EPJ Web of Conferences **214**, 02042 (2019) https://doi.org/10.1051/epjconf/201921402042
CHEP 2018

Conditions and Alignment Extensions of the DD4hep Detector Description Toolkit

Markus Frank[1,*], *Frank* Gaede[2,**], *Marko* Petric[1,***], and *Andre* Sailer[1,****]

[1]CERN, 1211 Geneva 23, Switzerland
[2]Desy, 22607 Hamburg, Germany

Abstract. The detector description is an essential component to analyze data resulting from particle collisions in high energy physics experiments. The interpretation of data from particle collisions typically requires auxiliary data which describe in detail the state of the experiment. These accompanying data include alignment parameters, parameters describing the electronics as well as calibration- and environmental constants. We present a mechanism to manage such data in multiple simultaneous versions depending on their validity. The detector conditions data are made available to the physics algorithms through a number of transient objects grouped to collections. Such a collection represents a coherent slice of all conditions data necessary to process one or several events depending on the valid interval of the slice being the intersection of the individual conditions.
A multi-threaded application may hold several such collections in parallel depending on the time-stamps of the events currently processed. Once prepared, these collections are read-only and can easily be shared between threads with minimal requirements for locking and hence minimal overhead. We deliberately restrained ourselves from providing a persistent data solution, which in the past were fields of expertise of the experiments, but rather provided the necessary hooks to populate the conditions cache. We will present the use-cases that have driven the development, the main design choices and details of the implementation.

1 Introduction

The development of a coherent set of software tools for the description of high energy physics detectors from a single source of information has been on the agenda of many experiments for decades. In this context it is indispensable for data processing applications, which analyze the detector response of particle collisions with high precision to not only have access to basic detector information such as the geometry, but also to other time dependent data. These data not only include environmental measurements such as temperatures and pressures used to recalibrate e.g. the response of gaseous sub-detectors, but also derived quantities, which may be the result of lengthy computational work such as alignment constants to model geometrical imperfections. These so called conditions data items or short conditions, change slowly

*e-mail: Markus.Frank@cern.ch
**e-mail: Frank.Gaede@desy.de
***e-mail: Marko.Petric@cern.ch
****e-mail: Andre.Sailer@cern.ch

(> O(minutes)) compared to the frequency of particle bunch collisions, which at the Large Hadron Collider (LHC) are delivered to experiments with a frequency of up to 40MHz [1]. In general each condition is valid for a given period of time, the *Interval of Validity* (IOV); in practice many or at least several items have the same interval of validity. Conditions data are an essential ingredient to understand the particle collisions and data processing frameworks must provide means to easily access these quantities given the time-stamp of a particle collision. The access to the conditions in a coherent manner within the context of the DD4hep Detector Description toolkit [2] was realized in the extension package DDCond. The design is strongly driven by ease of use; developers of detector descriptions and applications using them should have to provide only minimal information and minimal specific code to achieve the desired result. DDCond was specifically designed to support modern CPU architectures favoring multi threaded applications. Hence, special attention was put to achieve a design, which minimizes locking requirements by construction.
The organization of this paper is the following. We briefly recapitulate the general structure of the DD4hep geometry description toolkit in Section 2. Then, the guiding requirements for DDCond and the architectural design is discussed in Section 3. Finally, we discuss in Section 4 a special type of conditions, the alignments.

2 The DD4hep Detector Description Toolkit

The design of the DD4hep toolkit [2] is shaped by the experience of detector description systems, which were implemented for the LHC experiments, in particular the LHCb experiment [3, 4], as well as the lessons learnt from other implementations of geometry description tools developed for the Linear Collider community [5, 6]. DD4hep aims to widely reuse existing software components, in particular the ROOT geometry package [7] TGeo, part of the ROOT project [8], a tool for building, browsing, navigating and visualizing detector geometries. The second component is the Geant4 simulation toolkit [9], which is used to simulate the detector response from particle collisions in complex designs [10].

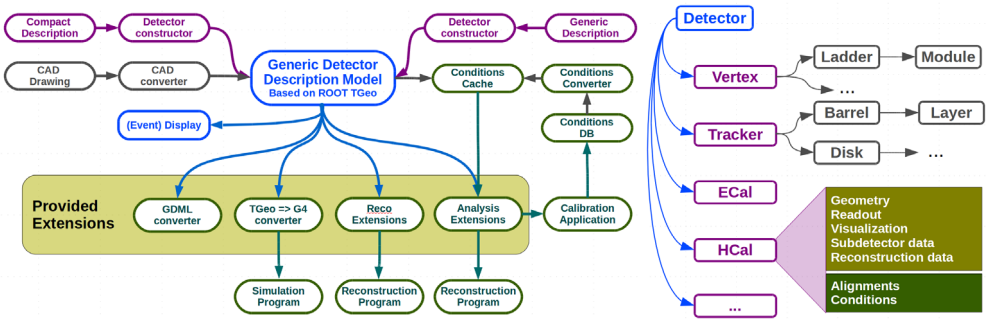

Figure 1. The components of the DD4hep detector description toolkit. To the left the mechanisms to populate the hierarchical structure of the detector giving access to the geometry and other quantities as shown to the right. This in-memory model of a generic detector description base on ROOT TGeo is the base representation to support all data processing applications.

DD4hep implements a mechanism to create a memory based detector description from adaptable input sources as shown in Figure 1. The code fragments interpreting the input data instantiate a model of the detector defined by a set of C++ classes. This in-memory model of an experiment is represented as a tree-like hierarchy of detector elements which allows access to all detector quantities. The model is discussed in detail elsewhere [2].

3 Concepts and Toolkit Design

3.1 Toolkit Requirements

The following list of requirements describes the necessary functionality of the toolkit implementing the access to conditions data within the DD4hep framework:

- A **minimalistic approach** ensures to keep the interaction of users with the framework at a minimum. Users must supply DDCond with knowledge to access raw conditions data using an abstract interface to a experiment defined persistent backend. Secondly, users must supply transformation callbacks to perform the computation to obtain derived quantities.
- **Ease of use**: Condition data are typically used by algorithms within the context of a subdetector or a part thereof, defined in DD4hep by a `DetElement`. Corresponding conditions are obtained from the cache using this `DetElement` together with a mnemonic identifier.
- **Customization** is necessary to replace components that satisfy not foreseen requirements.
- **Computational units** provide an appropriate mechanism to obtain derived conditions. User friendliness to develop and apply such extension components is mandatory.
- **Compatibility** ensures to at least technically satisfy the former requirement where each condition may change individually at any time.
- **Performance and optimal CPU utilization** of event data processing applications is indispensable in view of the planned upgrade programs of the LHC experiments. If performance and flexibility aspects lead to contradictory designs, performance is favored.
- **Multi-threading support** must be intrinsic and overheads to avoid data races must be minimal. Locking mechanism must be simple and comprehensive to framework developers.
- **Parallel processing with different conditions data** to process non time-sequential events as it occurs e.g. during run-changes must be seamlessly supported.

3.2 Design Choices

Condition change slowly, but contrary to the assumption stated at the startup of the LHC experiments, where any conditions item was allowed to change at any moment, in reality for most experiments nowadays many conditions are valid for a relatively long time interval O(one year) such as survey data. Other large fractions of conditions are valid for time-spans of O(one hour) as envisaged by LHCb, where conditions supposedly change every run. Conditions may also change more frequently O(20 seconds) corresponding for example to the duration of a luminosity section in CMS [11]. These time intervals in the following are called *Interval-Of-Validity* (IOV). For these empiric reasons we postulate for DDCond that conditions data change in batches at certain time intervals and not individually. It is understood that such an approach requires some effort, discipline and compromises within the subdetector experts of an experiment.
Experience from the LHC experiments has shown that this goal can be achieved and the approach is feasible. While this postulate does not strictly inhibit conditions changing randomly in time, it allows for designs leading to efficient solutions as described in Section 3.3. Large benefits evolve from a strategy to prudently choose the IOV of conditions. In view of today's budgetary situation for computing and the needs of the LHC experiments for the upgrade scenarios it is favorable to opt for fast and performant solutions. The fact that several LHC experiments currently review the decisions taken nearly two decades ago underlines this [12].
To allow for better optimization and later functional refinement, DDCond takes advantage of the DD4hep factory mechanism to instantiate components optimized for not foreseen functionality.

3.3 Toolkit Implementation

During the design of the project three different domains crystallized, which are largely independent:

- **The conditions data items** are the main entities of interest to clients. They contain the actual numeric constants both of raw or derived origin.
- **The conditions cache** manages the ensemble of conditions in memory. The cache - though it is an important ingredient - is not presented to the toolkit client. A door-keeper service, called the `ConditionsManager`, encapsulates all client interaction. The cache allows to build sub-ensembles of conditions according to the time stamp of an event.
- **The Conditions slices**, which contain all conditions of a given type projected for an event time-stamp. The ensemble is valid for a time-span being the intersection of the IOVs of the conditions contained. These are defined by the *conditions content*. A configured slice may be used by any number of threads simultaneously.

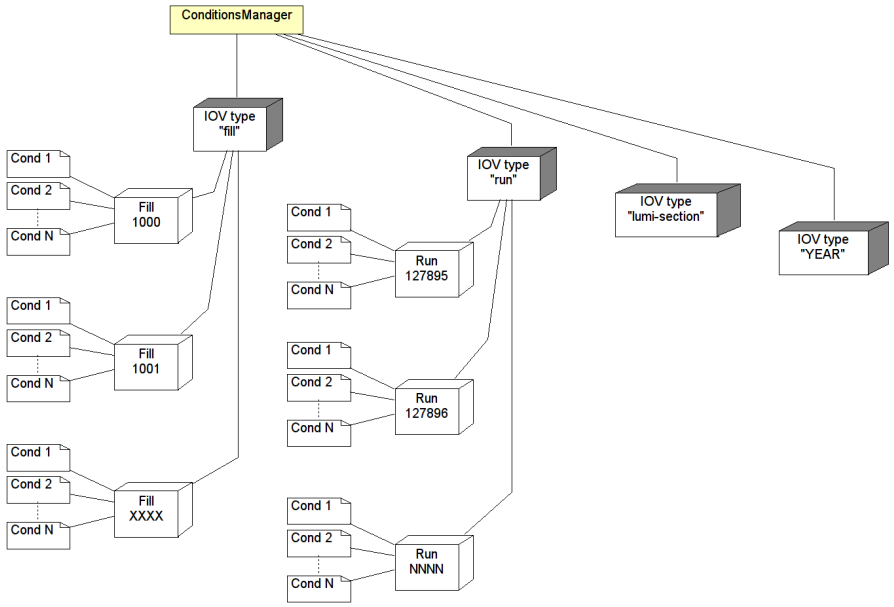

Figure 2. An object diagram of the conditions data cache. Conditions are firstly partitioned by independent types with associated pools populated by conditions with the same IOV.

A condition is uniquely identified by the *conditons-type* such as runs, fills, or luminosity sections, the IOV and by a 64 bit integer: the higher 32 bits identify the unique key of the detector element the condition belongs to, the lower 32 bits identify the data item. This definition allows for a fast conditions lookup and can support range scans of conditions which belong to the same detector element. The condition gives access to a dynamically bound opaque payload containing the actual data. An IOV is only loosely bound to time: it may correspond to a time-interval, an interval of runs, fills, etc. Derived conditions fit the model: they share the same conditions type and the resulting IOV is the intersection of the IOVs of the input data. Dependencies between conditions of different types would result in significant overheads normalizing the different IOV types and hence are not allowed.
To accommodate large systems the conditions cache is partitioned to pools identified by a *conditons-type*. The partitions are populated by sub-pools containing conditions sharing

the same IOV as shown in Figure 2. Such a partitioned system is useful in setups where large fractions of conditions have largely different IOVs i.e. largely different lifetime. This approach supports both, backwards compatibility with system where conditions data change randomly with time and after well defined intervals. Once condition items are added to the cache, they are considered read-only and may not be altered. The overall size of the conditions cache is only limited by the available memory resources and the sophistication of the hashing algorithm of the conditions keys. The two concepts lead to nearly intrinsic cache coherence for multi-threaded applications and several performance advantages:

- The projection of a conditions slice is based on the IOV of the sub-pools. Since a slice may contain a superset of conditions, set differences are computed efficiently without nested iterations to lookup individual conditions. Sub-pools are attached slices using shared pointers, which allows to use the sub-pool and its content consistently even if removed from the cache.
- Projections to be created benefit from existing sub-pools. Only conditions which do not satisfy the requested event time need to be loaded.
- Cache cleanup strategies are only based on sub-pools. They are simple and performing.
- Cache locking is only required when adding or removing sub-pools. Threads processing events using a configured slice during such operations may continue to access this projected set without protective locks, an important measure to improve performance.

In Section 3.4 the interaction of a framework client to obtain a valid *ConditionsSlice* for a given IOV is elaborated in detail. Otherwise the system is entirely opaque.

3.4 Client Interaction with the Conditions Cache

Ease of use quickly conflicts with flexibility, both requirements stated in section 3.1: more flexibility requires more tuning parameters to be presented to the client and hence increasing complexity rather than ease of use. In DDCond this working point is defined as follows:

- The client must define the `ConditionsContent` containing a set of persistent addresses which describe how to find a condition in a database. The transformation recipes to obtain derived values must be supplied as instances of the base-class `ConditionUpdateCall`. Such a `ConditionsContent`, which may be reused, must be associated to each `ConditionsSlice`[1].
- During the preparation step all conditions defined in the `ConditionsContent` matching the specified time stamp are attached to the `ConditionsSlice` and the derived conditions evaluated. The slice is now a read-only entity with an IOV being the intersection of the contained IOVs and ready to be used for all matching events simultaneously for any number of threads.
- A cleanup procedure must be applied regularly to eliminate unused condition sub-pools from the cache and limit resource usage. A default cleanup object may be installed, but clients are encouraged to call this action explicitly and control the cache with fine granularity.

Specialized utility classes may easily be implemented, which ease the burden on developers by grouping and hiding some of these component interactions specifically for certain experiments. Based on user feedback, such conveniences may become part of DDCond in the

[1]Please note: DDCond does not provide a database solution, but rather uses an abstract interface to load conditions based on these address descriptors. Historically, for performance reasons, such database implementation were the domain of the experiments.

future. All concrete implementations mentioned above, the `ConditionsManager`, the conditions sub-pools and the conditions container embedded in the slice are instances created by factory objects. This ensures flexibility necessary to satisfy special needs or to exploit certain performance aspects.

4 Alignment Support

The `Alignments` describe geometrical imperfections using offsets between the local and global volume coordinate systems with respect to the ideal geometry, the so called *alignment delta*s. The *deltas* contain a translation and a rotation around an (optional) pivot point. This quantity is transformed to derived alignment objects containing the transformation matrix to the global frame (typically called "world") as used e.g. for track reconstruction. In DDCond they are handled as derived conditions.

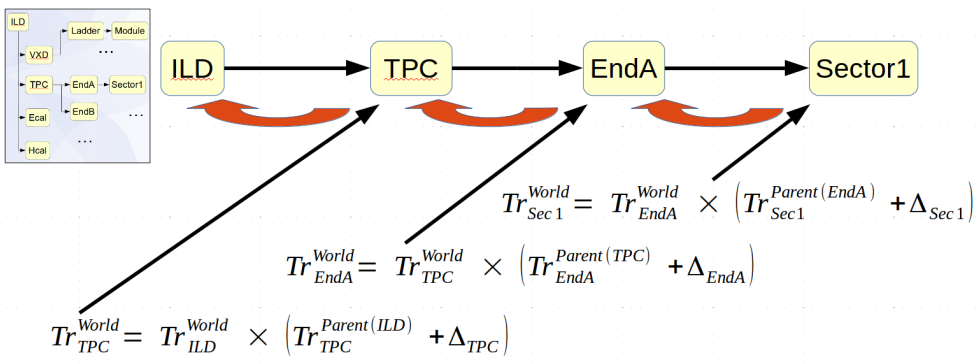

Figure 3. An example of the computation of the alignment matrices to the global coordinate frame taking into account all intermediate misalignments. The computation starts at the top and each subsequent hierarchical layer of detector elements takes into account the computational results of the parent layer.

To perform this computational step efficiently by reusing intermediate results as shown in Figure 3 it is advantageous to compute all the derived alignment transformations at once. This functionality is provided by the DD4hep core through the `AlignmentsCalculator` component taking as input a map of alignment deltas indexed by the detector elements giving back a similar map containing the fully computed alignment objects including the transformations for all nodes where a parent element got shifted.

5 Conclusions

The extensions to support access to conditions data and alignment data complete the DD4hep toolkit ensemble based on the DD4hep core [2]. The chosen approach requires very few user actions to provide data processing applications with conditions data and provides means to compute alignment constants. The chosen approach, where all key components may be easily specialized by clients injecting other components using the factory mechanism should provide all the flexibility needed to not only satisfy the use cases used to develop DDCond, but also to handle more sophisticated and specific use cases.

6 Acknowledgements

This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 654168.

References

[1] L. Evans, P. Bryant (editors), LHC Machine, 2008 JINST 3 S08001

[2] M. Frank et al., DD4hep: A Detector Description Toolkit for high energy physics Experiments, International Conference on Computing in High Energy and Nuclear Physics (CHEP 2013), Amsterdam, NL, 2013.

[3] LHCb Collaboration, LHCb, the Large Hadron Collider beauty experiment, reoptimised detector design and performance, CERN/LHCC 2003-030

[4] S. Ponce et al., Detector Description Framework in LHCb, International Conference on Computing in High Energy and Nuclear Physics (CHEP 2003), La Jolla, CA, 2003, proceedings.

[5] The ILD Concept Group, The International Large Detector: Letter of Intent, ISBN 978-3-935702-42-3, 2009.

[6] H. Aihara, P. Burrows, M. Oreglia (Editors), SiD Letter of Intent, arXiv:0911.0006, 2009.

[7] R. Brun, A. Gheata, M. Gheata, The ROOT geometry package, Nuclear Instruments and Methods **A** 502 (2003) 676-680.

[8] R. Brun et al., ROOT - An object oriented data analysis framework, Nuclear Instruments and Methods **A** 389 (1997) 81-86.

[9] S. Agostinelli et al., "Geant4 - A Simulation Toolkit", Nuclear Instruments and Methods **A** 506 (2003) 250-303.

[10] M. Frank et al., DDG4, A Simulation Framework based on the DD4hep Detector Description Toolkit. International Conference on Computing in High Energy and Nuclear Physics (CHEP 2015), Okinawa, Japan, 2015, proceedings.

[11] M. Borisyak et al., Towards automation of data quality system for CERN CMS experiment, International Conference on Computing in High Energy and Nuclear Physics (CHEP2016) San Francisco, USA, 2017 J. Phys.: Conf. Ser.898 092041

[12] B. Couturier et al., Perspectives for the migration of the LHCb geometry to the DD4hep toolkit, poster No. 2937633, International Conference on Computing in High Energy and Nuclear Physics (CHEP 2018), Sofia, Bulgaria, proceedings.