ATLAS File and Dataset Metadata Collection and Use

2012 J. Phys.: Conf. Ser. 396 052005

(http://iopscience.iop.org/1742-6596/396/5/052005)

Journal of Physics: Conference Series **396** (2012) 052005 doi:10.1088/1742-6596/396/5/052005

ATLAS File and Dataset Metadata Collection and Use

S Albrand[1], E J Gallas[2], J Fulachier[1], F Lambert[1], on behalf of the ATLAS Collaboration.

[1]Laboratoire de Physique Subatomique et Corpusculaire, Université Joseph Fourier Grenoble 1, CNRS/IN2P3, INPG, 53 avenue des Martyrs, 38026, Grenoble, FRANCE

[2]Department of Physics, Oxford University, Denys Wilkinson Building, Keble Road, Oxford OX1 3RH, UNITED KINGDOM

E-mail: solveig.albrand@lpsc.in2p3.fr

Abstract. The ATLAS Metadata Interface ("AMI") was designed as a generic cataloguing system, and as such it has found many uses in the experiment including software release management, tracking of reconstructed event sizes and control of dataset nomenclature. The primary use of AMI is to provide a catalogue of datasets (file collections) which is searchable using physics criteria. In this paper we discuss the various mechanisms used for filling the AMI dataset and file catalogues. By correlating information from different sources we can derive aggregate information which is important for physics analysis; for example the total number of events contained in dataset, and possible reasons for missing events such as a lost file. Finally we will describe some specialized interfaces which were developed for the Data Preparation coordinators. These interfaces manipulate information from both the dataset domain held in AMI, and the run-indexed information held in the ATLAS COMA application (Conditions and Configuration MetadatA).

1. Introduction.

AMI is a framework consisting of relational databases which contain their own description, a software layer to exploit this auto-description, and a set of generic interfaces for the usual database functions (insert, update, delete and select). AMI can manage different database schema deployed on geographically distributed servers with different relational database management systems (RDBMS) simultaneously. This enables AMI to support schema evolution in a seamless way.

The AMI framework [1] is used by ATLAS for several functions, including the Tag Collector application [2] , the cataloguing of dataset nomenclature reference lists and the tracking of reconstructed event sizes [3]. The principle use of AMI in ATLAS is to provide a catalogue of the official datasets of the experiment, searchable on physics criteria.

Datasets are collections of files which are either:

- Real data from the data acquisition of the ATLAS detector.
- Simulated data or reprocessed real data, from one or more production system tasks.

ATLAS metadata information is available from many sources, with different granularities (run, stream, dataset, file, production task number…). Users need to be able to navigate from one granularity to another in order to find and combine the data from these different sources. This is not a trivial task because the information is distributed across many applications, and each source application presents a different interface and exports the data in a different format. A mediator interface is defined as "a software module that exploits encoded knowledge about some sets or subsets of data to create information for a higher layer of applications" [4]. To put it another way, a mediator is an application which puts some domain expertise between the user and a group of data sources, so that information coming from these different sources can be aggregated. AMI aims to be a mediator interface for ATLAS physics metadata.

This article describes how the AMI dataset and file catalogues are filled by pulling data from the different data sources of the experiment, or in the case of the COMA application, by sharing of data. It gives some examples of the metadata parameters which can be derived by aggregation of information from these sources.

In the next section the different sources of data are briefly described. Section 3 gives some detail of the server side of the data puller operations. Following this we consider each of the communication mechanisms in turn, and discuss their relative merits, and scalability. Lastly we give several examples of derived quantities which aggregate information from several sources.

2. Sources of AMI metadata.

Almost no information is entered directly in the AMI databases. Instead the application pulls most of its information from several different sources. Each source has its own database(s) and corresponding schema. The source may be completely passive with respect to the data puller (ProdSys), supply special features for AMI (Tier0 and COMA) or implement a specialized messaging system for an AMI client (DDM).

2.1. List of data sources and metadata available.

2.1.1. Atlas Tier 0. Tier 0 [5] is the application which forms the files of data coming from the Detector Acquisition into datasets of "real data", and then controls the first processing of the data.

2.1.2. ATLAS production system and production task request (ProdSys). Physicists make requests to the production system for simulation tasks or for reconstruction of real data. These task requests are held in a dedicated database. A task maps to a set of jobs to be executed on the grid usually with a single input file of data. Once tasks pass to the "RUNNING" state their status can be followed in another database. When all the jobs defined are completed the production task is marked "FINISHED". At this point metadata for all the successful jobs is available for reading in the ProdSys DB. A production task may contain only one job, or as many as 100 000 jobs.

2.1.3. ATLAS Distributed Data Management. (DDM) [6]. All datasets are registered in the DDM system, which manages the physical location and existence of datasets. DDM manages the replication of datasets to different sites, and also the deletion of data. It is very important for AMI to know when data has been deleted or even lost. DDM also knows when the output of a production task is placed in a dataset, and this is essential information for calculating the number of files and events available for users.

2.1.4. ATLAS Conditions Metadata (COMA) [7]. COMA is a relational view of the ATLAS Conditions Database which stores all detector conditions indexed by run and luminosity block, or by interval of validity. The Conditions database has been defined for efficient and flexible storage of different conditions, which may change as calibration is performed. However it is not at all convenient

for user queries. COMA uses highly specialized knowledge of the conditions database to extract run metadata, and reorganize it in a schema to facilitate SQL queries.

3. The AMI framework task server.

The AMI framework includes a task management system. This system is in fact an instance of the AMI server which runs independently from the other AMI services. It is fundamentally a master thread which controls the running of other threads (see figure 1). Each of these threads is an AMI task. The code to run for the task, the frequency of task running, the priority, and any mutual exclusion with respect to other tasks are all defined within the central AMI database. Thus a task can be deactivated, or its frequency of running increased simply by editing the master task table. Data is entered by a set of specialized "data puller" tasks controlled by the task server.

Information is available from different sources in a chaotic way. The data puller does not know in advance whether a large number of datasets is about to become available for reading from the production system, and one cannot allow the reading of a large amount of information from one source to cause a backlog in the reading of information from another source. Therefore each task has a maximum amount of work it is allowed to do each time it runs. According to the task, this may be defined in the number of datasets treated, or the number of production tasks treated. Data puller tasks which do not finish all their work must store a stop point which is usually the source timestamp at the time of the last successful data treatment in AMI. This is not a time sharing mechanism; and only heuristic methods have allowed us to obtain a combination of work "bite size" and frequency of task running. Little and very often is best.

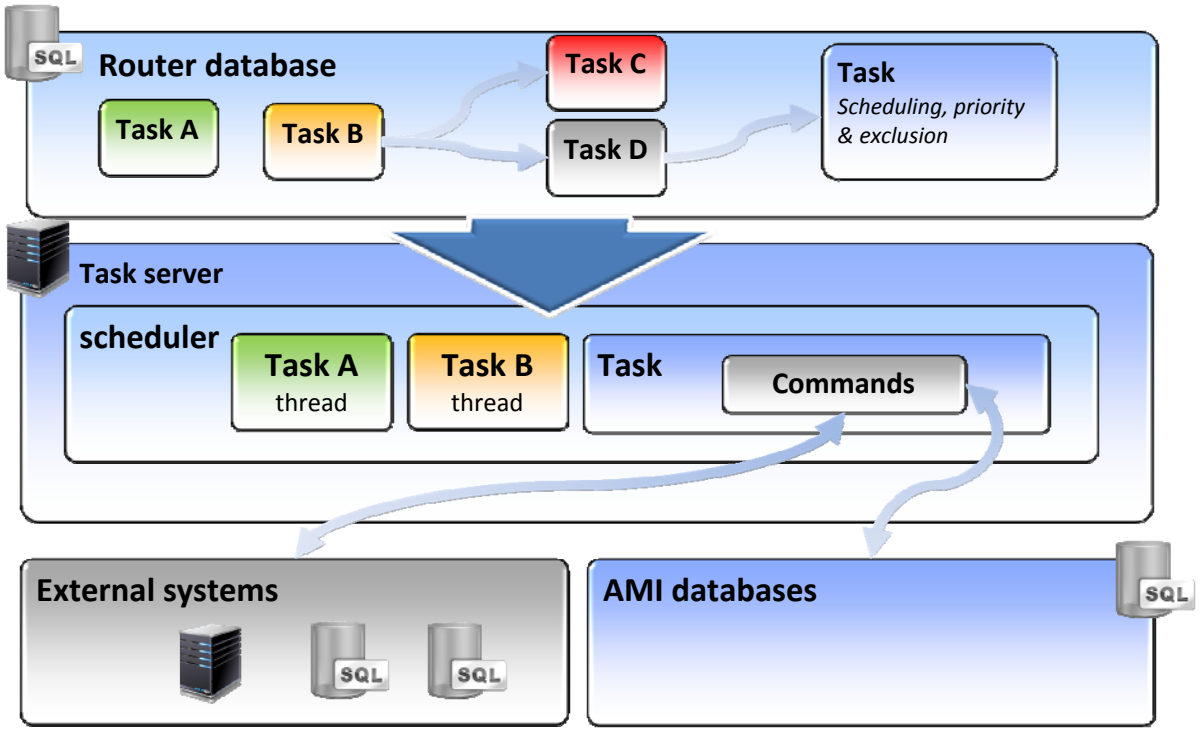


Figure 1. The architecture of the AMI Task Server. The "Router" database contains a list of tasks to be executed and their parameters of scheduling etc.

Each task has its own log file. Tasks record when they start, when they finish, and what they did. In principle any serious task error will send an email message to the AMI team. Fortunately this is a comparatively rare occurrence, and is usually due to some error or unforeseen change in the upstream input. A master log file records the start and stop times of all tasks – so one can see which tasks were running concurrently.

The AMI task server has a watch dog system which will intervene should the main log file be untouched for more than one hour.

4. Tier 0 : a semaphore mechanism.

Communication between Tier 0 and AMI works by a semaphore. Tier 0 management (T0M) determines which datasets should be registered in AMI. It is important to note that datasets are registered in the DDM system when they are empty, and they are filled with files as the acquisition proceeds. It is only when the dataset is completely filled in the DDM system that T0M raises the flag for AMI registration.

The AMI Tier 0 data puller task looks for new data about 60 seconds after the end of the last Tier 0 data pulling run. A selection of the information for datasets marked “READY” is copied from the Tier 0 databases. When this has been done, a coherence check verifies the copy. If there was no problem AMI lowers “READY” to “DONE”. AMI has write privileges on the flag column in the Tier 0 data base.

A maximum of 100 datasets is treated in any single reading run, and multi threading is used. On average a 9Hz. record insertion rate is achieved. Insertion is much faster for datasets with a large number of files.

Tier 0 is of course a highly critical component of ATLAS computing. As such it is provided with an excellent monitoring system, and a team of shifters. A Tier 0 shifter can tell if AMI is functioning correctly by observing the number of outstanding “READY” flags. This has been particularly useful for the AMI team since it is the only data source which monitors the AMI reading functions, and it is capable of detecting cases when the AMI task server is malfunctioning.

5. The production system: a timestamp mechanism.

Entering data into AMI from the production system is in fact done by several different AMI tasks, each with its own particular function. All the AMI tasks which read database tables in the production system rely on the automatically triggered last modification timestamps of the database records. The production system is completely passive, and the reader must decide which information is relevant

An ATLAS production task is a set of jobs; the number of jobs depends on the number of events processed by the task. A task must first be “requested”; it passes into production only after approval. The task request system takes care of registration in DDM, and releases an approved task for execution on the grid. As soon as a task is admitted into the production system AMI reads the names of the datasets which the task will form, ensuring coherence of naming. This task runs about every 5 minutes.

Once the production task is “RUNNING” AMI switches to following its status in the table of executing tasks. As soon as a production tasks is declared “FINISHED” AMI reads one record of information in the ProdSys database for every job of the task. These records contain in particular an XML field which reports the metadata for the files (names, size and number of events, cross section, etc.) produced by the job, and the metadata attached to both the complete production task, and hence to its associated datasets (conditions version, geometry version, trigger tags, etc.). As mentioned above, a production task may consist of only one job, or as many as 100 000 jobs. Reading the information for finished production tasks is the most time consuming of all the AMI data puller tasks. It is also the most error prone since the XML contents are defined by the job options, and a mistake can be easily made. We have adopted a “defensive” strategy of trying to be prepared for the unexpected. For example we detect and reject negative cross section values, and filter efficiencies of greater than 1.

The AMI task reading finished jobs has a 60 second rest period between each run. Production tasks are read in 20 task bites and treated in a multithreaded way. On average 7 finished jobs per second are treated. Database insertions are batched.

It can happen that a burst of simulation jobs all finishing at the same time provokes "indigestion" for AMI. Therefore we have put in place a mechanism to double our capacity should the backlog go above 1000 tasks. A second task server is started on another machine, dedicated to running only this AMI task. The two work in parallel, running over different sections of the backlog, partitioned by their time stamp.

Code profiling has revealed that there is some time to be gained in some of the functions used, by storing more information in database tables.

6. Distributed data management: publish & subscribe protocol

ATLAS DDM is the system which manages the physical location of datasets and files, and controls their transfer from one site to another. Registration in DDM is always upstream from registration in AMI, and there are many datasets in DDM which are never registered in AMI. For example the production system produces a certain number of transient dataset which have no interest for physics analysis and the grid site monitoring system sends a large number of test datasets. AMI at the present time does not catalogue any user analysis datasets, but this will without doubt change in the near future.

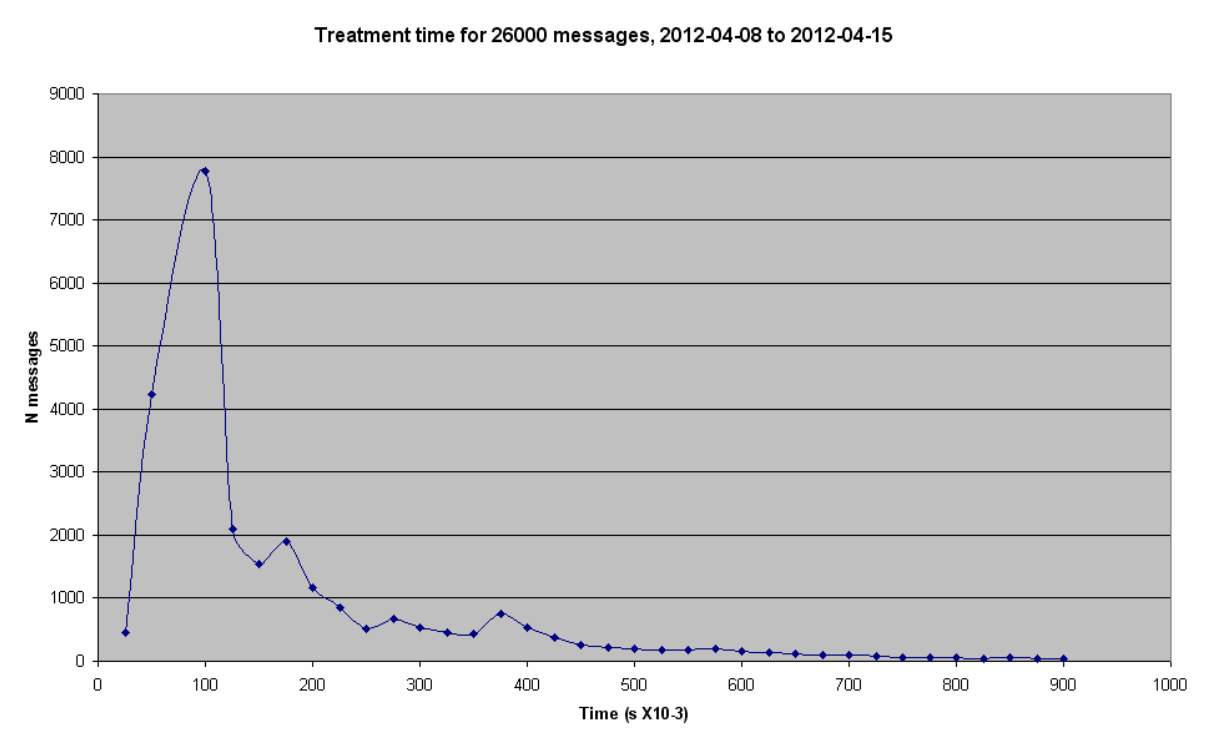


Figure 2. Analysis of the Active MQ logs for one week shows that different message types have different execution times. Almost all messages are treated in less than 500 ms.

DDM has put into place a “publish and subscribe” mechanism for communication with AMI, using Active MQ and Simple Text Orientated Messaging Protocol. (STOMP) protocol [8]. Information sent to AMI is of four kinds:

- Registration of a dataset
- Deletion of a dataset.
- Addition of production task output to a dataset.
- Removal of a production task output from a dataset.

At present AMI ignores the registration messages, since this information is obtained from the other sources. The other three messages must be analyzed, firstly to see if the dataset in question is catalogued in AMI, and then to determine the action required. The action required is usually a change of dataset “production status”[9] and a recalculation of the number of files and events in the dataset. It also may involve changing the “show or hide” status of the dataset. AMI does not show deleted datasets by default. At many points in the process AMI can perform a coherence check. For example it has been possible to detect cases where task output was signaled “added” to a dataset, when the task was still declared “RUNNING” in the ProdSys DB.

STOMP has proved to be reliable and fast. The only problem we experienced was when there was a peak in the number of datasets deleted and the publish queue became filled to capacity. To deal with this we increased the frequency of the AMI task runs.

In the case of STOMP, AMI always reads to the end of the message queue. An analysis of the time to treat a message shows clearly that a large number of messages received require a minimum of treatment. Almost all messages are treated in less than 500 ms. (see figure 2).

7. Conditions and configuration metadata: symbiosis

AMI and COMA are an example of mutualism. The two applications have different histories, and a different set of metadata parameters. AMI does not copy any data from COMA as is done for the other sources. Metadata parameters are completely shared. One could say that they "think" they are part of the same application. Parts of COMA were rendered "AMI Compliant” so that all the AMI generic code will work on COMA databases.

COMA has benefited from the AMI infrastructure, in particular pyAMI, the python web service client. Also AMI writes some aggregated quantities in the COMA DB.

AMI has benefited from the access to Conditions Data, which was quasi-impossible without the COMA formatting of this data.

In section 11 we describe an example of an interface developed for ATLAS Data Preparation which allows access to data in both COMA and AMI in a transparent way.

8. Discussion of the Relative Advantages and Disadvantages of Different Mechanisms.

After several years of experience it is possible to give an overall view of the various advantages and disadvantages of the different methods used for data reading or sharing.

From the AMI point of view the most satisfying relation is the synergy with COMA. But it is not suitable for all of the applications. COMA and AMI are both “downstream” applications; their function is to give physicist end users an aggregated view on ATLAS data. An application such as DDM must very closely control its clients; it is important not to allow clients free range on SQL queries which might affect the performance of the highly critical operations. Therefore the messaging system is very well adapted, as the client is completely decoupled from the data source database. Of course the client must rely on the source putting in place the message types which the client desires.

AMI reads ProdSys data mostly from tables which are effectively decoupled from the production system as they are repositories filled only before the tasks begin or after the task has finished. The critical transient job information is thus well shielded. Two main problems have been encountered by AMI when reading from ProdSys. Firstly it is not always completely coherent with itself, and a task can be for instance still marked “submitting” in one place and “FINISHED” in another. It is necessary

to add semantic knowledge to the AMI data pulling as these conflicting states are in different tables with different timestamps, and even if one is later than the other in time it cannot be later than the other in function. This particular problem is a known bug of ProdSys transaction management, and will become impossible when the next major version is released. The second problem has arisen only a few times, but when it does happen it requires manual intervention. If a schema change is required in one of the tables which AMI reads, then of course all the rows will be touched, and the reading mechanism breaks.

The Tier 0 semaphore method is a very good compromise. Readers are shielded from schema changes and other more aleatoric events. The system is compact, and there are no internal incoherencies. The reader can decide to enlarge what is copied, without intervention from the source application developers.

Table 1 Summary of the advantages and disadvantages of different methods used by AMI for obtaining dataset, file and run information.

Method	Advantages	Disadvantages
Semaphore	Elegant – No need to sift through all the information in the source database, but can read it if desired.	All insertion operations must be logged.
Timestamp controlled	Reader sees everything in source. Updates easy – just set clock back.	Any schema change in the or error of manipulation in the source must be anticipated by the reader (for example when the timestamps of all rows are updated)
Active MQ	Fixed message format and pre-arranged content. Completely decoupled from source application.	Many messages not useful – but must be read anyhow. Relies on the source to provide all the information. Careful logging needed
Data sharing	No data copying is involved, so no problems of coherence.	Some schema constraints imposed on both partners.

9. Scalability.

Whatever the method of insertion of data into the AMI databases it is imperative to consider the scalability. The number of files produced will increase over the next few years, and we must also bear in mind the requirement to catalogue the physics metadata for users.

We have spare capacity almost all the time (see figure 3); for example the ProdSys reading task often has nothing to do in spite of the observation of backlogs from time to time. It is evident that if

the ProdSys process which determines that production tasks are finished were to run more frequently, the work of the AMI data puller would be more evenly spread.

Profiling of the code has revealed functions which can be optimized, and not all the ORACLE operations are optimized.

Clearly insertion in AMI will always take longer than pure SQL insert operations on a database. Many coherence checks are performed, and quantities are derived. However we are at present inserting only a fraction of what other ATLAS databases are regularly achieving, it is safe to conclude that we will be able to scale up to the same level.

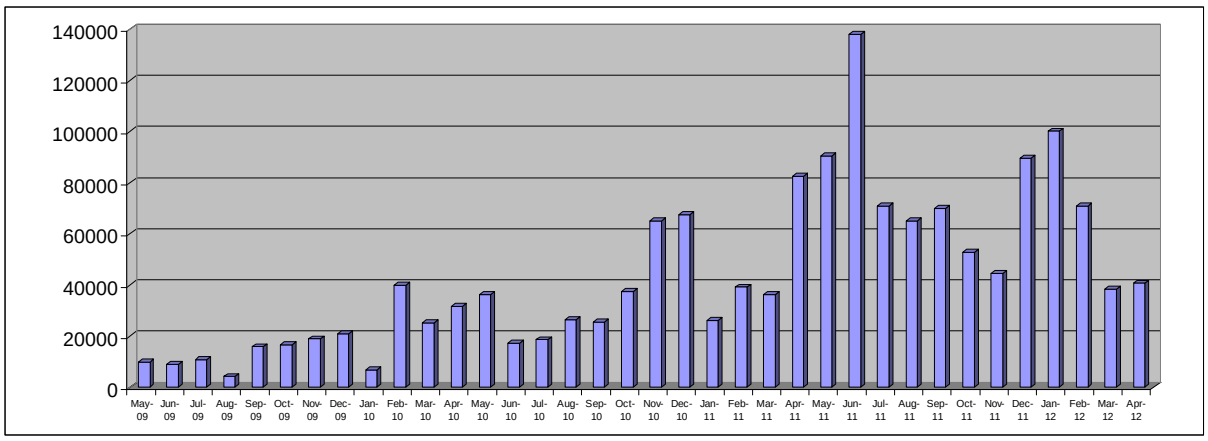


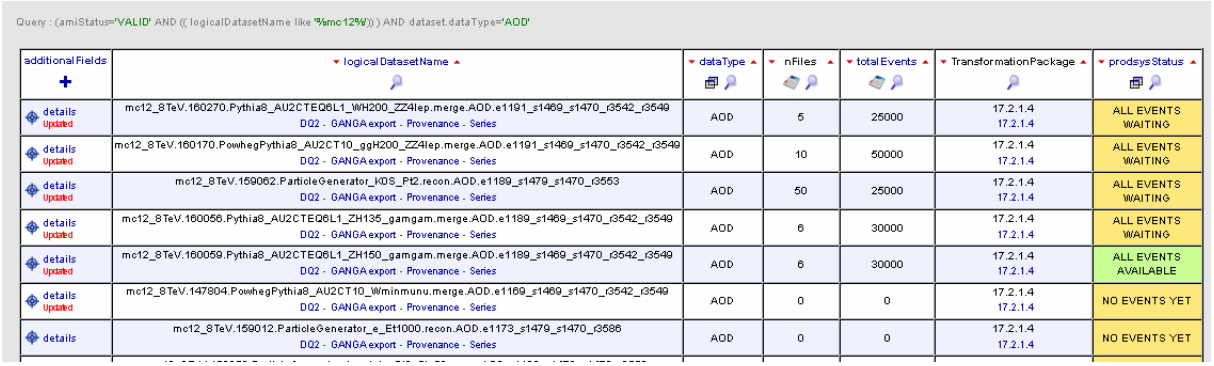
Figure 3. The number of datasets registered in AMI per month since May 2009. Note that the peak observed in June 2011 triggered the refactoring of the ProdSys data puller task. An 8-fold performance gain was achieved.

10. Some examples of derived quantities.

After each insertion of new information in the AMI databases some derived quantities are updated. The following list gives some examples. In each case the sources of the aggregated data are indicated:

- Complete dataset provenance (T0 & ProdSys). Every new dataset is inserted into its family tree, so that users can see the complete chain of input datasets, and also all datasets derived from a particular dataset. Family trees are held explicitly in a database table.
- File to file provenance (T0 & ProdSys). When a reprocessing task finishes, and the file list and metadata have been entered in the AMI catalogue, a further operation analyses the ProdSys database to extract the input file(s) for each job. Complete file to file provenance trees are only produced on demand, or if a missing file is reported, in which case the luminosity blocks lost can be extracted.
- Number of files and events in the dataset (T0, ProdSys, DDM). Every time new information is entered for a dataset the number of files and the number of events available for analysis is recalculated. Information may be the addition or removal of the output of one or more production tasks, or the loss of a file.
- Production Status[9] (T0, ProdSys, DDM). All datasets in AMI have a “production status” which is calculated from the state of the task or tasks which have provided or will provide events. This status value is colour-coded on the web page lists (Figure 4).
- Average, min and max cross sections (ProdSys). Every file of a Monte Carlo event generation task returns a cross-section value to AMI. When the dataset is complete an average, maximum and minimum value for the dataset is calculated. The average cross-section is transported down the provenance chain as each derived dataset is declared.

- Lost luminosity blocks in reprocessed data. (T0, ProdSys, DDM). The DDM has a service which tracks lost files, and attempts to restore them from replicas. There are rare cases when this is not possible, and a file of simulated or reprocessed data is declared to be lost. AMI monitors the list of lost files, and then by a recursion up the file provenance tree makes a list of the luminosity blocks which were in the lost file. The status of the dataset is changed to show that luminosity blocks have been lost.
- Run period reprocessing errors (ProdSys & COMA). Data Preparation coordination keeps a close watch on the validity of the reprocessed campaigns. AMI has provided functions to spot a few of the possible errors. Firstly if a reprocessing task finishes without completing all its jobs this is marked in the dataset status as a reprocessing error. Secondly a specialized interface compares the number of events obtained at each stage of the reprocessed chain with the number of events in the RAW datasets, and highlight differences.
- Datasets in run periods. (COMA). Using information in COMA, AMI marks the datasets containing run data from a particular run period.


Query : (amiStatus='VALID' AND ((logicalDatasetName like '%mc12%'))) AND dataset.dataType='AOD'

additional Fields	logicalDatasetName	dataType	nFiles	totalEvents	TransformationPackage	prodsysStatus
details Updated	mc12_8TeV.160270.Pythia8_AU2CTEQ6L1_WH200_ZZ4lep.merge.AOD.e1191_s1469_s1470_r3542_r3549 DQ2 - GANGA export - Provenance - Series	AOD	5	25000	17.2.1.4 17.2.1.4	ALL EVENTS WAITING
details Updated	mc12_8TeV.160170.PowhegPythia8_AU2CT10_ggH200_ZZ4lep.merge.AOD.e1191_s1469_s1470_r3542_r3549 DQ2 - GANGA export - Provenance - Series	AOD	10	50000	17.2.1.4 17.2.1.4	ALL EVENTS WAITING
details Updated	mc12_8TeV.159062.ParticleGenerator_K0S_Pt2.recon.AOD.e1189_s1479_s1470_r3553 DQ2 - GANGA export - Provenance - Series	AOD	50	25000	17.2.1.4 17.2.1.4	ALL EVENTS WAITING
details Updated	mc12_8TeV.160056.Pythia8_AU2CTEQ6L1_ZH135_gamgam.merge.AOD.e1189_s1469_s1470_r3542_r3549 DQ2 - GANGA export - Provenance - Series	AOD	6	30000	17.2.1.4 17.2.1.4	ALL EVENTS WAITING
details Updated	mc12_8TeV.160059.Pythia8_AU2CTEQ6L1_ZH150_gamgam.merge.AOD.e1189_s1469_s1470_r3542_r3549 DQ2 - GANGA export - Provenance - Series	AOD	6	30000	17.2.1.4 17.2.1.4	ALL EVENTS AVAILABLE
details Updated	mc12_8TeV.147804.PowhegPythia8_AU2CT10_Wminmunu.merge.AOD.e1169_s1469_s1470_r3542_r3549 DQ2 - GANGA export - Provenance - Series	AOD	0	0	17.2.1.4 17.2.1.4	NO EVENTS YET
details	mc12_8TeV.159012.ParticleGenerator_e_Et1000.recon.AOD.e1173_s1479_s1470_r3586 DQ2 - GANGA export - Provenance - Series	AOD	0	0	17.2.1.4 17.2.1.4	NO EVENTS YET

Figure 4. Part of the list of ATLAS 2012 simulated data. Events are waiting when production has finished, but the output is not yet placed in the dataset.

11. Hybrid interfaces developed for Data Preparation and Reprocessing.

Since AMI and COMA share their data, it has been possible to develop interfaces which access both of them transparently, and in particular, some features which are heavily used by the Data Preparation coordination. Runs which have been marked “Ready for Physics” (COMA information) are placed in a “Data Period” (written in COMA by an AMI interface). Users can then select datasets for this run corresponding to a physics stream and a set of reconstruction parameters (AMI information) to create a super dataset known as a “physics container”. The AMI interface registers the super dataset in DDM and then in AMI. Figure 5 shows a screen shot of the web version of this interface. Physics containers may also be made using a pyAMI command.

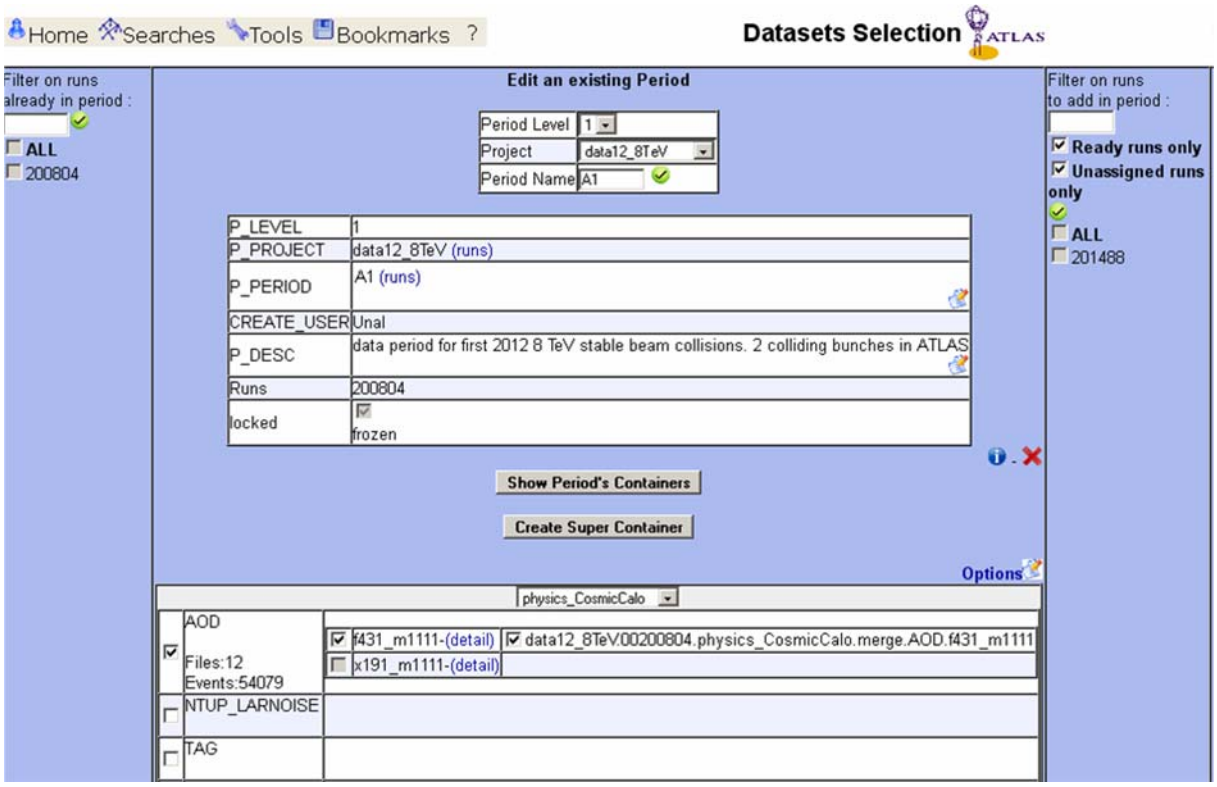


Figure 5. A screen shot of the AMI web interface for making "physics containers" based on a selection of runs. The run and data quality information is taken from COMA, the dataset, files and number of events is taken from AMI in a totally transparent way for the users.

References

[1] Albrand S, Fulachier J and Lambert F, The ATLAS metadata interface 2010 *J. Phys.: Conf. Ser.* **219** 042030 **doi:10.1088/1742-6596/219/4/042030**

[2] Obreshkov E, Albrand S, Collot J, Fulachier, J , Lambert, F, Adam-Bourdarios C, Arnault C , Garonne V , Rousseau D, Schaffer A et al., 2008 Organization and Management of ATLAS Offline Software Releases, *Nuclear Inst. and Methods in Physics Research, A,* **Vol 584**, pp 244-251

[3] Rousseau D, Dimitrov G, Vokotic I, Aidel O, Schaffer RD and Albrand S, 2012 Monitoring of computing resource utilization of the Atlas Experiment, *this conference*

[4] Wiederhold Gio, Mediators in the architecture of future information systems, 1993, *IEEE Computer Magazine*, Vol 25, No3, p38-49 March 1993

[5] Elsing M, Goossens L, Nairz A and Negri G The ATLAS Tier-0: Overview and operational experience 2010 *J. Phys.: Conf. Ser.* **219** 072011 **doi:10.1088/1742-6596/219/7/072011**

[6] Branco M , de Roure David , Lassnig M, Zaluska E, Garonne V. 2010 Managing very large distributed data sets on a data grid *Concurr. Comput.: Pract. Exp.* 22 (2010) 1338-1364

[7] Gallas E J et al. 2012 *this conference*

[8] http://activemq.apache.org/index.html

[9] http://ami.in2p3.fr/opencms/opencms/AMI/www/DatasetStatus.pdf