iterations. To assess the quality of the TRT alignment the Athena package, InDetAlignmentMonitoring [2], which produces numerous monitoring histograms, is used to compare distributions before and after the TRT alignment. During the FDR, the alignment monitoring package was used to validate the TRT alignment and saw similar performance as was reported in ref [3].

RunTRTAlignment.py is the python script that controls the TRT alignment process and is called in the script, InDetAlignExample_SuperScript.py, which steers the entire Inner Detector(ID) alignment. RunTRTAlignment.py is located in the Athena package InDetAlignExample, and relies on the python helper class TRTAlignmentJob located in InDetAlignExample_TRTAlignmentToolbox.py. Parameters specific to the TRT alignment are set in RunTRTAlignment.py and include: number of iterations, number of events, number of sub-jobs, location of the input data, Athena setup commands, workarea, jobOptions location, input alignment/calibration constants, and reconstruction options. The number of iterations, events, and sub-jobs, set the number of the respective quantities to be run in the L1 and L2 alignment. The Athena setup commands specify the software release that is used in the reconstruction. The workarea determines the location of the output of the TRT alignment. Under this directory the directories IDAlignmentWorking and IDAlignmentOutput, referred to as the work directory and output directory respectively, are created. The jobOption location provides the path to the directory which contains the job options that configure the TRT alignment and ID alignment monitoring packages. The locations of the input alignment and calibration constants are provided by the InDetAlignExample_SuperScript.py script and point to the silicon alignment constants that have been reached earlier in the ID alignment process and, to the TRT drift time calibration [2]. Finally, the reconstruction options configure settings in the ID reconstruction job options located in the Athena package InDetRecExample. After the settings are defined, RunTRTAlignment.py ensures the various job options can be located and moves them to the workarea. The handling of the alignment settings and the control of the reconstruction jobs is done through the python class TRTAlignmentJob. RunTRTAlignment.py creates an instance of TRTAlignmentJob, which is initialized by the settings defined earlier. The rest of the TRT Alignment procedure is handled through calls to member functions of this class.

The first task of the TRTAlignmentJob, is the creation of the input files. The member function getFirstList() (or getFirstListFromCastor()) is called and returns a list of the input files with their corresponding directories. The function getFirstListToNumEventMapping() takes this list as an input and returns a mapping of the input file names to the number of events in the file. The function createInputFiles() then creates a file, inputFiles_{X}.py, for each sub-job, (X indicates the sub job number) located in the workarea. This file contains the input file name, the number of events to skip, and the number of events to process. The number of events per sub-job is set in RunTRTAlignment.py and the number of events to skip is determined by tracking how often the given input file was already used in a sub-job. All of this functionality is handled internally in the TRTAlignmentJob class.

After the inputFiles_{X}.py files have been created, RunTRTAlignment.py loops over the number of iterations, calling the member function runIteration() each time. runIteration(), in turn, loops over the number of sub-jobs and calls the functions createBatchScript() and submitParallelBatchJobs(). createBatchScript() creates a script TRT_AlignmentJob_{X}_{Y}.sh that contains the commands that are input to the batch system, here X is the iteration number and Y, the

---

2) A job option not being located causes the alignment procedure to abort with an error message.
The first is a command to create a directory of the form L2iter_{X,Y}. This directory is located in the work directory and is where the reconstruction will be run. Commands that copy the input alignment constants and job options to the directory are then written, as well the call to run Athena. Finally commands that copy the alignment output files to the output directory and remove the working directory are included. As well as creating the batch script, createBatchScript() calls the function createJobOptions() which creates the InDetRecExample job options in the working directory. The createJobOptions() function contains the skeleton of InDetRecExample job options internally and simply modifies the settings based on values specified in RunTRTAlignment.py. After createBatchScript() finishes, submitParallelBatchJobs() is called and the batch script is submitted to the CAF. Special care is taken to ensure the batch scripts of a given iteration are not submitted before the previous iteration finishes. Once the jobs are submitted, the batch scripts of a given iteration are executed in turn and the events are processed in parallel.

After all the sub-jobs of a given iteration finish the batch job TRT_{combo X}.sh is executed. This script is created and submitted to the batch similarly to the TRT_{Alignmentjob X Y}.sh files during the execution of the runIteration() function. The TRT alignment algorithm TRTAlignAlgs supports the combining of sub-jobs via adding the corresponding $\chi^2$ matrices to be minimized, as described in [2]. The TRT_{combo X}.sh scripts create the directories L2iter X, copy the relevant input files from the sub-jobs, and run the Athena job that determines the alignment parameters. At the end of the iteration the directory L2iter X contains the TRT alignment output corresponding to all the sub-jobs in the iteration.

After all the TRT Alignment iterations have been run and the alignment parameters determined, several TRTAlignmentJob functions designed to assess the quality of the alignment are executed. First, checkInputfile() ensures that the correct alignment is input to each iteration. It checks that output of one iteration was correctly copied for the input to the next, and flags any potential problems. Next the alignment monitoring files are merged. Each sub-job creates its own monitoring.root file which then have to be combined. The function mergeMonitoringFiles() does the merging and creates the file TotalMonitoring.root with the output in the corresponding L2iter X directory. Once the combined monitoring files have been created the script CompareIDAlignmentMonitoring.py, located in the TRT_AlignAlgs Athena package, is run. The purpose of this script is to plot the alignment monitoring distributions sensitive to a TRT misalignment from the monitoring file before the alignment together with those from after the alignment. The output is a single root file, AlignmentOutput.root from which one can directly compare the various distributions before and after the alignment. The final function to be executed is analyzeAlignment(). This runs a script AnalyzeTRTAlignment.py, again located in the TRT_AlignAlgs package, designed to monitor the alignment parameter convergence and quality as a function of iteration. AnalyzeTRTAlignment.py grabs the relevant information from the L2iter X directories and creates the output files AnalyzeAlignment.root and AnalyzeAlignment.log. AnalyzeAlignment.log is a text file containing the values of the alignment parameters determined and a report of any errors coming from failed jobs or missing files. AnalyzeAlignment.root contains plots of various quantities as a function of iteration. These include the change in the alignment parameters, the $\chi^2$, the number of tracks and hits, and the alignment parameters themselves. A separate file is created for the L1 and L2 alignment iterations and the plots are made separetly for each alignable object. (eg:each TRT module at L2 or the TRT Barrel and Endcaps at L1) Also included in the file for the L2 alignment is a visualization of the TRT Barrel module displacements in the x-y plane and rotations around the global z-axis. During the FDR each of these validation methods proved valuable both in term catching simple mistakes and in validating the ultimate alignment.

The FDR exercises were the first real tests the TRT alignment infrastructure designed for use on the

---

3) Throughout the rest of this section X will refer to the iteration number an Y to the sub-job number.

4) the exact prefix is set in RunTRTAlignment.py

5) Recently the functionality to run post processing on the monitoring files had been added.
CAF and in with the CERN batch service. Throughout the exercises many improvements were made, and functionalities added. Thanks to the FDR activities, the set of tools for the running TRT alignment on the batch that now exist are flexible and easily extensible. Through the FDR and beyond, the aim has been to evolve these tools from something accessible only to experts, into a well defined set of tasks for a shifter. The FDR has provided the first real opportunity to realize this goal.

### 3.3 Center of Gravity Algorithm

During the FDR exercise the Centre-of-Gravity (CoG) algorithm was applied twice in each alignment round: to the silicon system after its standalone alignment and to the entire ID after completion of the TRT alignment. The sole purpose of the CoG algorithm is to correct back any effective change of the generalised centre of gravity\(^6\) of the detector system as an artifact of the unconstrained alignment\(^7\) [4]. This step is required because, according to the ATLAS-wide consensus, the ID provides the alignment reference for the rest of ATLAS (calorimeters, muon system). Lack of the explicit CoG algorithm could result in an uncontrolled and artificial drift of the entire ATLAS system in its nominal reference frame used for physics analysis [5].

The FDR used a very simple algorithm for the CoG correction. It calculated the required correction to the global position and orientation of the system according to the following formula:

\[
\Delta \text{CoG}_i = \frac{1}{N} \sum_k \frac{\partial G_i}{\partial L_j} \Delta L_k^j \quad \text{with} \quad i, j \in \{1, \ldots, 6\},
\]

where the sum goes over all the detector elements, \(\Delta L_k^j\) denotes the alignment corrections of element \(k\) in the local frame and \(\frac{\partial G_i}{\partial L_j}\) is the jacobian transformation from the local to the global frame. More specifically it takes the form:

\[
\frac{\partial G_i}{\partial L_j} = \begin{pmatrix}
\frac{\partial \text{Tra}}{\partial \text{tra}} & \frac{\partial \text{Tra}}{\partial \text{rot}} \\
\frac{\partial \text{Rot}}{\partial \text{tra}} & \frac{\partial \text{Rot}}{\partial \text{rot}}
\end{pmatrix}
\]

where the capital and the small letter symbols for translation and rotation are used to denote the global and local frame respectively. This way the correction to the 6D CoG of the detector system is defined as the average of the 6D changes of the individual detector elements. The algorithm has been implemented under the name of \textit{InDetAlignCog} algorithm in the \textit{InDetAlignGenAlgs} package.

Only recently this approach has been diagnosed to be unsafe as it can lead to sub-optimal results under certain circumstances. Discussion of this problem is out of the scope of this note. Let us only note that recently new algorithm based on the least square minimisation of all detector element displacements replaced the described above one.

### 4 Determination of the Beam Spot

In the ATLAS data processing scheme [6], beam spot reconstruction runs both on the Inner Detector Calibration Stream and on the Express Stream. On the Inner Detector Calibration Stream it runs as part of the alignment procedure using a track-based algorithm in order to provide a beam spot position that can be used as a constraint by the alignment algorithm. On the Express Stream, a vertex-based beam spot algorithm is used to derive both the position and width of the beam spot for subsequent use by reconstruction and analysis jobs. During FDR-2, beam spot reconstruction was exercised on both of these streams. In the following, only the beam spot reconstruction on the Inner Detector calibration

---

\(^6\) Understood as the “average position and orientation” in the 6 DoF’s in space.

\(^7\) An internal alignment (irrespective of the method) leaves the 6 DoF’s of the global position and orientation undetermined.
stream is discussed. More comprehensive documentation on beam spot reconstruction in ATLAS can be found in a separate note [7]. Monitoring of the beam spot is described in [8].

The Inner Detector Calibration Stream does not contain full events and therefore a track-based beam spot algorithm is used that can determine the beam spot position even if only a single track is available from each event. The track-based algorithm has been implemented as the InDetBeamSpotTrackChi2 tool in the InDetBeamSpotFinder package. It relies on the correlation between the impact parameter $d_0$ and the azimuthal angle $\phi_0$ of tracks in the transverse plane. Four parameters of the beam crossing area are reconstructed:

1. $X$: $x$ intercept at $Z = 0$,
2. $a$: $x$ slope as a function of $Z$,
3. $Y$: $y$ intercept at $Z = 0$,
4. $b$: $y$ slope as a function of $Z$.

The impact parameter of a track with respect to the assumed beam position is given by:

$$ip = d_0 - [\sin\phi_0 * V - \cos\phi_0 * W],$$

$$V = X + a * Z_0,$$

$$W = Y + b * Z_0,$$

where the meaning of the above geometrical variables is best explained by the $XY$-view sketch in Figure 3. A simple $\chi^2$ minimisation of impact parameters 3 of all considered tracks with respect to the four beam parameters can be done analytically. As the problem is linear it does not require any approximations. By construction and by the requirement to work on single-track calibration stream the track-based algorithm does not measure the width of the beam spot. Details of the implementation and performance of the track-based algorithm can be found in [9].

A beam spot position is reconstructed for a set of $N$ consecutive luminosity blocks (LB), where the value of $N$ is determined based on how many events with suitable triggers and how many tracks per event are available per LB, how stable the beam spot is over time, and how precisely it needs to be known. During FDR-2, $N = 5$ was used in most cases, resulting in $O(10k)$ tracks available for each beam spot calculation.
In order to reconstruct the beam spot, the relevant information from all tracks in the $N$ consecutive LBs must be processed in a single job. This is achieved by a two-step process: In the first step, the different byte stream files coming out of the SFOs are reconstructed individually by separate reconstruction jobs. Each of these jobs runs the Inner Detector reconstruction code (InDetRecExample) and produces a DPD with a track collection made without using any beam spot constraints. In the second step, a single job reads all relevant DPDs for a set of $N$ LBs and calculates the corresponding beam spot. For the data samples available in FDR-2, the second step was so fast that all beam spot positions for the whole data sample could be calculated in a single job within a few minutes.

The output of this two-step process is, in addition to log files with a detailed printout of beam spot parameters, a COOL SQLite file with the calculated beam spot position(s) that can be used as input in subsequent alignment jobs (first pass beam spot), or can be uploaded to the conditions database if desired (second pass beam spot).

5 Upload to the Database

The constants produced during the FDR-exercise were used for the reconstruction of the data at Tier-0. The information was located in two files: A root file containing the alignment numbers and a SQL file with the relevant information like the tag name, the interval of validity (IoV) and the folder. The files were passed to the reconstruction via the database and therefore the constants had to be uploaded beforehand. The sample used for the ID alignment during the FDR2 described in Section 2.2 was produced in a standalone procedure and not produced within the general FDR2 Monte Carlo sample generation. This lead to the complication, that the files produced by ID alignment algorithm contained incorrect timing information. In particular the IoV was not consistent with the run numbers of the official Monte Carlo sample. For this reason the IoV of the alignment files had to be changed.

The constants produced by the alignment algorithm were located in the ConstantsFolder (/afs/cern.ch/user/a/atlidali/w0/ConstantsFolder/). A new SQL file with the consistent IoV was produced using the following command:

AtlCoolCopy.exe
"sqlite://schema=/afs/cern.ch/user/a/atlidali/w0/ConstantsFolder/day1/FinalConstants.db;dbname=OFLP200"
"sqlite://;schema=idalign_remap.db;dbname=OFLP200" -create -folder /Indet/Align -tag cog.tag -outtag InDetAlign-FDR-02
-aliov -nrls 52280 0 -nrlu 52285 4294967295
for the Si-part of the Inner Detector were FinalConstants.db corresponds to the file with the wrong IoV and idalign_remap.db to the file with IoV run 52280 to run 52285. The database tag name was also changed to InDetAlign-FDR-02. Similar the IoV was changed for the TRT:

AtlCoolCopy.exe
"sqlite://schema=/afs/cern.ch/user/a/atlidali/w0/ConstantsFolder/day1/FinalConstants.db;dbname=OFLP200"
"sqlite://;schema=trtalign_remap.db;dbname=OFLP200" -create -folder /TRT/Align -tag cog.tag -outtag TRTAlign-FDR-02
-aliov -nrls 52280 0 -nrlu 52285 4294967295
The root file with the corresponding constants was then uploaded using the following command:

/afs/cern.ch/user/a/atlcond/condmgr/registerFiles of1cond.000001.conditions.recon.pool.v0000
Finally the new files idalign_remap.db and trtalign_remap.db with the correct IoV were then uploaded to the database with the command
for the Silicon part and
for TRT constants. During the FDR several alignment tags have been produced and uploaded to the database. The alignment sets produced during the FDR-exercise are summarized in Table 5.

<table>
<thead>
<tr>
<th>tag name</th>
<th>used in which offline database tag</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>InDetAlign-FDR-01</td>
<td>OFLCOND-FDR-02-01-00</td>
<td>FDR2 exercise, tag for processing express stream</td>
</tr>
<tr>
<td>InDetAlign-FDR-02</td>
<td>OFLCOND-FDR-02-02-00</td>
<td>FDR2 exercise, tag for reprocessing express stream</td>
</tr>
<tr>
<td>InDetAlign-FDR-03</td>
<td>OFLCOND-FDR-02-03-00</td>
<td>FDR2 exercise, tag for processing bulk reconstruction</td>
</tr>
<tr>
<td>InDetAlign-FDR-04</td>
<td>OFLCOND-FDR-02-04-00 and OFLCOND-FDR-02-05-00</td>
<td>FDR2a exercise, ES2 processing</td>
</tr>
<tr>
<td>InDetAlign-FDR-05</td>
<td>OFLCOND-FDR-02-08-00</td>
<td>FDR2b exercise, ES1 processing</td>
</tr>
<tr>
<td>InDetAlign-UPD2-FDR-01</td>
<td>OFLCOND-FDR-02-06-00</td>
<td>FDR2c exercise, ES1 processing</td>
</tr>
<tr>
<td>InDetAlign-UPD2-FDR-03</td>
<td>OFLCOND-FDR-02-07-00</td>
<td>FDR2b exercise ID August 5 processing</td>
</tr>
<tr>
<td>InDetAlign-UPD2-FDR-04</td>
<td>OFLCOND-FDR-02-09-00</td>
<td>FDR2c exercise, ES2 processing</td>
</tr>
<tr>
<td>InDetAlign-UPD2-FDR-05</td>
<td>OFLCOND-FDR-02-10-00 and OFLCOND-FDR-02-10-01</td>
<td>FDR2c exercise, bulk reconstruction</td>
</tr>
</tbody>
</table>

Table 1: Alignment tags produced within the FDR2 exercise.

6 Monitoring

The monitoring of the Inner Detector Alignment is described in a separate note [8].

7 Summary

The production of track based ID alignment constants was successfully tested within the FDR exercise. A track sample simulating the ID calibration stream was used to produce a new set of alignment constants and a consistent beam spot. The alignment constant production run in a parallel environment to enhance the turnaround of the produced constants. The whole alignment production chain run with the 24h loop of the ATLAS calibration chain. Constants were passed to the monitoring and applied to the final reconstruction of events.

References


