Constraints on future analysis metadata systems in High Energy Physics

T. J. Khoo[5], A. Reinsvold Hall[10], N. Skidmore[16], S. Alderweireldt[15], J. Anders[13], C. Burr[3], W. Buttinger[9], P. David[11], L. Gouskos[3], L. Gray[4], A. Krasznahorkay[3], P. Laycock[1], A. Lister[14], Z. Marshall[6], A. Meyer[2], T. Novak[2], S. Rappoccio[12], M. Ritter[7], E. Rodrigues[8], J. Rumsevicius[3], L. Sexton-Kennedy[4], N. Smith[4], G. Stewart[3], and S. Wertz[11]

[1]*Brookhaven National Laboratory Physics Department, Upton, NY, US*
[2]*Deutsches Elektronen-Synchrotron DESY, Hamburg, Germany*
[3]*European Organization for Nuclear Research (CERN), Geneva, Switzerland*
[4]*Fermi National Accelerator Laboratory, Batavia, IL, USA*
[5]*Humboldt-Universität zu Berlin, Institut für Physik, 12489 Berlin, DE*
[6]*Physics Division, Lawrence Berkeley National Laboratory, Berkeley, CA, USA*
[7]*Ludwig-Maximilians-Universität, München, DE*
[8]*Oliver Lodge Laboratory, University of Liverpool, Liverpool, UK*
[9]*Rutherford Appleton Laboratory, Didcot, OX11 0DE, UK*
[10]*United States Naval Academy, Annapolis, MD, USA*
[11]*Université Catholique de Louvain, Centre for Cosmology, Particle Physics and Phenomenology (CP3), Louvain-la-Neuve, Walloon Brabant, Belgium*
[12]*University at Buffalo, State University of New York, Amherst, NY, USA*
[13]*University of Bern, Laboratory for High Energy Physics, Bern, CH*
[14]*University of British Columbia, Vancouver, BC, CA*
[15]*The University of Edinburgh, Edinburgh, GB*
[16]*University of Manchester, Schuster Building, Manchester, M13 9PL, UK*

March 2, 2022

Abstract

In High Energy Physics (HEP), analysis metadata comes in many forms – from theoretical cross-sections, to calibration corrections, to details about file processing. Correctly applying metadata is a crucial and often time-consuming step in an analysis, but designing analysis metadata systems has historically received little direct attention. Among other considerations, an ideal metadata tool should be easy to use by new analysers, should scale to large data volumes and diverse processing paradigms, and should enable future analysis reinterpretation. This document, which is the product of community discussions organised by the HEP Software Foundation, categorises types of metadata by scope and format and gives examples of current metadata solutions. Important design considerations for metadata systems, including sociological factors, analysis preservation efforts, and technical factors, are discussed. A list of best practices and technical requirements for future analysis metadata systems is presented. These best practices could guide the development of a future cross-experimental effort for analysis metadata tools.

Keywords: High energy physics, data analysis, metadata, scientific computing, databases

1   Introduction

This document attempts to motivate and codify a set of requirements on systems for storing analysis metadata for future experiments in High Energy Physics (HEP). It arises from a series of meetings held by the Data Analysis Working Group (DAWG) of the HEP Software Foundation (HSF)[1] in early 2021 and the subsequent cross-experiment discussions.

High Energy Physics is a big data endeavour, and significant research efforts have been dedicated to the management of and access to physics data, which is usually interpreted as the combination of detector readout and simulated events. While metadata (i.e. data about data) has long been an important part of detector simulation and reconstruction, in the context of analysis it has received less direct attention. Metadata is a crucial element of data analysis and analysis preservation, but in the software for current experiments, such as those at the Large Hadron Collider (LHC), tools for handling it have less frequently been designed with analysis applications in mind. Instead, analysis metadata systems were either adapted from systems designed for central dataset production or emerged ad-hoc from within the analysis community. In future HEP endeavours, such as the LHC's High Luminosity upgrade (HL-LHC) [1], the lack of a coordinated approach to storing and retrieving analysis metadata may become a limiting factor in the efficiency of analysis. The longer lifetimes of future HEP experiments means that analyses are bigger, more collaborative enterprises that will require more coherent, persistent metadata solutions.

In this document, we will first give examples of different types of metadata and their various scopes. Then we will discuss motivations for the design choices of metadata systems, including current examples. Finally, we will outline technical specifications that should be considered during the design of any future metadata handling system. As the issues motivating these are common to many HEP experiments, these could become the foundation of a common cross-experimental software project, or could help define the specifications of systems developed within the experimental collaborations.

Types of Metadata

For the purposes of this discussion, metadata is taken to refer to any information other than the event data of a simulated or recorded dataset. For example, in the context of an LHC experiment, a recorded bunch crossing is described by event data in the form of tracker hits, calorimeter cell energies and other detector readout, while the corresponding metadata may include the instantaneous luminosity, magnetic field conditions, data quality assessments and so on. These need not be stored in the same location. Frequently, the event data is streamed to `ROOT` [2] files containing tree data structures, with some metadata retained in the file, but other details being relegated to a relational database (henceforth "database"), using identifiers to associate database information to specific files or datasets. Still more relevant information needed to correctly analyse the events may be stored in other formats.

An extensive, but by no means exhaustive, list of metadata examples is given below, together with some mechanisms by which the metadata is stored and accessed. All of these are found to be necessary for carrying out LHC data analysis. In this document, we focus on the information needed to carry out the analysis, rather than on information about analysis team members or other collaboration details. Important as this is for collaboration organisation, it does not impact computing requirements as strongly and does not impact the reproducibility of the results.

1. Dataset provenance – software versions used to generate or process the data, input and affiliated event samples. The full information is typically stored in databases, but some information is available in files or encoded in dataset names.

2. Book-keeping information – cut flow records from filtering samples during processing, as well as initial (possibly weighted) numbers of events generated, which are important for normalising MC generated samples. These values are typically stored in files, potentially as a dedicated data structure or, e.g., as `ROOT` histograms filled during processing.

3. Data quality assessments – flags or lists indicating whether blocks of data are suitable for analyses, at varying levels of granularity, which may also be used for luminosity calculations. Event-level

---

information may be recorded as flags in the data files, while other data formats, e.g. databases or `XML` files, may be used to describe data quality assessments over longer timescales.

4. Calibration data – incredibly diverse, from detailed detector information including alignment constants and magnetic field conditions to calibration corrections for physics object selection efficiencies or four-momenta. Consequently, this type of metadata tends to be stored in a myriad of forms such as databases, text files or code in version-control repositories, `ROOT` files in common `EOS` [3] or `CVMFS` [4] directories, and text or attachments on webpages.

5. A special case of calibration data is that of reweighting information, used to correct distributions across the full dataset, and potentially requiring recalculation for different samples. Common applications include correcting the distribution of the generated pileup multiplicity or adjustments of event kinematics based on control regions or published measurements.

6. Information pertaining to Monte Carlo (MC) datasets – notably event generator input parameters, which are often not published in full detail, and production cross-sections, which in complex signal samples may need to be correlated with the subprocess generated in every event. This information may be stored in a database, but commonly needs to be looked up from tables on `Twiki` pages[2] or other common filesystem storage.

7. An emerging feature is the ability to add user tags to datasets for dataset discovery, organisation of production campaigns, or other purposes. This is currently being done in MC production databases, for example in the ATLAS Metadata Interface (AMI) system[5].

Metadata scopes

The examples above may be loosely classified by the scope of the data to which they pertain:

- Analysis metadata (including examples 4 and 7 above) – describes features of an analysis, such as lists of required datasets and how they are used, versions of calibration metadata used to produce final results, and so on. "Datasets" here refer to samples of events as they are organised in persistent storage, usually according to some useful common criteria, e.g., data recorded during the same run with the same triggers or data simulated with common parameters

- Dataset metadata (includes examples 1, 6, and 7, arguably 4 and 5) – describes either features of datasets, or information about how to analyse datasets.

- Time-dependent metadata (includes examples 3, 4, and 5) – describes information that varies over the course of data collection, typically by being tied to timestamps on the detector data, defining "intervals of validity" (IOVs). In the case of calibration data derived through analysis of simulated or recorded data, IOVs may be as wide as "one specified year" or "one multi-year run", and handled similarly to dataset metadata.

- File-dependent metadata (includes examples 1, 2, and 3) – information about a single file, therefore typically related to the mechanics of file processing. Note that this is not the same as metadata stored in the file, which may in fact be dataset metadata or time-dependent metadata.

2 Motivations

Sociological factors

One of the major challenges when designing any metadata system is encouraging widespread use of the tool(s). There are many examples of useful, well-intentioned tools that failed to be adopted by the community and were eventually replaced. A key aspect that was discussed during the analysis metadata workshops is a good user interface. Analysers prefer a `POSIX`-like command-line/scriptable interface over web access to a database, with the overhead of repeated authentication (e.g. via the

---

[2]For example, the webpage of the LHC Higgs Working Group (https://twiki.cern.ch/twiki/bin/view/LHCPhysics/LHCHWG) or the LHC SUSY Cross-Section Working Group (https://twiki.cern.ch/twiki/bin/view/LHCPhysics/SUSYCrossSections)

`X.509` protocol[3]) being considered one of the disadvantages of the latter. Reading information off `CVMFS` is one popular `POSIX`-like approach that can accommodate unobtrusive authentication procedures. Any new metadata system needs to give careful consideration to how to incentivise analysers to use the system as intended.

Another challenge here is coordinating who is allowed to update the information stored in the metadata system and ensuring that there is sufficient personpower to keep any metadata system up-to-date with the latest recommendations. In some cases, such as information extracted after dataset production (e.g. N(N)LO cross-sections or k-factors), it makes sense to allow for vetted user submissions. In other cases, such as centrally derived corrections to the final physics four-momenta, it makes sense to restrict write-access to specific people or groups. The metadata tools should be flexible enough to handle both situations. If users do not feel they can trust the results in a given system, then that encourages more ad-hoc solutions such as looking up corrections on various `Twiki` pages. Overcoming the concept that "busywork equals validation" may require effective and convenient validation tools and simple APIs to update the metadata as needed. Training is important, especially for new analysers who do not know who to ask.

Analysis preservation

Metadata systems should be designed to satisfy the requirements of analysis preservation, i.e. the capability to repeat an analysis workflow for reproduction or reinterpretation after the analysis has been finalised, a la `RECAST`[4] or `REANA`[5] and other similar projects. Different requirements may apply depending on whether the reproduction is meant to be collaboration-internal, with access to the full software stack, or publicly executable. In either case, the analysis description needs to be recorded in sufficient detail so that it can be reimplemented. In the case that this is addressed via analysis code preservation, then event data and metadata dependencies must be preserved as well, together with access APIs.

For many current analyses, the key metadata sources — e.g., which corrections were applied or even what datasets were used — are only documented in internal notes or `Twiki` pages. This makes it difficult or impossible for future analyzers or theorists to accurately reproduce the results. Capturing all of the analysis metadata associated with a published result is therefore an important goal when designing an analysis system. For analysis preservation, clear versioning is also crucial. But there can be significant complexity involved, as analysis groups often wish to use different versions for individual corrections. In order to validate corrections and calibrations and compare between different sets of conditions, good tools to customise and inspect metadata payloads are a necessity.

There are several promising solutions in existing metadata systems. For example, Belle II has a well-tested infrastructure for analysis-conditions handling that relies on metadata "global tags", in other words single identifiers encapsulating the full metadata configuration [6]. This simplifies the documentation of metadata used by an analysis, because the relevant information is encoded in the global tags. Multiple global tags can be passed to the software framework configuration, and a framework service takes care of presenting this as though it were one global tag. Each analysis group in Belle II can use this to define their own lightweight "analysis global tags" that only need to capture conditions that are not included in the centrally managed reconstruction and simulation global tags. Thus while analysis preservation requires recording the complete set of global tags that were used for an analysis (usually a handful), the analysis global tags are only as complicated as they need to be.

LHCb offline data processing steps up to creation of analyst level datasets are centrally preserved in the `LHCbDIRAC`[6] book-keeping. `LHCbDIRAC` is an extension of the `DIRAC` Grid solution [7] that implements the LHCb data processing workflows, now including the creation of analysis specific datasets containing custom high-level physics variables through the use of Analysis Productions[7]. Analysis Productions are submitted by individual analysts declaratively via `YAML` files by providing the job configuration and input data. To provide assurance that user-prepared configurations are correct, extensive tests are run on the `GitLab`[8] Continuous Integration platform prior to approval. The `LHCbDIRAC` book-

---

[3]https://www.itu.int/rec/T-REC-X.509
[4]https://github.com/recast-hep
[5]https://reanahub.io/
[6]https://lhcb-dirac.readthedocs.io/
[7]LHCb DPA project, https://indico.desy.de/event/28202/contributions/105606/
[8]https://about.gitlab.com/

keeping system preserves metadata such as detector and data-taking conditions used to process the data, versions of applications used and the corresponding options files, enabling high-quality analysis preservation.

Another important consideration is the stability and longevity of the metadata formats throughout and beyond the lives of the experiments. That is, starting with a sound design should permit stability, but the strategy also needs to be adaptable enough to support changes in behaviour and available resources. For example, CMS is currently improving their approach by pushing for a unified `JSON` format for metadata, with files stored in a central `Git` repository[9]. These choices ensure longevity of the payload and flexibility to accommodate diverse needs and take advantage of a widely used versioning system.

Book-keeping

As recorded datasets grow, so does the lead time for investigative operations on these and associated simulated event samples. A recurring challenge in distributed analysis is handling small fractions of job failures due to intermittent technical failures at computing sites. Fully processing the last few percent of events may take several times longer than the bulk, leading to significant lost productivity. For final results, processing 100% of the recorded events is necessary, whereas this criterion can be relaxed for intermediate studies, provided the capability exists to correctly scale results to the full target integrated luminosity.

In analysis of simulated datasets, the main requirement is that the generated number of events (sum of weights in the case of weighted datasets) for the processed files is accessible at the stage when analysis yields are determined. Additional attention may be needed in certain cases, e.g. where procedures involving reweighting by ratios of data and simulated distributions require knowing the full simulated distribution. When sample sizes are limited, variations or biases in the available samples may lead to undesirable effects.

3 Technological considerations

A future analysis metadata system must meet a number of technological criteria to function. The sociological and analysis preservation factors already discussed must be accommodated in its design, as there is a strong precedent of workarounds being established to achieve perceived goals of simplicity, even at the cost of robustness.

Metadata formats

It is likely impossible to use a single format for all analysis metadata, but matching formats to use cases c.f. the aforementioned metadata scopes (dataset, file, and time-dependent) could avoid unnecessary proliferation of formats and requisite infrastructure.

In-file metadata is natural for information that may be needed at the point of job configuration, avoiding the overhead and connectivity requirements of database lookup. It has been pointed out that the boundary between in-file metadata and event data is essentially arbitrary, particularly in systems that have more fluid hierarchical levels than `ROOT`'s `TTree`, permitting better optimisation of metadata content that may span a few files vs. a few events vs. a full dataset.

Similarly, book-keeping of filter fractions and cut flow information from selections applied over multiple data processing steps is probably natural to keep local to the processed data. There may be advantages to allowing easy extension of cut flows by user code, at least if lightweight common libraries are available. Information must be recorded in some format to permit processing of fractional datasets, when the data volumes become large enough to preclude regularly analysing the full datasets prior to the final publication.

Databases are a natural repository for centrally defined dataset-scope information such as dataset provenance and production configurations, as well as user-added information including cross-sections and other contextual labels. Detector calibration metadata tends to be in databases as a standard, and could also accommodate object calibration information, particularly those calibrations that are derived frequently and applied or updated at analysis time. For high-throughput analysis jobs, access

---

[9]https://git-scm.com/

overheads need to be kept down, perhaps via network speed or processing improvements such as a query mechanism with robust caching. This could encourage the use of unified object calibration file formats, and a good user submission and validation interface would keep submission efficient.

There are several potential barriers to widespread database usage. One such barrier may be the need for authentication and the primacy of web-based tools for browsing the database, where analysers prefer `POSIX`-like access with minimal (e.g. `Kerberos`[10]) authentication. This could be overcome by providing simple APIs for payload retrieval, possibly including export of a version of relevant metadata to a local or distributed disk location e.g. `CVMFS`. Effective web interfaces are also needed to support browsing and comparison operations on versions of interest. `Git` hosting sites are one concrete example permitting version diffs, which are widely used.

Another barrier, particularly for smaller collaborations, is the expertise, person power, and resources needed to operate a database at scale. One tested solution (described in more detail below) to this problem is to factorise the database that captures the metadata versions and dependencies from the actual metadata payloads, which can be offloaded to a file system or `Git` or elsewhere. All payloads may not need to be stored in the same system, and in fact, the wide variety of metadata types that need to be accommodated may encourage heterogeneous systems rather than a monolith. A consistent payload location system, however, is desirable. Finally, any metadata database needs to be able to swiftly integrate with the actual scripts, notebooks, and other tools used by analysers. These last two points are discussed in more detail below.

Repository structure

Some form of versioning is strictly necessary. First and foremost, analysers need to be able to specify which version of the metadata is to be used, whether for validation of changes that have been made, or to ensure reproducibility once analysis design has been "frozen" (e.g., post-unblinding). Examples of versioning methods include:

- As mentioned previously, Belle II uses the "global tag" formalism, adopted from conditions databases, adding the capability to merge tags in order to override configurations for specific subsets of the metadata.

- ATLAS and lately CMS uses a set of write-once directories provided on `CVMFS`, holding analysis calibration data. Write protection and timestamps or version numbers as directory paths constitute a free-form versioning solution that minimises constraints on developers.

- ATLAS analysis recommendations are tightly tied to analysis software release tags of the collaboration's common "Athena" software repository on `Gitlab`, wherein the `C++` or `Python` code includes default or hard-coded paths to the calibration data held on `CVMFS`. Overrides of these defaults in analysis configurations may themselves be committed to `Git` together with the analysis frameworks.

- The new `JSON` format for CMS calibration metadata will be self-documenting and include the version number directly in the file.

- For detector and data-taking conditions, LHCb uses `GitCondDB` [8], an experiment-independent conditions database system that leverages `Git` to manage versions and tags. The conditions database uses a 3-dimensional structure dependent on the condition IDs in question (e.g., an `XML` file), the version (a tag or branch name) and the IOV. A bare clone of the conditions repositories is distributed on `CVMFS` making the conditions data available to all jobs, both local and distributed.

Transparency and analysis preservation are likely served well by the capability to define the full metadata versioning information in a single identifier. The metadata identifier should encapsulate the complete description of metadata used for an analysis, but as mentioned previously, the identifier could still point to `Git` commits in a repository, specific files on `CVMFS`, or other locations where the metadata payloads are stored. In-file metadata, in particular metadata related to the details of file processing, are a likely exception to this goal. For flexibility, a scheme by which such identifiers can

---

[10]https://web.mit.edu/kerberos/

be customised and combined — overriding parts of a generic "tag" with more specific requirements — would be required.

There is a balance to be struck between convenience, e.g. allowing the latest version of any/all metadata to be used if not explicitly overridden during R&D periods, and stability, i.e. avoiding silent changes under the feet of analysers. Choosing or even defining new tags for analysis milestones or publication addresses the latter requirement. The point arose that explicit instructions to update settings are considered a way to prevent such unexpected changes, but this effectively equates to busy-work that could be avoided with a more robust validation system. Rather, schemes for transparently associating metadata settings and versions with the resulting analysis datasets should be investigated. These would improve reproducibility and facilitate debugging of unexpected issues as needed.

The full repository should be relocatable. For efficient use of resources, a subset of the metadata should be extractable based on user configuration. This is needed for distributed analysis at sites that may have limited connectivity, such as High Performance Computing and cloud computing nodes, as well as for local analysis on user laptops while in transit or otherwise away from a fast internet connection. Enabling analysis for people who may not have a fast connection to the main processing sites is also an important goal in the interest of equity, diversity, and inclusion. Some existing solutions for relocatability include:

- Runtime download of required files to a local cache via `http`, implemented as an option in the ATLAS "PathResolver" code, which serves as a file search interface with awareness of a variety of sources including `Git` and the `CVMFS` calibration area.

- On Belle II, the payload data are referenced in the relational database as a `URL`, which allows the payload server information to be prepended to the filepath. Much like the ATLAS PathResolver, the client can then specify alternative sources (`CVMFS` is the most common) and a failover strategy (local, `CVMFS`, central server). The central server is the main repository and payloads are copied to `CVMFS` after a short delay, meaning that the majority of read cases are supported by `CVMFS`. It is also possible to specify a local (squid) cache by setting a proxy at a computing site, which then takes care of pulling the information only once from the central server, which is only usually necessary if files are not available on `CVMFS` for some reason (e.g. prompt calibration).

User submissions can enrich metadata and avoid duplication of the work involved in extracting commonly used information such as higher order cross-sections or k-factors. Validation of such submissions is crucial. Consequently, the submission procedure should incorporate steps for checking correctness of the submitted information. A special case of user-added information is labelling of datasets with context beyond the mechanics of their production. This includes association of datasets with specific analyses or applications (background, signal, systematic variation, etc.). For example, LHCb has recently developed such functionality whereby datasets are automatically tagged by properties such as data-taking year and magnet polarity but additionally can be given custom tags to identify them for use in a particular analysis or shared among several. This aids analysis preservation, provides a straightforward and safe way to share analysis datasets, and prevents the need for hard-coded file paths.

While there is a mild risk that such labels could proliferate almost infinitely, there is significant potential to improve analysis workflows by simplifying dataset discovery and identification. These labels could also help to improve data curation operations: automating the obsolescence and deletion of unused data, or notifying analysers automatically when a problem is identified with a dataset. This application motivates interfacing dataset metadata tools with dataset management infrastructure and job management systems. This functionality is now supported in the Rucio [9] dataset management system.

Access interface

A simple API should be provided for requesting and retrieving metadata payloads. This is important for ensuring frictionless access in analysis code. Ideally, access tools should have minimal dependencies and be easily installable, so as to minimise restrictions on analysis framework design and analysis job payload sizes, rather than coupled to heavier collaboration software libraries that may be more cumbersome to install and use on local hardware. The ability to preload only required metadata based

on job configuration may be helpful for optimising metadata access for distributed analysis jobs. A few issues need to be taken into account in considering these targets.

Providing raw metadata payloads through a minimal interface may be straightforward via RESTful APIs [10]. However, depending on how the payloads themselves are structured, it may be more or less convenient to translate them into a form that is useful for analysis. For example, it may be efficient to store calibration information in the form of parameters for fitted functions, which if sufficiently complex may be impractical for users to reimplement. While this need not directly impact the metadata system itself, a choice may need to be made between defining a more substantial adjacent software layer for application of the stored metadata, and choosing a simpler format that may be less precise or efficient to apply.

Trends in analysis software evolution may also have implications for the programming model by which the metadata access tools are provided. In the near term, both `C++` and `Python` are likely to make up a significant share of analysis codes, with the adoption of data science tools and columnar analysis models growing. Consequently, multilingual support may be needed (in the absence of strong collaboration enforcement of language choices). Distribution via package installers or managers may be desirable.

The development of new analysis formats like `NanoAOD` and `DAOD_PHYSLITE` [11, 12] may affect the degree of metadata access needed by the end user. On one hand, routine calibration operations might be applied centrally rather than in user code. On the other, file size reduction may require some operations, notably the application of systematic variations, to be done in memory.

4 Technical specification

Taking into account the motivations discussed in Section 2 and the technological considerations from Section 3, we specify a general set of requirements and desired features for any future analysis metadata system. Here the focus is on features that a future system needs to satisfy, rather than an attempt at prescribing a specific solution. See the previous sections for examples of how current metadata systems satisfy these requirements.

Technical requirements

- Versioning

  - Reproducibility of analysis results demands that metadata identifiers access payloads that are immutable once published. Payloads themselves will need to be periodically updated, in some cases frequently, without invalidating earlier versions. Therefore the payloads will need to be hosted on systems that support a write-once model with version tracking.

  - Versioned payloads of different types will need to be combined in order to serve the needs of a full analysis, for example particle ID or calibration recommendations for a wide range of objects. A mechanism for aggregating payload versions will be required to cleanly communicate these groupings. User friendly interfaces for browsing and comparing single or aggregate identifiers will be needed, with both scriptable and web APIs likely being important.

  - For experimentation and incremental changes, combinations of aggregate identifiers may be needed. Merging and override capabilities for the combination of identifiers will be needed, and the syntax should permit no ambiguity in these operations.

- Relocatability

  - Distribution of the metadata payloads to multiple sites is needed in order to avoid connectivity bottlenecks and barriers. In particular, to ensure that workers without direct access to the hosting servers (e.g. at HPC facilities or for work while in transit), the capability to serve metadata from a local cache is needed. Partial caching of predetermined payloads is important for efficiency.

  - Database lookups must be easy to redirect to a preferred source.

- Lightweight API

- Diverse end-stage analysis code (e.g. `Python` or `C++` analysis frameworks) must be able to access the metadata, so access APIs should be lightweight with minimal dependencies. Programmatic access, e.g. in `Python` scripts, must be supported as a first-class use case.
- Protections should be implemented such that API usage cannot overload a database, even if a high volume of requests are made from batch or grid jobs. Both payload storage and identification systems need to have high availability and robustness, as well as aggressive caching of requests.

- Extensible system

  - User submission of metadata must be allowed, to support various types of additions. The main use case will be for updates of metadata derived externally from central sample production systems, e.g. calibrations, cross-sections etc. A special case is the extension of cutflow information, which may be stored as in-file metadata.
  - Robust access control systems will be needed to restrict additions to vetted submitters. Validation systems are also necessary, to ensure submissions can be tested stringently before being made accessible.

- Unobtrusive authentication

  - Access to the metadata content should be granted based on persistent authentication methods, rather than burdening users with repeated sign-in steps. This is particularly important for programmatic access to metadata.

- Intervals of Validity

  - The metadata identification system must have the ability to store information about relevant IOVs, particularly for partial-dataset calibrations.
  - This feature might also be usable for additional contextual configuration, such as identifying MC vs. recorded data.

Desirable features and other considerations

- Complete analysis description

  - If all metadata descriptions can be captured in a single system, then it is natural to extend aggregation of metadata tags to provide a full description of software and inputs needed for a single analysis.
  - Recording software versions, job configuration parameters, input datasets and auxiliary data fully serves multiple needs, including the tracking, preservation, or combination of analyses.
  - While all analysis metadata identifiers would need to be encoded in the system, payloads themselves could be stored in any repository, although long-term analysis preservation requires guarantees on the longevity of all relevant storage systems.

- User-applied dataset tags

  - Free-form labels attached to event datasets can be used to serve various purposes, including identification and categorisation of analysis inputs, management of storage system capacity, and invalidation of obsolete or incorrect data. Allowing users to attach arbitrary labels and share those labels would allow substantial flexibility.
  - Directly interfacing the dataset labelling system to dataset management and job submission systems would generate additional hooks for efficient analysis management.

- Interpretation of metadata content

  - Additional tooling, separate from the metadata access systems, may be useful for payload interpretation. For example, it may be efficient to store calibration corrections in the form of fit constants, or even as neural network parameters. Centrally provided tools would then be a more reliable solution for consistent and correct translation of the payloads to the final corrections.

5 Summary

Based on the common experiences of analysers of LHC and contemporary particle physics experiments, this document has categorised types of analysis metadata in terms of scope and content, as well as identifying not only current technological solutions but also major challenges in the storage and access of analysis metadata. A list of technical requirements for future analysis metadata systems addressing the needs of analysis at larger scales has been compiled, accounting for the diverse needs for metadata access in evolving analysis ecosystems, including practical and sociological concerns for individual analysers, sharing of information within experimental collaborations and the long-term preservation of analyses for reuse and reinterpretation. The list presented does not define particular solutions to the problems posed, but is rather intended to guide future R&D on the concrete implementation of such systems.

6 Acknowledgements

7 Declarations

Data availability

Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

Competing Interests

On behalf of all authors, the corresponding authors state that there is no conflict of interest.

References

[1] G. APOLLINARI, O. BRÜNING, O., T. NAKAMOTO, L. ROSSI: High Luminosity Large Hadron Collider HL-LHC. In: *CERN Yellow Rep.* (2015), Nr. 5, S. 1–19. http://dx.doi.org/10.5170/CERN-2015-005.1. – DOI 10.5170/CERN–2015–005.1

[2] R. BRUN AND F. RADEMAKERS: ROOT - An Object Oriented Data Analysis Framework. In: *Nucl. Inst. & Meth.* A389 (1997), S. 81–86. http://dx.doi.org/10.5281/zenodo.3895860. – DOI 10.5281/zenodo.3895860

[3] PETERS, AJ. ; SINDRILARU, EA. ; ADDE, G.: EOS as the present and future solution for data storage at CERN. In: *J. Phys. Conf. Ser.* 664 (2015), Nr. 4, S. 042042. http://dx.doi.org/10.1088/1742-6596/664/4/042042. – DOI 10.1088/1742–6596/664/4/042042

[4] BLOMER, J. ; BUNCIC, P. ; CHARALAMPIDIS, I. ; HARUTYUNYAN, A. ; LARSEN, D. ; MEUSEL, R.: Status and future perspectives of CernVM-FS. In: *J. Phys. Conf. Ser.* 396 (2012), S. 052013. http://dx.doi.org/10.1088/1742-6596/396/5/052013. – DOI 10.1088/1742–6596/396/5/052013

[5] F. LAMBERT, J. ODIER, AND J. FULACHIER: Broadcasting dynamic metadata content to external web pages using AMI (ATLAS Metadata Interface) embeddable components. (2018), Nov. http://dx.doi.org/10.1051/epjconf/201921404004. – DOI 10.1051/epjconf/201921404004

[6] T. KUHR, C. PULVERMACHER, M. RITTER, T HAUTH AND N. BRAUN: The Belle II Core Software. In: *Comput. Softw. Big Sci.* 3 (2019), Nr. 1, S. 1. http://dx.doi.org/10.1007/s41781-018-0017-9. – DOI 10.1007/s41781–018–0017–9

[7]   F. Stagni, A. Tsaregorodtsev, P. Charpentier et al.: DIRACGrid/DIRAC: v6r20p15. (2018), Oct. http://dx.doi.org/10.5281/zenodo.1451647. – DOI 10.5281/zenodo.1451647

[8]   M. Clemencic and LHCb Collaboration: GitCondDB. (2020), Jun. http://dx.doi.org/10.5281/zenodo.5724490. – DOI 10.5281/zenodo.5724490

[9]   Barisits, Martin u. a.: Rucio - Scientific data management. In: *Comput. Softw. Big Sci.* 3 (2019), Nr. 1, S. 11. http://dx.doi.org/10.1007/s41781-019-0026-3. – DOI 10.1007/s41781–019–0026–3

[10] Fielding, Roy T.:   Architectural Styles and the Design of Network-based Software Architectures. In: *Doctoral dissertation, University of California, Irvine* (2000). https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[11] A. Rizzi, G. Petrucciani, M. Peruzzi: A further reduction in CMS event data for analysis: the NANOAOD format. In: *EPJ Web Conf.* 214 (2019). http://dx.doi.org/10.1051/epjconf/201921406021. – DOI 10.1051/epjconf/201921406021

[12] J. Elmsheuser and others: Evolution of the ATLAS analysis model for Run-3 and prospects for HL-LHC. In: *EPJ Web Conf.* 245 (2020), S. 06014. http://dx.doi.org/10.1051/epjconf/202024506014. – DOI 10.1051/epjconf/202024506014