Updates of the ATLAS Tracking Event Data Model (Release 13)

T. Cornelissen, M. Elsing, A. Wildauer
CERN
N. van Eldik, E. Moyse
University of Massachusetts, USA
W. Liebig
NIKHEF, Amsterdam, The Netherlands
N. Piacquadio
Albert Ludwigs Universität Freiburg, Germany
K. Prokofiev
University of Sheffield, UK
A. Salzburger*
Leopold Franzens Universität Innsbruck, Austria & CERN

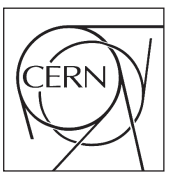
December 6, 2007

ATL-SOFT-PUB-2007-003
13 December 2007

Abstract

In a previous document [1] we have presented the ATLAS tracking *Event Data Model* (EDM) that has been developed during the recent restructuring of the ATLAS offline track reconstruction. The tracking EDM has become a cornerstone of the new modular track reconstruction algorithms of both tracking devices of the ATLAS detector, the Inner Detector and the Muon System. Recently, some components have undergone yet another design evolution targeted at completing missing modules and at establishing anticipated functionality for the startup of the ATLAS experiment. One particular aspect of the EDM is that is does not only have to fulfill the requirements of today's algorithmic modules, but has to provide the flexibility for future developments. This document is based on the ATLAS software release 13.0.40.

ATLAS NOTE
The ATLAS Experiment, *http://www.atlas.ch*



*corresponding author: Andreas.Salzburger@cern.ch

1 Introduction

A common event data model (EDM) is inevitable for the component software structure given by the ATHENA framework [2]. It allows to define interfaces for the single tasks of the overall reconstruction process. In other words, it defines the language that is *spoken* by the reconstruction algorithms. In a previous paper [1] we have presented the concepts, design and implementation of the ATLAS tracking EDM to very detail. The tracking EDM has become a cornerstone for the main ATLAS track reconstruction algorithms in both the offline and third level trigger (*Event Filter*) application.

The tracking EDM has been deployed at full production level for almost three major release circles and has undergone several extensions and modifications, adapting either to new algorithmic developments or simply as an outcome of constant evaluation of the complied functionality. Recently, yet another substantial design evolution has taken place which was mainly motivated to complete missing modules and establish full functionality for first data taking with the ATLAS detector. A new, neutral track parameterisation has been included to enhance constrained vertex fitting that includes both charged and neutral particle representations. Additionally, the common `Track` class has been extended to store material descriptions on reference surfaces that have been used in the track fit. Pseudo-measurements have been introduced to stabilise poorly constrained fits, and the segment representations have been coherently modified to enhance the inclusion of the pseudo-measurement objects.

A major design revolution has taken place for the representation of the tracking information for physics analyses, the `TrackParticle`. This has been motivated to achieve full support of `TrackParticle` objects by common tracking tools[1] and to optimise the class shape for a common use of track representations from both tracking devices, the *Inner Detector* (ID) and the *Muon Spectrometer* (MS).

This document describes the extensions and modifications that have been applied to the tracking EDM since release 12.0.0 and presents — if necessary for the understanding of the presented context — a short review of the various modules that are integrated in the tracking EDM. This document is based on ATLAS offline release 13.0.40, a consecutive production release on basis of the ATLAS release 13.0.0. The authors are fully aware that this document only presents the corresponding status of the EDM at one particular point in time. The EDM is the language that is used for the communication between the individual parts of the full reconstruction chain — and consequently, like every language, it is matter of modification and evolution. Future changes to the tracking EDM may thus be presented in consecutive documents.

1.1 The `Track` Class

The ATLAS Tracking EDM is concentrated around a flexible `Track` class that is realised as a container of `TrackStateOnSurface` (TSOS) objects. The single `TrackStateOnSurface` objects are able to hold polymorphic tracking information: track expression with respect to a given surface in form of track parameters objects, hit information by an extended class of a common `MeasurementBase` class, traversed material or integrated material effects, a `FitQuality` object and an identification type. Table 1 lists the possible base classes that can be held by the `TrackStateOnSurface` class. The fitted track representation to the nominal beam line or vertex position, the so-called `Perigee`, is e.g. represented by one `TrackStateOnSurface` object, just like every single hit, a track segment, or even a fitted scattering angle. The flexible container structure of the `Track` object guarantees hereby that one single trajectory representation can be used for different fitter types, tracking devices or reconstruction algorithms. With release 13.0.0, the way how to store material interactions on a `Track` object has been changed (see Sec. 6), which led to an extension of the `TrackStateOnSurface` class; Figure 1 shows an illustrated track as a collection of `TrackStateOnSurface` objects.

Track fitting requires, in general, two main data structures: the representation of measurements to be integrated in the track fit, and the representation of the (fitted) trajectory, in the following referred to as *track parameterisation*. In ATLAS, track fitting is carried out on calibrated hits or hit collections that extend a common base class, the `MeasurementBase`. The track parameterisation is done using classes that extend a `ParametersBase` interface, restricted to a charged parameterisation for child objects that can be stored in the `Track` class. This is because neutral particles are not matter of tracking in the sense of track finding and fitting, since they do not cause hit signatures in the tracking

[1] E.g. the extrapolation engine or the vertexing tools.

Table 1: The different (base) classes that can be contained by the `TrackStateOnSurface` class. All of the given classes are contained and returned by pointers, thus they can be left out optionally.

Leaf Package	Description
`FitQualityOnSurface`	χ^2 and number of degrees of freedom
`TrackParameters`	the track parameterisation (or intersection) w.r.t. a surface
`MeasurementBase`	the measurement expression
`ScatteringAngleOnTrack`	a fitted scattering on track
`MaterialEffectsOnTrack`	applied material effects description

devices. The found tracks are input to higher level reconstruction modules such as vertex fitting or topological event fitters that include neutral particles in particular for mass-constrained fits. Common tracking tools such as the extrapolation engine or the various vertex fitters need to operate on both neutral and charged track representations. The formerly existing model that was limited to charged track parameterisations has therefore been extended to cope with charged and neutral trajectories in a type safe way. The newly integrated schema is presented in further detail in Sec. 2, the adaption of the vertexing data model to this new base class is described in Sec. 5. The full object tree of measurement representations that has been recently adapted with pseudo-measurements and modified track segment representations is presented in Sec. 3.

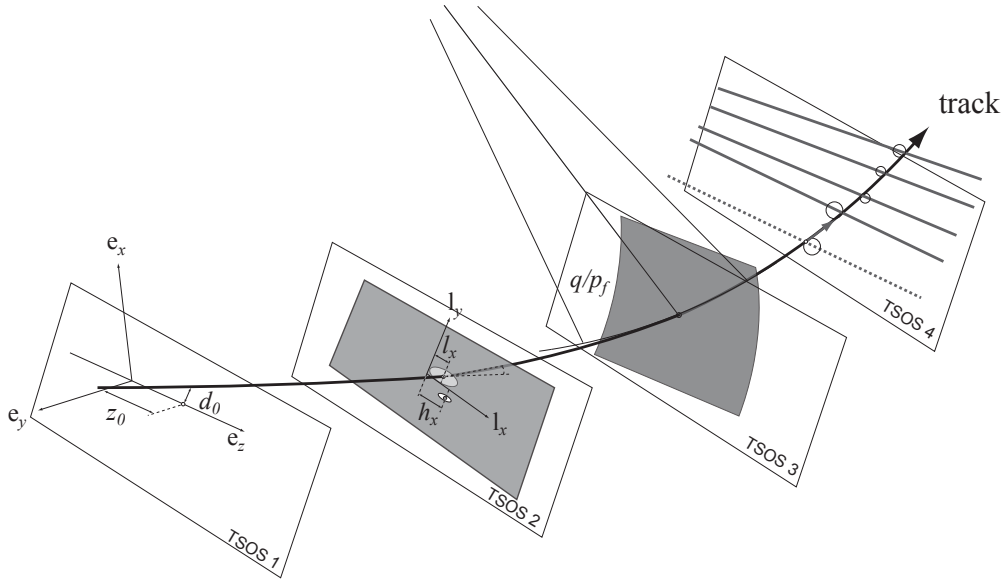


Figure 1: An illustrated `Track` as a container of four different `TrackStateOnSurface` (TSOS) objects. TSOS 1 contains the track representation to the nominal beam line, the so-called `Perigee` representation. TSOS 2 holds both a measurement through a hit information and the fitted track parameters on the measurement surface, while TSOS 3 represents an applied material interaction — illustrated through a curvature change of the track. Finally, TSOS 4 integrates several fitted measurements through a single segment representation.

1.1.1 Track Slimming

One important aspect of the tracking EDM is the size which the written data takes when being stored on disks. This is far less trivial than it first sounds: on the one hand, small *persistent* representations of the tracking EDM are necessary to comply with the computing budget of the experiment, but on the other hand, as much information as possible should be accessible for the physicist to allow optimal event analysis techniques. In the context of the tracking EDM, a major step towards achieving a good balance between disk size and usability was to identify all information that can be recreated when the track is read back from disk storage. In principle, this includes all fitted parameters and estimated

covariances, but excludes obviously the hit collection. A simple refit such a *slimmed* track after it has ben read from the persistent storage would recreate the full track information as achieved in the original event reconstruction. The flexible TSOS container design of the `Track` class was hereby a key feature, since it allows to create a track collection of stripped hits and a `Perigee` representation[2] that is then written to disk. The track collection size could be significantly reduced (depending on the track collection, the reduction factor varies between 6 to 10).

Representation for Physics Analysis Few analyses based on data taken with the ATLAS detector will directly incorporate the `Track` objects. The `Track` itself is, in general, not more than a trajectory representation of the particle when passing through the detector, while the — for the event analysis — most important representation of the particle as a four momentum vector at the production vertex is not given by the `Track`; neither is particle identification[3] nor the vertex association performed at the stage of track reconstruction. In the ATLAS EDM, the `Track` information is represented as a `TrackParticle` object for further use in a particle-oriented event analysis. Vertex fitting with or without constraints can be performed on `TrackParticle` objects, but needs the extrapolation engine to express the trajectory with respect to the (iteratively fitted) vertex position. To enhance common tracking tools to work together with the `TrackParticle` object (which combines a broader bundle of aspects to be dealt with in event reconstruction), without breaking the philosophy of keeping the tracking modules independent from specific reconstruction algorithms, a new `TrackParticleBase` class has been introduced that concentrates the tracking-relevant information and builds the new interface for tracking tools. These tools are designed to operated also on event reconstruction and analysis level; a detailed description of the new `TrackParticleBase` class can be found in Sec. 4.

2 Trajectory Parameterisation: The `ParametersBase` class

The parameterisation of a particle trajectory with respect to a given surface is inevitable for track reconstruction. It can be done in many different ways, for a charged trajectory in magnetic field a minimal set of five parameters has to be chosen; it can be reduced by one parameter for a trajectory representation in a no-field environment or a neutral particle that follows a straight line. This is, since the charge q and the momentum magnitude p are superfluous for the purely geometrical description of a line. For constrained vertex fitting that includes both charged and neutral particle traces, however, the momentum (hypothesis) is necessary — see Sec. 5.

The trajectory parameterisations for both neutral and charged particles are thus realised in the ATLAS tracking EDM as a set of five parameters

$$\mathbf{x} = (l_1, l_2, \phi, \theta, c/p)^T, \tag{1}$$

when l_1 and l_2 denote the local coordinate expression on the given surface (and thus depend on the surface type), ϕ and θ are the azimuthal and polar angle, respectively, and c is defined as

$$c = \begin{cases} q & \text{if } q \neq 0, \\ \\ 1 & \text{if } q = 0. \end{cases} \tag{2}$$

For every surface type that is defined in the ATLAS reconstruction geometry [4], a dedicated parameterisation exists, realised by a specific class to ensure an unambiguous identification of the given measurement frame. In track fitting — since the trajectory itself can not be measured, but only a localisation at discrete points in the detector can be done — a set of *measurement mapping* functions h_j is needed to map the track parameterisation on a measurement surface to the measured coordinates and thus to establish a *predicted* measurement[4]. This yields for the single predicted measurement

[2] This is for the simple convenience of the user that is not forced to refit the track collection if the focus is only drawn onto the impact parameterisation.

[3] Only a `ParticleHypothesis` exists for the steering of material effects integration.

[4] Since the two most common track fitting techniques, the least squares method and the Kalman filter are both linear estimators, these measurement functions are even required to be linear, or at least approximated by a linear function.

components

$$m_j^{pred} = h_j(\mathbf{x}), \tag{3}$$

or, when using matrix notation and assuming linear measurement mapping functions at the detection device i

$$\mathbf{m}_i^{pred} = \mathbf{H}_i \mathbf{x}_i. \tag{4}$$

The predicted measurement is necessary when building the hit residuals $\mathbf{r}_i$, where the measurement position $\mathbf{m}_i$ on the ith device is compared with the predicted or fitted trajectory intersection on the measurement surface, explicitly

$$\mathbf{r}_i = \mathbf{m}_i - \mathbf{m}_i^{pred} = \mathbf{m}_i - \mathbf{H}_i \mathbf{x}_i. \tag{5}$$

Since in ATLAS the track parameterisation on a given surface has been chosen to be expressed in local surface coordinates, the measurement functions are simply given by projection matrices, yielding

$$\mathbf{H}_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{6}$$

for a one-dimensional measurement in the first local coordinate, and

$$\mathbf{H}_3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \tag{7}$$

for a two-dimensional position measurement, respectively. Most actual measurements taken with the technologies of the ATLAS sub-detector are given in these two forms. However, the ATLAS tracking EDM is not restricted to measurements of this type. Instead, every possible combination of sub-sets measured out of the full parameterisation can be chosen, which is exploited by the new `PseudoMeasurementOnTrack` class and the `Segment` classes in particular, see Sec. 3. This leads to 31 different projection matrices $\mathbf{H}_i$ and requires in addition an internal identification schema for the parameter vector[5]. The `LocalParameters` class that extends the `HepVector` class of the CLHEP [3] maths library establishes a mechanism that provides the user with an internal identification schema — realised in a bitwise packed key i — of the contained parameters and the appropriate projection matrix. An example of the simple usage of the `LocalParameters` class can be found in the Appendix, Sec. A.2. In this sense, the chosen track parameterisation can be regarded as being motivated by the measurement setup of the detector. Figure 2 shows a track expressed w.r.t. two different reference surfaces in the ATLAS tracking EDM.

Since the parameterisations of a trajectory state with respect to a given surfaces differ for charged and neutral particles only slightly in the interpretation of one parameter, it would be a natural attempt to represent them with the same EDM object. Equation (2), however, shows the potential danger of such a choice: a particle with charge $q = 1$ would then be expressed through the same signature as a neutral particle represenation of same momentum. One main design principle of the ATLAS tracking EDM is *type safety*, i.e. any misinterpretation of a given data object has to be avoided[6]. It can be argued that a simple check of the charge parameter q could distinguish the two identical cases, but this relies on every user to perform this check before the parameter vector can be interpreted. A type safe model has to be favored preferably without code duplication. In ATLAS, this is realised through a C++ template method that allows to define the actual class type through the template parameter. A small helper class, the so-called `ChargeDefinition` that is extended by a `Neutral` and a `Charged` class is hereby used to define the appropriate template argument of the given track representation. A charged trajectory — when being expressed with respect to a planar surface — is then defined as a class extending an `AtaPlaneT<Charged>` class, while a neutral trajectory representation is enhanced by an `AtaPlaneT<Neutral>`. Since the common implementation is shared by the two classes, but type safety is preserved through the template mechansim, both main requirements — type safety and avoiding code duplication — have been met. The `ChargeDefinition` classes define for convenience an implicit cast operator to a `double`, returning zero for the `Neutral` and ± 1 for the `Charged` class, respectively. Since the charge is not stored otherwise in the parameter classes, it is impossible to construct an inconsistent object.

[5] There are, in total, $32 = 2^5$ possible subsets of parameters, but the empty set is excluded, since there exist no invalid measurement representations in the ATLAS tracking EDM.

[6] Clearly no one wants to have a curved photon track in magnetic field, simply for the fact that the used propagation tool misinterpreted the $1/p$ parameters of a neutral representation as a q/p information.

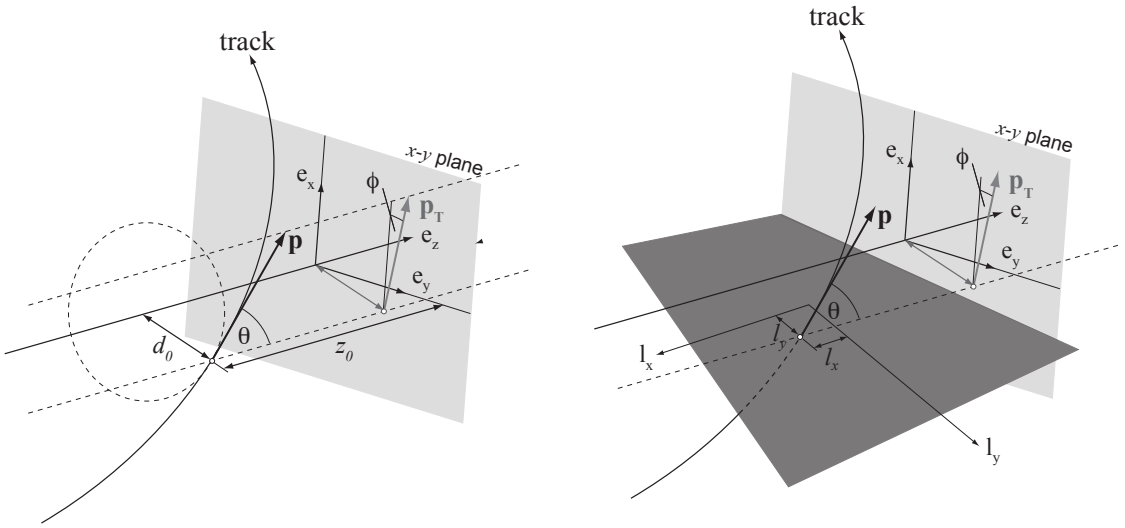


Figure 2: A track parameterised with respect to two different surfaces: the expression to the nominal z axis yields the `Perigee` representation of the track to the left, while the expression of an intersection with a planar surface (right) is described by the `AtaPlane` object. The parameterisations differ only in the first two local coordinates that are defined by the surface type and are optimised with respect to the given detector layout. The momentum expression through the azimuthal angle ϕ, the polar angle θ and the (charged) inverse momentum is identical for both cases.

Hidden Template Method The authors are aware that template solutions are in general not amongst the most popular techniques within the client community and track representations belong clearly to the most widely spread classes of the ATLAS tracking EDM. The template resolving has therefore be *hidden* from the user through inserting actual class types for the track parameterisations on the various surfaces for charged and neutral particles that extend the class templates to non-virtual objects[7]. Figure 3 shows an UML class diagram that illustrates the charged and neutral track parameterisation with respect to a planar surface.

The `ParametersBase` base class is restricted to the attributes that are identical for both a neutral and a charged trajectory parameterisation and can be used for applications that only work on the global parameters of a trajectory expression, i.e. a position, a momentum and the charge. The template mechanism, on the other hand, forces the client to resolve the template argument and consequently an object has to be identified to be either of `Neutral` or `Charged` flavor, before the parameters vector can be retrieved[8].

3 Measurement representation: The `MeasurementBase` Class

Measurement representations exist in manifold ways in the ATLAS tracking EDM: in most of the cases, measurements are directly integrated as fully calibrated representations clusters or drift radii. These objects are realised as classes that extend the `RIO_OnTrack` class, and represent either one-dimensional or two-dimension measurements; the calibration applied on the input objets from the clusterisation process (in ATLAS terms `PrepRawData` objects) is hereby based on the already collected track information. In the MS, a second additional calibration step is applied on `RIO_OnTrack` objects in the preparation phase for track fitting (*pre-tracking*), that is based on the local pattern recognition output for the various detector chambers.

As described in [1] an even more flexible way of representing single and combined measurements with a extended `MeasurementBase` object has been implemented in ATLAS. These types include pre-grouped (and fitted) measurements as `Segment` realisations and a dedicated competing measurement collection

[7] The technically interested reader may find that the class templates mark virtual class descriptions and can thus not be instantiated in the program flow.

[8] In C++ terms this is done using the `dynamic_cast` operator.

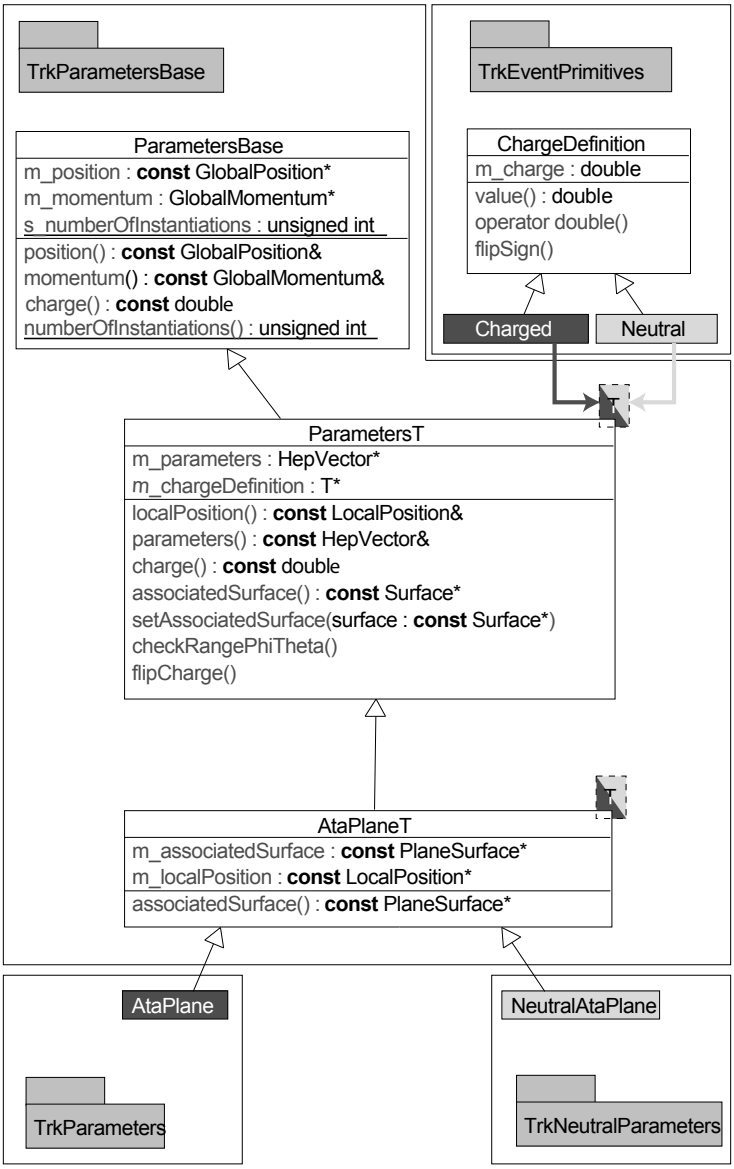


Figure 3: The charged and neutral track parameterisation as realised in the ATLAS tracking EDM. A `ChargeDefinition` base class and its extended `Charged` and `Neutral` classes are located in the TrkEventPrimitives package. The two child classes are used to define the type of the templated trajectory representations — in the illustrated example with respect to a `PlaneSurface` of the ATLAS reconstruction geometry.
The `flipCharge()` and the `flipSign()` methods are hereby necessary just for technical aspects, but are not accessible for the client.

(`CompetintRIOsOnTrack`) on one detection surface that enhance fuzzy hit assignment techniques. Both classes can be resolved into their constituents and thus can the hits be individually included in the track fit. On the other hand, the `Segment` class represents already several measurements at different discrete positions in the detector and holds, in general, more information than a simple localisation of the track trajectory on a reference surface. This is in many cases at least one directional parameter of the momentum vector, or even a momentum estimate itself. The `Segment` classes can therefore be used for fast track matching and play a particularly important role in the MS pattern recognition.

`RIO_OnTrack` and `Segment` objects are mainly used in track fitting or local pattern recognition. For pattern recognition, a global representation is often used for trajectory seeding on the one hand, or for global pattern techniques (such as histogramming methods or Hough transformations) on the other hand. The `SpacePoint` class object establishes global hit representations of measurements. A `SpacePoint` can hereby represent a single or a collection of measurements if the spatial information provided by one measurement together with the measurement surface is not sufficient to constrain a global position.

All these different measurement representations extend one single base class, the `MeasurementBase` interface. The `MeasurementBase` concentrates the minimal information needed for track fitting, i.e. a one- to five-dimensional measurement, the according measurement errors and the corresponding

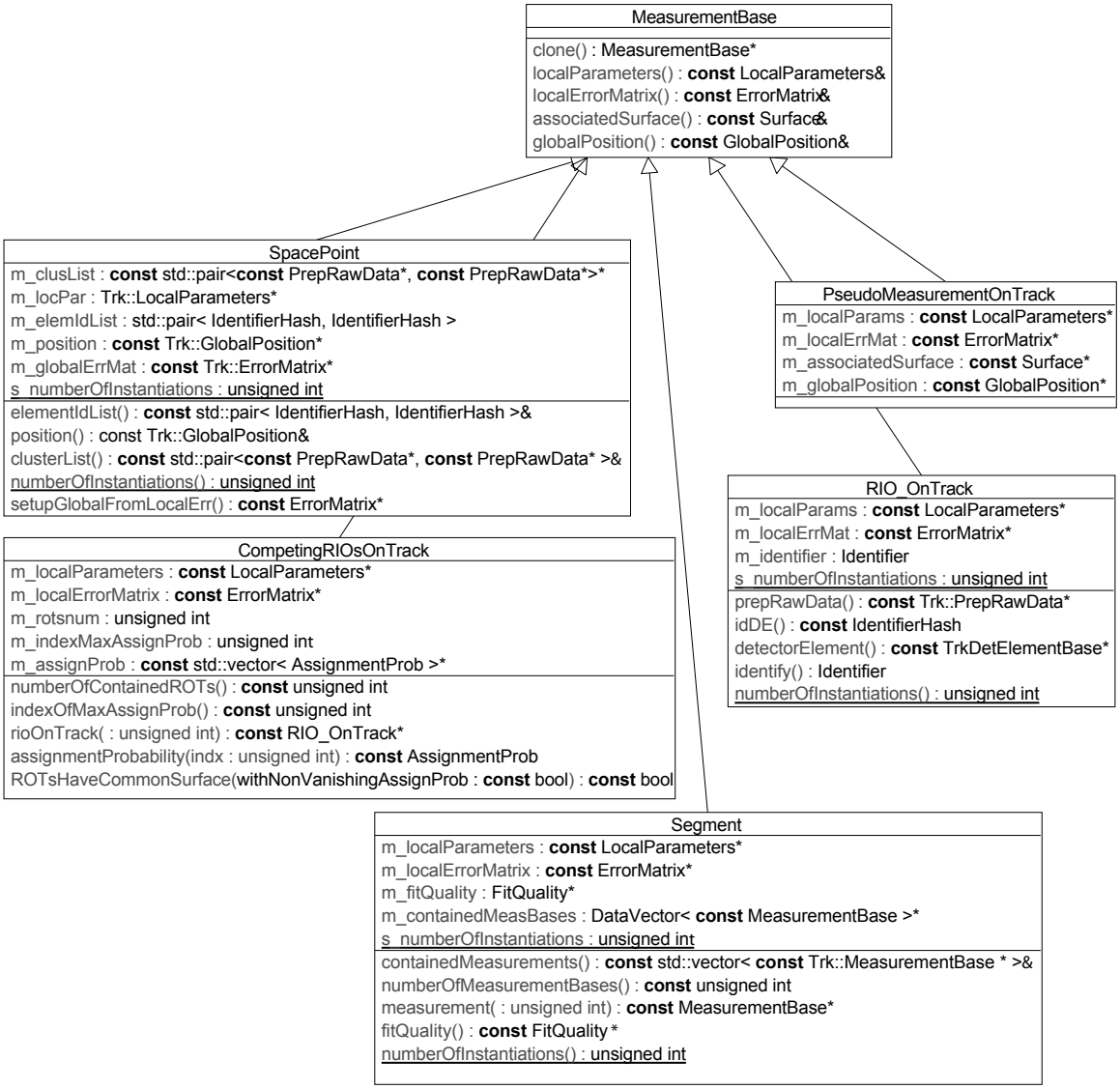


Figure 4: Simplified UML diagram, showing the classes that extend the `MeasurementBase` class and can be used for the representation of one to many measurements in the ATLAS tracking EDM. Only the base classes of the inherited families are shown, concrete detector-specific implementations of most of the classes can be found in the corresponding sub-detector repositories.

reference frame that is provided by a `Surface` object of the ATLAS reconstruction geometry. In fact, the `ITrackFitter` interface is defined to operate only on a set of `MeasurementBase` objects and thus common track fitter implementations are completely shielded from actual measurement representation. A polymorphic track fit, combining several different types is often performed and eases successive hit inclusion, track segment matching or simply saves CPU time when a prior fit has already compacted the information of several single hits into one representation.

Pseudo-Measurement Representations In ATLAS, there exist several tracking devices that can not determine the full track parameterisation without information from other devices: e.g. the *Transition Radiation Tracker* (TRT) in the Inner Detector or the *Monitored Drift Tube* (MDT) chambers in the Muon Spectrometer. Although a full track parameterisation can not be obtained, it is necessary that isolated measurement sets from these devices can serve as an input for track fitting. For the ID this in particular important to enhance track segment search from the detector boundary towards the

interaction point. In the MS, on the other hand, the track finding starts often with a segment search on chamber level. To constrain the track fit it is necessary to include an artificial measurement with a crude estimate of the missing coordinate. This estimate can be at the level of *within the chamber* or *in the barrel*, i.e. something that can be included without biasing the fit result but can be postulated on the other hand by the pure existence of the measurement. A generic `PseudoMesurementOnTrack` has therefore be created for such a purpose. It extends the `MeasurementBase` base class, and implements the minimal interface necessary for fitting. The UML class diagram can be seen as part of Fig. 4

Track segment adaption to the pseudo-measurements The inclusion of fake measurements evoked a necessary adaption of the `Segment` classes. Segments are often used for local pattern recognition in the track fit of hit collections from single detector components that are prone to lead to unconstrained fits. It is thus necessary that `Segment` objects can contain the new `PseudoMeasurementOnTrack` objects which led to a shift from `PrepRawData` to `MeasurementBase` as the top level class contained by hit collections in a single track segment. Since segments are the most complex objects in the `MeasurementBase` tree and depend on the algorithm that has formed them, a new authorship schema has been included to achieve transparency about the object origin in higher level reconstruction algorithms. A list of currently registered `Segment` authors can be found in Sec. A.3.

4 The Analysis Object: the `TrackParticleBase`

In the ATLAS computing model, there exist two main levels of event data processing, that are reflected by the written data collections: the *Event Summary Data* (ESD) and the *Analysis Object Data* (AOD). ESD based algorithms and modules are in total part of the event reconstruction, and clearly first stage analyses to be done with the ATLAS detector, mainly targeted at evaluating the detector performance and calibrating or aligning the detector components will need access the event data to full detail level. However, recording and transferring all data taken with the ATLAS detector to remote sites at ESD level would exceed the computing capacities of the experiment and — in particular — of the attached institutes. Therefore a compression of the ESD to AOD is done, that should still provide sufficient information for modern analyses techniques, while reducing the disk storage significantly. The track representation in the AOD containers is the so-called `TrackParticle`. Since the `TrackParticle` combines information from tracking devices, calorimetry and particle identification algorithms, it can not be subject of the tracking EDM. Until recently this resulted in the fact that the `TrackParticle` class was not known to common tools used in track reconstruction. Reconstruction tools therefore acted mainly as wrappers for common tracking tools, forwarding the only available information — the measured track representation at the `Perigee` definition close to the interaction point — to common tracking tools such as vertex fitters and the extrapolation engine. In particular for tracks being found with stand-alone reconstruction algorithms in the Muon Spectrometer, this situation was far from optimal.

Recently, the `TrackParticle` class has undergone a major design evolution that established a new `TrackParticleBase` class in the `Tracking` repository. The introduced `TrackParticleBase` concentrates the tracking relevant data, while remaining independent from higher level reconstruction algorithms. The restrictive definition of the `TrackParticle` to be defined only through a `MeasuredPerigee` object has vanished in the base class and has been replaced by a vector of `ParametersBase` objects. Yet another reason why an intervention has become inevitable was the integration of the neutral track parameterisation that, consequently, has to be followed in the data hierarchy down to the analysis representation. However, keeping in mind that all inherited child classes are aimed to describe objects to be used in physics analyses and thus a Lorentz-vector representation should be existent, one of the parameters provided has to be used to define the four-momentum vector. This parameter has to be identified explicitly at the construction of the `TrackParticleBase` object. The widely used `TrackParticle` now inherits from the `TrackParticleBase` class and again restricts the *defining parameter* to be the `MeasuredPerigee` representation (or, e.g. the `NeutralMeasuredPerigee` for a neutral particle representation). This is, since the analysis of the underlying event clearly deals with the particle parameters closest to production vertex. Figure 5 shows a simplified UML class diagram for the newly created `TrackParticleBase`, Fig. 6 illustrates how the muon `TrackParticle` profits from the more flexible way of holding parameters at several stages within the detector.

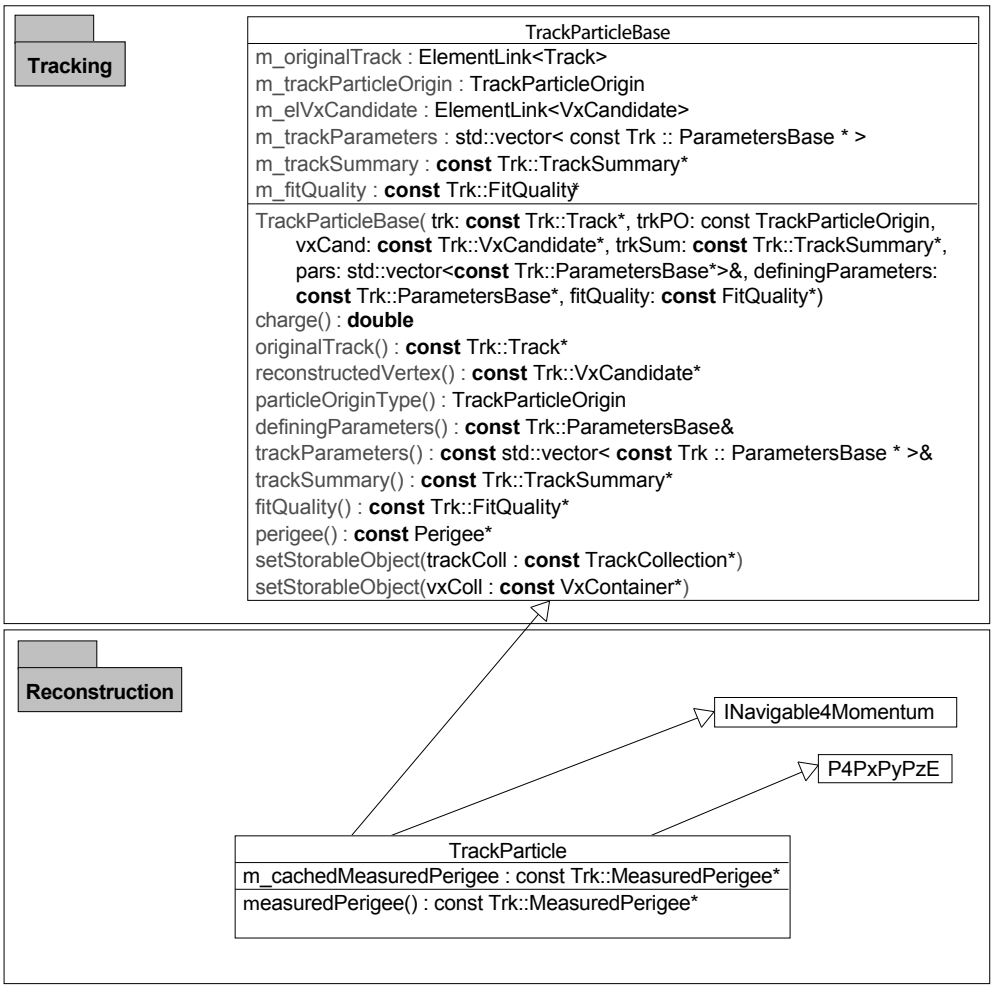


Figure 5: Simplified UML diagram, showing the new `TrackParticleBase` and the extended class `TrackParticle` that marks the analysis representation of tracking. The constructor of the `TrackParticleBase` shows the new philosophy that allows multiple representations of the underlying track within the detector, while keeping one `ParametersBase` object specifically outstanding to identify the track state where the four-momentum is defined.

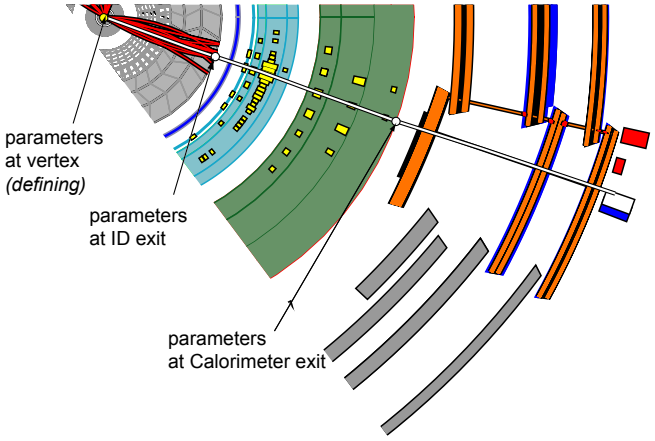


Figure 6: The new `ParticleBase` object illustrated in an example based on the ATLANTIS [5] event display. The `Track` is hereby represented with one single `TrackParticleBase` object at three different stages in the detector: as a `MeasuredPerigee` expression close to the interaction point (defining parameters), through `TrackParameters` at the exit of the Inner Detector and the Calorimeter, respectively.

5 Event Data Model for Vertex Reconstruction

The part of the ATHENA event data model that is related to the vertex reconstruction has changed significantly between releases 12 and the latest production release on top of the 13.0.0 base release. In general, this changes are related to the introduction of two new EDM concepts: the use of the `ITrackLink` and the use of the `Trk::ParametersBase` class — which is described in more detial in Sec. 2. The new parameterisation base class replaces the formerly used `Trk::TrackParameters` class, that is limited to charged trajectories. This marks a first step of preparation towards the vertex reconstruction EDM for fully constraint and kinematic fitting algorithms[9]. Starting from ATHENA rel. 13.0.20, the dependencies on the `Trk::TrackParameters` are dropped in all classes of the vertexing EDM; the use of `Trk::ParametersBase` is introduced instead. This change has been motivated by the fact that the `Trk::ParametersBase` is the new common base class for both neutral and charged track parameters. The future inclusion of neutral trajectories in the vertex fits thus becomes possible[10].

5.1 The `Trk::ITrackLink` class

The `ITrackLink` class is a new part of the Tracking EDM, implemented in the dedicated TrkTrackLink package. The class itself is purely abstract and so far has only two concrete implementations: the `Trk::LinkToTrack` class in the `TrkTrack` package and the `Trk::LinkToTrackParticleBase` in the `TrkParticleBase` package. The latter two classes also inherit from the `ElementLink <TrackCollection>` and `ElementLink <TrackParticleBaseCollection>` respectively.

This schema of inheritance allows one to use a pointer to the `Trk::ITrackLink` to manipulate both a `Trk::Track` and `Trk::TrackParticleBase` and thus use the same EDM components on both AOD and ESD levels.

An example of the use of a pointer to the `Trk::ITrackLink` as a private member, can be found in the new version of the `Trk::VxTrackAtVertex` class. Here the use of the `ITrackLink` is justified by the fact that it gives the access to initial parameters of tracks used to fit a vertex, disregarding, whether the fit was done on the AOD or the ESD level. This allows to reduce significantly the storage size of reconstructed vertices: the initial perigee parameters of each trajectory does not need be stored anymore.

The disadvantage of the use of the `ITrackLink` is that the user needs to perform a `dynamic_cast` operation each time the knowledge of the exact type of the `ElementLink` is required. The parameters of the trajectory, however, can be accessed directly via the a dedicated access method.

6 Material Effect Description on Track

The correct treatment of effects caused by the interaction of the particle with detector material is inevitable for track reconstruction. A dedicated reconstruction geometry [4] is therefore built and provides a simplified version of the ATLAS detector setup to track reconstruction algorithms. The integration of material effects happens, in general, during the track fit itself and is thus not necessarily matter of the event data model. However, the tracking EDM has been recently expanded to store the applied corrections due to material interactions directly on the track class. This can be either a fitted parameter from global track fitting, realised through a `ScatteringAngleOnTrack` object, or a more profound description of the traversed material in terms of radiation length X_0 and applied energy loss corrections. Latter is made possible through a `MaterialEffectsOnTrack` object. Both descriptions can be stored as primary classes on a `TrackStateOnSurface` instance, see Sec. 1.1.

Allowing to store material effects together with the track object has been motivated by the following considerations:

- recent developments, e.g. in electron track fitting, can provide information about the applied material effects for further processing, such as track kink finding, or calorimeter cluster matching;

[9] In a second step, which is currently under preparation, a new extended track parameterisation will be introduced that adds the mass parameter to the existing track parameterisation as described in Sec. 2

[10] The obvious disadvantage of this change is the need to `dynamic_cast` the parameters returned by a particular class in order to understand whether this is a charged or a neutral trajectory.

- stored material effects on track will allow refitting of tracks from different sources that use different material descriptions, in particular for global refitting, where several competing material descriptions exist. This enhances a dedicated validation and comparison of the different material integration algorithms;

Both `MaterialEffectsOnTrack` and `ScatteringAngleOnTrack` have been extended to host a surface which defines the spatial information along the track, where the actual update had been integrated.

Material Integration from custom Sources The `MaterialEffectsOnTrack` class has also become the data object for custom material integration into the track extrapolation engine [6]. In track extrapolation, the description of the material is, in general, taken from a simplified detector description. The extrapolation engine, however, allows users to optionally provide custom material effects to the extrapolation process. The integrated `MaterEffectsOnTrack` objects can hereby originate from other parametric descriptions, material maps, or even from detector measurements such as the energy loss application based on calorimeter measurements for combined track reconstruction.

7 Conclusion

Four years after the *Final Report of the Reconstruction Task Force* (RTF) [7] invoked a redesign of the ATLAS reconstruction software (based on a common EDM and a component software structure), and less than one year before data taking with the deployed ATLAS detector, the tracking EDM is close to completion. The tracking EDM classes provide the necessary functionality for all current aspects of track reconstruction and have been optimised in both functionality and memory usage. The tracking EDM has been recently adapted for new developments and expanded to support common tacking tools in the user event analysis.

It has proven its stability and functionality during test beam data taking and the commissioning runs using cosmic rays, while being flexible enough to integrate new developments and corrections without breaking existing service. The ultimate aim is to provide a powerful EDM for the long lifetime of the experiment that allows backward compatibility for reading data during the entire period of data taking.

A Appendix

A.1 CVS Repository Structure

The described EDM classes in this document can be entirely found in the Tracking/TrkEvent container package of the ATLAS CVS software repository [8]. The referred surface classes are located in the Tracking/TrkDetDescr/TrkSurfaces package, and the event primitives, e.g. the position vectors, algebraic vectors and matrices are either part of the CLHEP math library or bundled in the TrkEvent/TrkEventPrimitives utility package.

A.2 The `LocalParameters` Class

While track parameterisations always provide the full five parameter set to describe the track (see Sec. 2), measurements can be expressed as any subset of one to five parameters of the full parameterisation (Sec. 3). In track fitting, where usually residuals on measurement surface are build that are the differences between the predicted measurement that is given through the track parametersiation and the track extrapolation and the actually taken measurement, it is necessary to collaps the full parameters vector or the track parameters to the ones represented by the measurement. This is done by so-called measurement mapping functions. Measured parameters can hereby exceed the simple localisation of a track on a surface, since track segments can also be included into a track fit as a measurement (and usually include already directional information).

In the tracking EDM, the full parameter vector is expressed through a five-dimensional `HepVector` object from the CLHEP maths library. The `HepVector` class is not restricted to any dimension and

in a naive approach the measurement representation could simply use the same class to represent the (mostly lower-dimensional) measured parameters. However, for the client algorithm it is not obvious which of the five parameters — or what subset of the five parameters — is stored in the `HepVector` object. To save both disk space and the consumption of physical memory it is however desired to store only the obtained parameters of the measurement in the appropriate data class. Clearly this can be handled by convention, but in a generic model, where track measurements can be used without knowing the underlying detector technology, this is far from trivial. The solution to this problem has been the introduction of the `LocalParameters` class that extends the `HepVector` by an identification schema. The measurements are hereby expressed by a key which is the `integer` representation of a binary number that puts 0 for non-measured and 1 for measured.

A pixel measurement that localises both coordinates on a silicon module is then represented as a binary number 11000. For technical aspects that are based on the fact the the overwhelming majority of measurements represent purely local coordinates, the digits of the binary number are flipped, and the identification key is calculated as the transformed `interger` representation, yielding in the given example:

```
int(0b00011) = 3
```

Given the identification through the integer key it is possible to check the measured parameters contained by the `LocalParameters` class. Some examples of the usage of the `LocalParameters` class will be shown in the following.

```
// retrieve the local parameters from the measurement
const Trk::LocalParameters& measPars = measurement->localParameters();

// check whether it measured the local Y coordinate
bool measLocY = measPars.contains(Trk::locY);
```

The argument given through the method signature of the `contains(...)` method is the the same `enumeration` type as used for accessing the parameters from the track parameterisation, the so-called `ParamDefs`. The `ParamDefs` are also used to define the parameter(s) to be put into the `LocalParameters` class at construction level.

```
// created a defined parameters - it is a segment that defines x/phi
Trk::DefinedParameter measuredX(0.452, Trk::locX);
Trk::DefinedParameter measuredPhi(1.452, Trk::phi);

// create a vector of defined parameters
std::vector<Trk::DefinedParameter> defParameters;
defParameters.push_back(measuredX);
defParameters.push_back(measuredPhi);

// finally create the LocalParameters
Trk::LocalParameters segParameters(defParameters);
```

For the most common usecases there exist dedicated constructors for the convenience of the clients. The `LocalParameters` class also provides the projection matrices that correspond to the measurement mapping functions, which are in the ATLAS tracking EDM simply realised as projections matrices. A static member of type `ProjectionMatricesSet` that holds the 31 possible matrices is therefore registered to the `LocalParameters`. A residual calculation as given in Eq. 5 can then be calculated as

```
// get the track parameters vector - always 5-dimensional
const HepVector& parameters = trackParameters->parameters();

// retrieve the local parameters from the measurement - n-dimensional ( n <= 5 )
const Trk::LocalParameters& measPars = measurement->localParameters();
```

```
// build the n-dimensional residuum vector
HepVector residuum = measPars - measPars.reductionMatrix()*parameters;
```

It is worth noticing that the last line hereby is only possible since the `LocalParameters` object extends the `HepVector` and thus all operators are defined accordingly. The transposed measurement mapping matrices $\mathbf{H}_i^T$ are also accessible for the convenience of the user.

A.3 Segment Authors

The adapted `Segment` class now incorporates an authorship schema for later identification of the algorithm used for the segment creation. The authorship is hereby registered by an `enumeration` object; the current list of possible segment authors can be found below.

```
enum Author {
      AuthorUnknown = 0,
      MooMdtSegmentMakerTool = 1,
      MooCscSegmentMakerTool = 2,
      Muonboy = 3,
      DCMathSegmentMaker = 4,
      MDT_DHoughSegmentMakerTool = 5,
      CSC_DHoughSegmentMakerTool = 6,
      Csc2dSegmentMaker = 7,
      Csc4dSegmentMaker = 8,
      TRT_SegmentMaker = 9,
      NumberOfAuthors = 10
};
```

References

[1] F. Akesson et al., *The ATLAS Tracking Event Data Model*, ATLAS Public Note, ATL-SOFT-PUB-2006-004.

[2] Athena homepage, *http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/ architecture.*

[3] CLHEP homepage, *http://cern.ch/clhep*

[4] A. Salzburger, M. Wolter, S. Todorova, *The ATLAS Reconstruction Geometry Description*, ATLAS Communication to be published, ATLAS-COM-SOFT-2007-009, 2007.

[5] ATLANTIS homepage, *http://cern.ch/atlantis*

[6] A. Salzburger, *The ATLAS Extrapolation package*,ATLAS Communication to be published, ATLAS-COM-SOFT-2007-008, 2007.

[7] V. Boisvert et al, *Final Report of the ATLAS Reconstruction Task Force*, ATLAS Note, ATL-SOFT-2003-010, 2003.

[8] ATLAS software CVS reporsitory, online CVSView, *http://atlas-sw.cern.ch/cgi-bin/viewcvs-atlas.cgi/offline/*