Storing EPICS process variables in HDF5 files for ITER

R. Castro[1], Y. Makushok[2], L. Abadie[3], B. Bauvir[3], A. Neto[4], J. Vega[1]

1. Laboratorio Nacional de Fusión, CIEMAT. Madrid. Spain
2. MINSAIT - INDRA, Madrid, Spain
3. ITER Organization, St Paul lez Durance, France
4. Fusion for Energy, Barcelona, Spain

EPICS (Experimental Physics and Industrial Control System) is the main technology used by ITER for distributed control of all its systems. EPICS uses an architecture based on a distributed protocol that allows the exchange of control data (process variables) between different elements and subsystems. In its previous versions, EPICS used "Channel Access" as the communication protocol, and its control variables had a simple structure that allowed the exchange of a very limited set of primitive data types. From its version 7, EPICS uses a new "pvAccess" protocol, and its process variables allow for nested data types that can form very complex data structures which can contain thousands of fields. From the perspective of data storage, these pvAccess variables and their nested data types include important challenges both at a functional level and at a performance level.

In this paper, the EPICS storage system that has been implemented for ITER is presented. This system stores the EPICS process variables in standard HDF5 files that ensure not only the accessibility and maintainability of the data, but also its full compatibility with the rest of the ITER data storage system.

Keywords: Data archiving, HDF5, EPICS, pvAccess, Channel Access

1. Introduction

The ITER fusion device is a large experimental facility that integrates a large number of systems. From a control systems point of view, the complete system required to operate ITER is called the "ITER I&C system" [1,2]. The ITER control system architecture, as illustrated in Fig. 1 , has a hierarchical structure consisting of 171 instrumentation and control (I&C) plant systems that are distributed throughout the experimental facility, and are grouped into 18 control groups at the organizational level. These I&C plant systems are interconnected and are composed of elements using very different control technology solutions. Taking all these factors into account, ITER adopted EPICS as the distributed control system solution. Each plant system will account for tens or hundreds of thousands of data/signals to be exchanged and therefore stored.

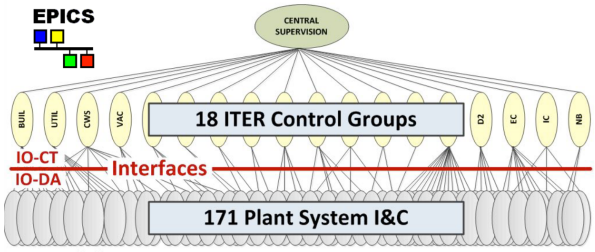


Fig. 1: ITER I&C system diagram

EPICS is a set of software tools and applications which provide a software infrastructure for use in building distributed control systems to operate devices such as Particle Accelerators, Large Experiments and major Telescopes. Such distributed control systems typically comprise thousands of computers, networked together to allow communication between them and to provide control and feedback of the various parts of the plant. To achieve this goal, each system in EPICS defines a set of control variables, the values of which are published and accessible to all other systems in the network. EPICS calls each of these variables a "Process Variable" (PV). Due to the large size of ITER, it will generate more than one million control signals in its first phase. And for its full operational state, it is estimated at more than three million control signals.

rodrigo.castro@ciemat.es

Throughout its different versions, EPICS has implemented two communication protocols for the exchange of data from its PVs. Up to version 3.16, EPICS has used the Channel Access protocol, and from version 7 onwards (inclusive), it has incorporated the pvAccess protocol, whose main change lies in the format of the PVs.

1.1. Requirements

The main requirements to be met by the EPICS control variable storage system in ITER [3,4] are:

- Continuous storage: Data must be able to be stored continuously over time regardless of the pulse number or other parameters associated with the experiment. Archiving services must support storage rotation mechanisms.

- Limited latency: Data that is stored must be accessible within a limited amount of time. Any solution that forces the file to "close" before making the data available is not acceptable.

- Changes over time: In the case of control variables, some of their properties may change over time. The storage system must be able to support them. Such as, for example: different type of data, different unit, different labels on enumerated types or different graphical properties (for example, limit values).

- HDF5 file-based storage: The base element for data storage in ITER is the HDF5 file. Although, this HDF5 file does not need to correspond at the physical storage level to an ".h5" file (as will be seen in point 2.1 below), it must correspond at the HDF5 API level.

- Minimizing the number of files: Due to the enormous complexity of ITER, it is key to design efficient solutions. As previously mentioned, the number of control variables will exceed one million. This forces the storage system to be able to pack in the order of 50K variables per file.

1.2. Alternative implementation

At present, there is one implementation "The EPICS Archiver Appliance" [5], that has some characteristics that are not aligned with ITER requirements.

Firstly, the implementation is done in Java, while all the rest of the services in ITER require to be implemented in C++, due to efficiency and optimisation criteria. Secondly, this implementation is based on a typical architecture (frontend - buffer - backend) but does not have a frontend scheme that allows an acceptable integration of other ITER data types, which would force to maintain several solutions with similar purpose and architecture. Finally, the storage is based on ProtocolBuffers technology that requires prior definition and compilation of the data type, which makes it difficult to store variables continuously, as they may change their structure and data type over time.

2. HDF5 data models

HDF5 [6] is a scientific-oriented file format with a complete set of libraries and management tools (supported by a wide scientific community), and with the main objective of making data platforms independent and portable. There is a list of important advantages that support the use of HDF5 files for data archiving:

- Self-description data management: any client can completely know and manage the format of the data stored in a file. This feature also helps to keep version compatibility.
- Huge file size support: there are no size limits apart from those derived from the operating system or file system.
- Optimized performance for data segment access: HDF5 perfectly fits the data access requirements for scientific data archiving in long pulse experiments.
- Data appending: it is possible to append new data to a file without copies or redefinitions.
- Read while write: it supports concurrent write and read access by default, which is one of the most important ITER requirements. This feature is called HDF5 SWMR (Single-Writer Multiple-Reader).

An alternative, which could be considered in this sense, would be NetCDF. This technology is based on a file format that was born with similar objectives to those achieved by HDF5. Over the course of its versions, HDF5 has reached such a level of performance and functionality that NetCDF currently uses HDF5 files as a base and maintains two features: concurrent writer-reader access and simplicity of data definition. The first functionality has already been provided natively by HDF5 through its SWMR (concurrent writer-reader) mode. And the second one is not within the HDF5 philosophy, as HDF5 has a less user-friendly, but more flexible and functional data definition system, which is one of the keys of this work.

2.1. HDF5 Data Model

The HDF5 technology has a multi-layer architecture in order to decouple the HDF5 data model and final storage technology. As it is showed in Fig. 2, the upper level is the Abstract Definition Model (ADM) [7]. This

layer includes concepts for defining and describing complex data stored in files. The ADM is a very general model which is designed to conceptually cover almost any specific data model. Many different kinds of data can be mapped to ADM objects, and therefore stored and retrieved using HDF5. The most significant ADM objects are:

- File: Main component that groups and contains the rest of elements of the model.
- Group: a collection of objects (or other groups).
- Dataset: a multidimensional array of data elements with attributes and other metadata. The dataset component is crucial in HDF5 library because the data read and write operations are optimized for every dataset and are decoupled of the rest of datasets.
- Datatype: a description of a specific class of data element. The most significant families or classes of HDF5 datatypes are: integer, floating-point, string, bitfield, opaque, compound, enum and array. HDF5 allows to define new datatypes by combining other types.

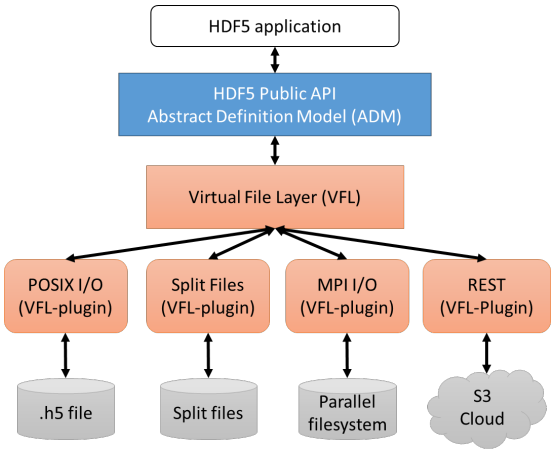


Fig. 2: HDF5 multi-layer architecture

Under the ADM layer we find the Virtual File Layer (VFL). It is an open interface that allows different concrete storage models to be selected. The VFL defines an abstract model, an API for random access storage, and an API to plug in alternative VFL driver modules (VFL-plugins). The model defines the operations that the VFL driver must and may support, and the VFL-plugin implements all the necessary read and write actions over a specific storage technology. Each driver isolates the details of the reading and writing storage so that the rest of the HDF5 library and user program can be almost the same for different storage methods.

2.2. Channel Access HDF5 data model

EPICS control variables in the Channel Access [8] protocol have a composite type format that mainly comprises the following fields:

- Status: System status
- Severity: Criticality of the alarm if in an alarm state.
- EPOCH Timestamp seconds: Section of seconds of the EPOCH timestamp from 1990-01-01T00:00:00Z.
- EPOCH Timestamp nanoseconds: Nanoseconds section of the EPOCH timestamp.
- Value: Value that the variable stores. This value field supports different basic data types: string, 16-bit integer, float, enumerated (16-bit integer for representing 16 states, one per activated bit), char, 32-bit integer, 32-bit float, and 64-bit float.

The data type of the value field can be obtained dynamically through the Channel Access protocol, so it is not necessary to have it defined in some kind of previous configuration, and in fact (as we will see later), it can vary over time.

In addition to the variable structure, there is another series of variable metadata that must be taken into account as part of the information to be stored and which are provided dynamically through the Channel Access protocol. There are two that are of particular importance. The first one, in the case of enumerated type variables, describes the list of values that are encoded. That is, the label represents each of the values of an enumerated variable. The second one describes relevant aspects when displaying the values of the variable, such as the limit values or value unit when representing the variable.

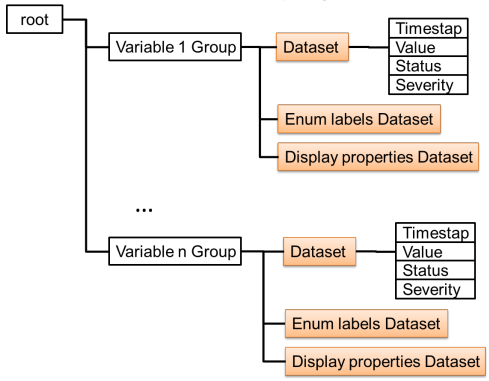


Fig. 3 Diagram of the HDF5 abstract data model design for Channel Access PVs archiving

In the case of the Channel Access protocol, the data model that has been implemented is presented in Fig. 3. It assigns an HDF5 group to each process variable. In such a group, a "payload" dataset is reserved for values and two additional datasets store display metadata and enumerated type labels. The payload dataset has a

compound type with the following fields: “timestamp” (uint64), “status” (int16), “severity” (int16) and “value”. The “value” field will have a datatype that will be aligned with its corresponding PV (“Process Variable”) value datatype.

This model presents important advantages. The first one is high aggregation rate. Thanks to this model it is possible to aggregate more than 65.000 variables in a single file. Because the huge number of PVs that are planned for ITER, this property reduces dramatically the complexity of the ITER data archiving system. The second advantage is a high data management granularity. Associating every PV with a payload dataset, every write and read operation will be optimized because it will not be affected by data from other PVs.

2.3. pvAccess HDF5 data model

EPICS incorporates, from version 7 onwards, the "pvAccess" protocol [9]. This allows the transmission of control variables (PVs) with a much more complex and highly recursive data structure, which, in the case of ITER, can contain more than 5000 fields in a single control variable. An example of this type of variables can be seen in Fig. 4, where different nested structures, arrays of structures and common data types, such as: integers, unsigned integers, floats, strings, etc., can be seen.

In order to store this type of nested data, an HDF5 data model has been implemented that conforms to this scheme. Specifically, a recursive model has been created with the following correspondence:

- Data structure: An HDF5 group is created.
- For each field with a complex data type (non-basic data type): An HDF5 group is created.
- Contiguous set of fields of basic types (float, int, uint, string,...): An HDF5 dataset of type compound containing these fields is created.

This model has several advantages:

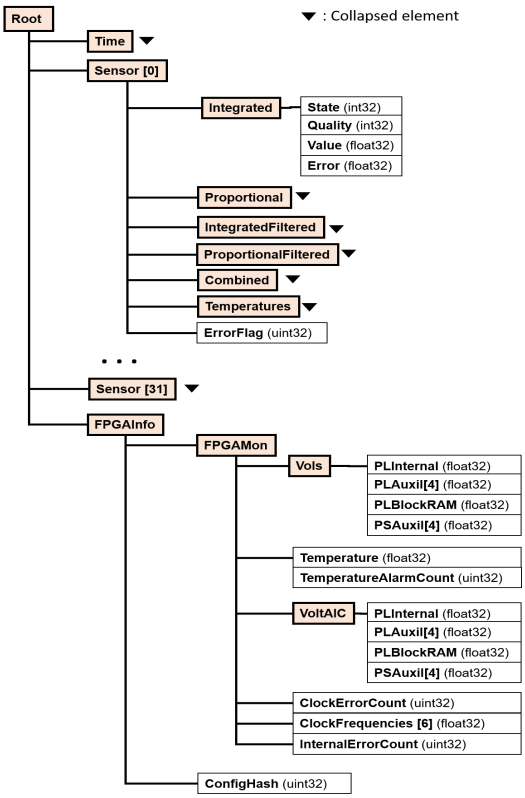
- There is a 1-to-1 map between the original structure of the variable pvAccess and the HDF5 model.
- The fields are accessible, thanks to the HDF5 API, through a simple path that follows the natural structure of the variable. For example based on the example in Fig. 4, "/main/Sensor[2]/Integrated/Quality".

Fig. 4 Example of pvAccess variable datatype with nested structures

- It is easy to access the data in a substructure (a branch of the GroupTree model).
- The model is optimized at the data reading level since accessing the values of a certain field only requires reading the data of its dataset, but not the rest of the data.

3. Data Archiving Server

The EPICS control variable storage server for ITER has been implemented on the basis of a plugin-based architecture with the aim of improving maintainability and code reusability. In addition to data storage services from SDN (ITER CODAC Synchronous Data Network), EPICS Channel Access and EPICS pvAccess protocols, ITER CODAC includes data streaming services to disseminate the same data over other systems. Thanks to this architecture it is possible to serve different types of data without the need to maintain different software.



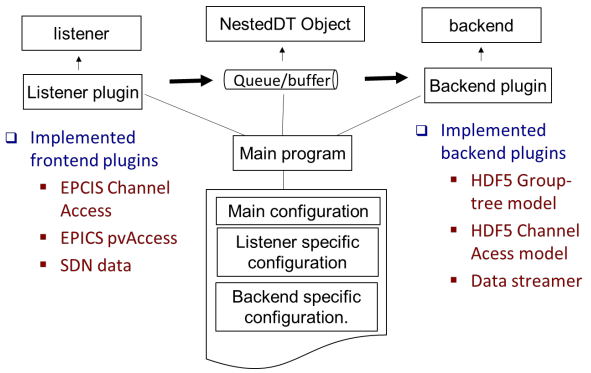


Fig. 5: Data archiver service architecture

As shown in Fig. 5, the architecture of the storage system is a classical frontend-buffering-backend architecture consisting of a front-end subsystem that receives data from the appropriate network and protocol, a buffering and queuing subsystem that stores the received data waiting to be processed, and a back-end subsystem that processes the buffered data and processes them according to the required functionality. This architecture implements most of the logic common to many frontend-buffer-backend systems and leaves it to the plugins to implement a small set of actions specific to the solution they support. This makes it easy to develop new services, to extend them to new data types, as well as to maintain those already implemented, since improvements and repairs are common to all of them.

A fundamental component of the system is the Nested Datatype Object (NestedDT). This internal object implements a common generic datatype that allows the same architecture to be maintained for very different datatypes. For this purpose, the data, which are received by the listeners, are translated into NestedDT and from this point on can be known and processed by the whole system, including the chosen backend plugin. NestedDT implements a recursive and self-descriptive type definition mechanism that allows the implementation of any of the data types already discussed. It is even possible to implement the complex pvAccess variable data types. NestedDT is designed in such a way that a NestedDT object mainly stores the data types, cardinality and payload positions of the different fields and structures. In this way, a NestedDT instance contains only the data type definition and is completely decoupled from the data payload, so that the "data type translation" of the variable to listen to NestedDT only has to be done once (usually at the time of connection to the data publisher) and is valid for the rest of the session until the publisher is either restarted or performs a data type change. In addition, this "definition-content" paradigm minimizes computational resources for data movement, handling and storage.

Finally, it should be noted that the system has a service configuration mechanism. This configuration includes a main section that is general to any service implemented with this architecture and two other sections specific to the listener plugin and the backend plugin to be used. Some examples of parameters that are part of the main section are: "storagePath" that defines the root location of the data to be stored (in case of storage), "streamDest" that defines the address of the destination server (in case of data streaming), "fileRotateSize" that defines the maximum size of the file before launching the rotation, "maxFlushDelay" that defines the maximum latency time before forcing data flush. In addition, this main section includes the list of variables to be stored. Each of them is identified as an “item” with a series of properties:

- name: Name of the variable to be stored.
- queueSize: Buffering size, in number of values, for this variable.
- timeField: Name of the field that represents the "timestamp" for this variable

Each of the subsystems that make up this architecture is detailed below.

3.1. Frontend and listener plugins

This is the subsystem responsible for, on the one hand, interacting through the chosen protocol and thus obtaining the data of the variables that have been defined in the configuration and, on the other hand, notifying and transmitting all the events and data received to the rest of the system. Its main functions are:

- Protocol management: it manages the connection, disconnection and error management through the protocol supported by the listener plugin. It is also responsible for generating the

corresponding events and notifications so that the rest of the system reacts appropriately.

- Auto-discovery data type and translation to NestedDT: Instantiates a NestedDT object (usually at the time of connection) for each of the variables to be monitored. To do so, it translates the format of the variable in the protocol corresponding to the format implemented by NestedDT and locates the position where to extract the timestamp of each value to be processed. For example, in the case of pvAccess variables, the data type of each variable is discovered at the moment of the connection with the protocol. At that moment, the pvaAccess listener is able to instantiate the NestedDT object associated to the data type of such variable.

- Buffer creation: after the frontend discovers the variable's data type, and can therefore calculate the size of its values, it creates a buffer associated with that variable for each backend to which it connects. As will be seen in the following section, the system allows for different frontend-backend connection configurations.

3.2. Buffering

In order to communicate asynchronously between the frontend and backend subsystems, a flexible buffering system is implemented to allow for different connection configurations, depending on whether it is needed to group variables on the same backend resource or if the values of one variable are to be used by several backends at the same time. In this sense, Fig. 6 shows a first example of a Channel Access Archiver service where several variables are grouped I,n the same backend, and a second example of a data streamer service where the data of a variable are distributed over the network to two destinations by means of the corresponding data streamer backends.

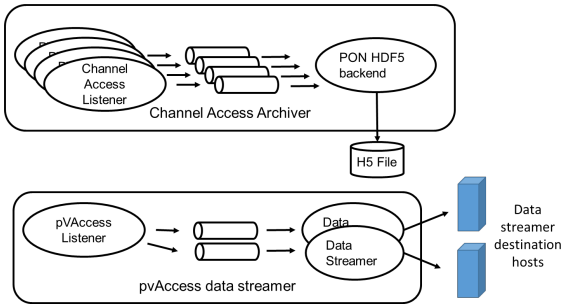


Fig. 6: Data buffering configurations

An auto-scaling mechanism has also been implemented in the buffering system, which allows adapting the size of a buffer depending on its progressive level of occupation. At the configuration level, a maximum buffer size limit (maxQueueSize), an occupancy rate that activates the auto-scaling mechanism (resizeQueueLimitRate) and a scaling rate -- applied each time the mechanism is activated (resizeQueueRate), are defined. At each insertion, the auto-scaling condition is checked:

```
if ((currentQueueSize / queueSize) * 100 > resizeQueueLimitPercentage)
```

If the condition is met and the maximum size is not exceeded, the scaling is performed:

```
queueSize += queueSize * resizeQueueRate
```

This mechanism ensures efficient buffering based on the activity level of each variable.

3.3. Backend and archiving plugins

The backend subsystem operates on a control loop in which it is able to detect new data in its buffers and pass it to the active plugin for proper processing. In addition, it implements timeout detection actions to force data flushing in case it is necessary, as there are storage plugins that perform some internal buffering in order to optimize write actions.

Also, the backend implements asynchronous file rotation mechanisms that avoid backend locking. In the case of the Channel Access PVs storage plugin, the rotation mechanism has to create a new file which (as previously mentioned) in the case of ITER can exceed 50K variables, and this action requires a non-negligible amount of time. For the implementation of the non-blocking rotation, and as shown in Fig. 7, we have chosen to

incorporate an asynchronous procedure carried out by a thread created ah-hoc. Once the file is completely created with all the required complexity, it becomes available as the active file of the plugin, and the previous one is closed.

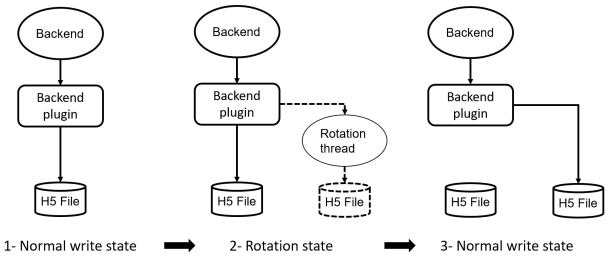


Fig. 7: File rotation mechanism of the Channel Access backend plugin

The main actions that are implemented at plugin level and are therefore specific to the storage or streaming mechanism it implements are:

- Related to the state of the file: Open / Close / AddVariable / RotateFile
- Data write related: AddVariableSamples / SetVariableMetadata / FlushAll

As can be seen from this list, a great effort has been made to simplify as much as possible the complexity at plugin level, in order to take advantage of the architecture and to facilitate the implementation of new services on these same data sources.

4. Conclusions

A storage service for ITER EPICS control variables (process variables or PVs) has been designed and implemented. This service is not only able to efficiently store the variables published through the classical "Channel Access" protocol, but also allows the storage of the EPICS pvAccess control variables (incorporated from version EPICS version 7) whose complexity lies in a highly nested data type that can also vary dynamically over time. The presented solution uses HDF5 files (which are de facto standard in ITER) and implements a data model that, on the one hand, preserves the original structure of the variable and, on the other hand, optimises the reading of certain fields of a variable, as it avoids the complete reading of the whole structure. In addition, this storage service is based on a frontend-buffering-backend architecture with plugins that makes it easy to develop new services by simply implementing the corresponding plugins.

References

[1] "Plant Control Design Handbook v7.0", https://static.iter.org/codac/pcdh7/Folder%201/1-Plant_Control_Design_Handbook_27LH2V_v7_0.pdf.

[2] A. Wallander, L. Abadie, H. Davea, F. Di Maio, et al., "ITER instrumentation and control—Status and plans", Fusion Engineering and Design, Volume 85, Issues 3–4, July 2010, Pages 529-534

[3] R. Castro, L. Abadie, Y. Makushok, et al., "Data archiving system implementation in ITER's CODAC Core System", Fusion Engineering and Design, Volumes 96–97, 2015, Pages 751-755, ISSN 0920-3796.

[4] Gheni Abla, G. Heber, D.. Schissel, D. Robinson, L. Abadie, et al., "ITERDB—The Data Archiving System for ITER", Fusion Engineering and Design, Volume 89, Issue 5, 2014, Pages 536-541.

[5] "EPICS Archiver Appliance!, "https://slacmshankar.github.io/epicsarchiver_docs/details.html"

[6] M. Folk, G. Heber, Q. Koziol, E. Pourmal, D. Robinson. "An overview of the HDF5 technology suite and its applications. In Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases (AD '11). Association for Computing Machinery", 2011, Pages 36–47. https://doi.org/10.1145/1966895.1966900

[7] "HDF5 User's GuideHDF5 Release 1.10", https://portal.hdfgroup.org/display/HDF5/HDF5+User+Guides

[8] "EPICS Channel Access protocol specification",https://docs.epics-controls.org/en/latest/specs/ca_protocol.html

[9] M. Sekoranja, M. Kraimer, G. White, et al.," pvAccess Protocol Specification", https://epics-controls.org/wp-content/uploads/2018/10/pvAccess-Protocol-Specification.pdf