Evolution of the ATLAS CREST conditions DB project

E.Alexandrov[a], A.Formica[b], M.Mineev[a,1], N.Ozturk[c], S.Roe[d], V.Tsulaia[e], M.Vogel[c]

[a] Joint Institute for Nuclear Research, Joliot-Curie 6, 141980 Dubna, Russia

[b] Université Paris-Saclay, CEA/Saclay IRFU, 91191 Gif-sur-Yvette, IRFU/CEA, France

[c] University of Texas at Arlington, 01 South Nedderman Drive, Arlington, TX 76019, USA

[d] CERN, CH - 1211 Geneva 23, Switzerland

[e] Lawrence Berkeley National Laboratory, 1 Cyclotron Rd, Berkeley, CA 94720, USA

The CREST project is a new realization of the conditions DB with the REST API and JSON support for the ATLAS experiment at the LHC. This project simplifies the conditions data structure and optimizes data access. CREST development requires not only the client C++ library (CrestApi) but also the various tools for testing software and validating data. A command line client enables a quick access to the stored data. A set of the utilities was used to make a dump of the data from CREST to the file system and to test the client library and the CREST server using dummy data. Now CREST software is being tested using the real conditions data converted with the COOL to CREST converter. The Athena code (ATLAS event processing software framework) was modified to operate with the new conditions data source.

INTRODUCTION

This article is focused on the CREST evolution aspects and considers new features in the CREST project [1]. This project is a new database for **C**onditions data with **REST** interface (**CREST**) and JSON support for the ATLAS detector [2] at the Large Hadron Collider (LHC) at CERN. The CREST prototype was described in detail in [3]. Conditions data are non-event data, such as detector calibration and alignment data, electrical and environmental measurements such as voltages, currents, pressures, temperatures, information about the run and data acquisition configuration, LHC beam information, trigger configuration, detector status data, used to describe the detector status, and constitute an essential ingredient for the processing of physics data, in order to reconstruct events optimally and exploit the full potential of the detector. The project was started for several reasons. The COOL and CORAL packages are used today for the ATLAS conditions data [4]. The long term maintenance and evolution for the COOL API and CORAL software were a concern. Some workflows have issues with poor caching efficiency of the retrieved conditions data. COOL lacks built-in support for managing global tags, which are important for labeling and organizing data. It is important to note that this article does not cover all aspects of CREST development. CREST DB infrastructure was also discussed in Ref [5].

ATL-SOFT-PROC-2023-041
21 September 2023

[1] E-mail: mineev@jinr.ru

CREST DATA MODEL

The CREST data model consists of five tables which contain metadata and payload data. Conditions data are stored in the payload table. Values are consumed as an aggregated set (typically a header and some parameter container(s)). Conditions metadata are organized in three tables: the tables for IOVs (Intervals Of Validity), tags, global tags (plus one table used essentially for mapping between tags and global tags). IOV has a time parameter - the start time (time can be represented as a timestamp or a run number, etc.). Each IOV also has a reference to the payload in the form of an sha256 hash. The tag is a label to group a set of IOVs. The CREST tags correspond to the COOL tags/folders. Several tags can be grouped together with a global tag. A tag can be associated to many global tags.

CREST SERVER

The data on the CREST server is accessed using HTTP requests via a REST API. The request and response bodies are formatted in JSON. The CREST server is based on standard Java technologies (JEE, Spring) and specifications (JAX-RS, JPA). It can be deployed in the same Tomcat server as Frontier or as a standalone service (using standard Java web servers like undertow, jetty, …). The CREST server is compatible with multiple DB technologies (e.g. Oracle, PostgreSQL). During the past year, the CREST server API was revised. The main changes are: the duplicated endpoints were removed and the request syntax was simplified. The authorization in CREST will be based on OAuth2 technology [6]. A developer's version of the server with the OAuth2 support was deployed, using CERN SSO for authentication and authorization.

CREST C++ CLIENT LIBRARY (CrestApi)

CrestApi is a C++ client library for the CREST server (the data can also be stored on the local file storage). It stores, reads (and updates) the data on the CREST server. CrestApi uses the NLohmann JSON library to operate with the data in JSON format and the CURL library to send the HTTP requests. Some new methods were added, for example the methods to create IOVs, to remove a tag from the global tag etc. OAuth2 authentication support was added in the CrestApi developer's version (currently only a prototype, with limited capabilities).

CREST COMMAND LINE CLIENT (crestCmd) AND UTILITIES

The CREST command line client (crestCmd) is a tool for a quick access to the data on the CREST server, written to simplify the development of the other CREST project components. Some new commands were added here, for example to get the size of the IOV list for a selected tag, as well as commands to get the version number of the CREST server and CREST client (CrestApi version) etc. For the CREST server validation, it is necessary to have some additional tools for quick server testing. The tag creation utility is used to create "dummy" test data: a tag, a tag meta info (this object contains the COOL channel list), IOVs with payloads. The payloads are generated randomly and the payload size can be set by the user. The crestExport utility can make a data dump which corresponds to a selected tag (the tag, the tag meta info, the IOVs with

payloads) from the CREST server to the local file storage. The crestImport utility reads the tag data from the local file storage and writes them to the CREST server. The crestCopy utility creates a tag copy on the CREST server. The tag removing command was added in crestCmd, but sometimes it is necessary to remove a global tag with all its tags. The removeTagList utility can do this. All these utilities and crestCmd are included in the crest_cmd package and are available on the CREST git repository.

COOL TO CREST CONVERTER

To simplify the task of reading data from COOL, it was decided to use the existing IDatabase interface of the COOL package. The implementation of this interface for CREST made it possible to use the AtlCoolCopy utility, specifying CREST as the output base. AtlCoolCopy is part of the standalone version of the ATLAS software. The source data can be filtered by folder, tag and IOV range. The COOLR API (a REST service dedicated to retrieval of COOL data via HTTP) is used to convert the data associated with the selected global tag. This utility can be used via a python client. One of its features is to get information in JSON format about all folders associated with a given global tag. The general scheme for converting all data associated with the selected global tag is shown in Figure 1.

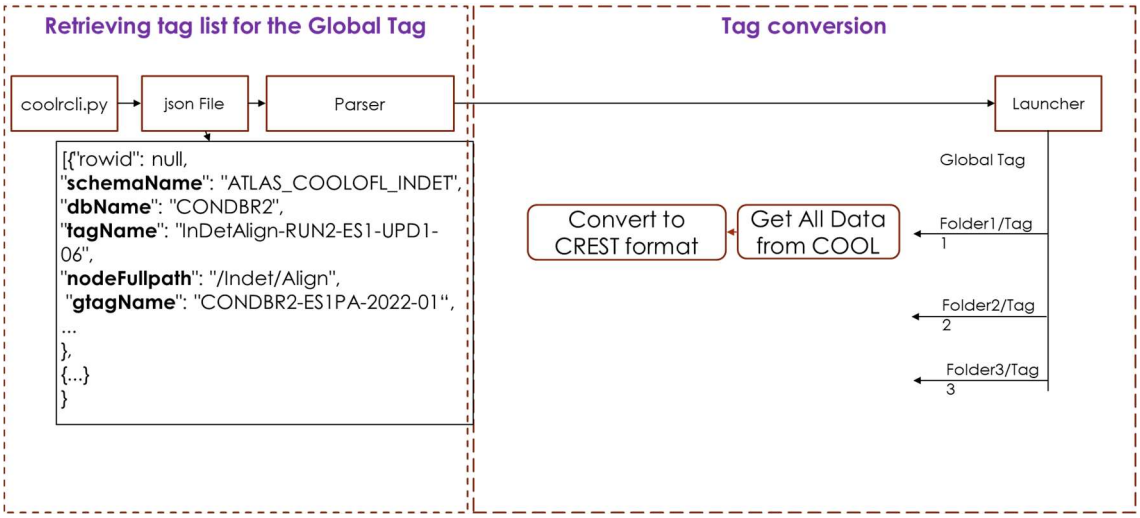

Fig. 1: Schema for converting all data from a global tag.

Using COOLR, we get data about the global tag in JSON format. The JSON contains a list of directories with additional parameters, such as local tags, the name of the database in which this information is stored, and so on. This information is required by AtlCoolCopy to read data from COOL. The JSON data is processed by a python script (launcher), then the conversion of all folders from the resulting list is started sequentially. Data about the global tag associated with the current tag is passed via AtlCoolCopy to the CREST plugin, which sends it to the CREST server, along with other metadata. The conversion will end after all folders from the list have been converted.

CREST AND ATHENA

One of the main tasks for the CREST project is to modify the ATLAS event processing software framework (Athena) [7] to make it work with the CREST data. The software adaptation began with the algorithms (defined in JobOption files, which are Athena tasks written in python) for the LAr and Tile calorimeters. The COOL to CREST converter is used for data migration for the ATLAS subsystems. The main changes were done in the IOVDbSvc package. Athena users need only small changes in their code to use CREST data (see Figure 2).

CREST options in Python:

```
...
acc.getService("IOVDbSvc").GlobalTag="CREST-RUN12-SDR-25-MC"                # set global tag
acc.getService("IOVDbSvc").Source="CREST"                                   # use CREST data
acc.getService("IOVDbSvc").crestServer="http://crest-01.web.cern.ch:8080/api"  # Server URL
acc.getService("IOVDbSvc").CrestToFile = True                               # CREST data dump
...
```

Fig. 2. The IOVDbSvc parameters in the JobOption.

It is necessary to specify the data source (with the Source parameter) and to set a global tag name (with GlobalTag). There are additional parameters. It is possible to change the CREST server URL with a port number and to make a CREST data dump on the file storage. The CREST data are dumped in the local file storage in the same directory where the JobOption was started. The CrestApi library is used to make this dump.

CONCLUSION

The CREST project has evolved with its components and the related Athena software in the past year. The CREST server API was optimized. The COOL to CREST data converter was rewritten to operate with the global tags. The new commands were added in the command line client for CREST software testing. The conditions data in the CREST format were successfully used with the offline data processing algorithms from the LAr and Tile subsystems. While testing and validation continue with other subsystems CREST shows already a well-developed system to be fully deployed for the Run 4 of the ATLAS experiment at the LHC.

REFERENCES

1. *L.Rinaldi et al*., Conditions evolution of an experiment in mid-life, without the crisis (in ATLAS) // EPJ Web Conf., Volume 214, 04052, 2019.
2. *ATLAS Collaboration*, JINST 3, S08003 (2008), https://jinst.sissa.it/LHC/ATLAS/chtt.pdf (accessed 19.09.2023)
3. *A.Formica et al.,* The Development of a New Conditions Database Prototype for ATLAS Run 3 within the CREST Project // CEUR Workshop Proceedings, ISSN:1613-0073, Vol. 3041, p. 86-90, 2021.

4. *R.Trentadue et al.,* LCG Persistency Framework (CORAL,COOL, POOL): status and outlook in 2012 // J. Phys. Conf. Ser. 396 053067, 2012.
5. *D.Costanzo et al*., Towards a new conditions data infrastructure // to appear in the proceedings of CHEP 2023, Norfolk, Virginia USA, May 8-12, 2023, to be published in EPJC Web of Conferences, ATL-SOFT-PROC-2023-024 https://cds.cern.ch/record/2870121 (accessed 14.09.2023).
6. OAuth 2.0 // Available at: https://oauth.net/2/ (accessed 10.08.2023).
7. ATLAS Collaboration. (2019). Athena (22.0.1). Zenodo. https://doi.org/10.5281/zenodo.2641997 (accessed 19.09.2023)