Editor for Remote Database used in ATLAS Trigger/DAQ

Christophe Meessen*, François Touchard* and Jan Valenta†

**CPPM, 163, avenue de Luminy, Case 902, 13288 Marseille cedex 09, France*

†Institute of Physics, Na Slovance 2, 18221 Prague 8, Czech Republic

Abstract. This article describes the RDB Editor, the application for user-friendly viewing and editing of the contents of the RDB database used in the ATLAS Trigger/Data Acquisition. The application provides convenient navigation between huge number of stored objects and the possibility of developing plug-ins that fit the special needs of different working groups.

Keywords: ATLAS, HLT, Remote Database

PACS: 07.05.Hd

REMOTE DATABASE

Many of the components of the ATLAS T/DAQ system use a common database for storing their configuration data. Recently, the database used is the Remote Database[1] (RDB). This database uses XML files for schema definition and data storage, and an RDB server for accessing them. The schema files describe the class structure of the database. It states what is the relation between particular classes (they can be inherited, for example), limits the types and values for particular attributes, lists the possible links to other classes, etc. The data themselves are stored in the database in form of objects, each of them implementing particular class. The objects already have concrete values assigned to attributes and relationships.

There is a pair of tools for direct access to the XML files and altering the database schema or the data themselves (OKS Schema Editor and OKS Data Editor, see Chapter 3 in [1]). The application described in this article uses the `rdm_server` program and accesses the XML files via this one additional layer.

RDB EDITOR

Because a huge amount of objects is foreseen in the RDB in the future and each project using the database will need to conveniently navigate between them, Java application called “RDB Editor” was developed. The RDB Editor provides the user with an easy-to-use GUI and implements the following features:

- easy navigation between already visited objects.
- several views on history of visited objects. There is also the possibility of searching in the history and of sorting it according to the name of the object or time.
- bookmarks, including managing.

- write support.
- graphical viewer for special classes of objects. As an example, the viewer of `EFD_Configuration` class is provided.

The application is written in Java to be possibly able to take advantage of integration into the ATLAS Online Software as a separate panel. The Eclipse Development Platform¹ was used during the development for analyzing, coding, and debugging.

Internal Structure

The structure of the RDB Editor application is outlined in Figure 1. The lowermost `rdbComm` layer cooperates with the RDB interface provided by the `rdb_server`, the `confCore` component implements the higher-level functionality including bookmarks and histories support and provides user-friendly interface for RDB access. Finally, the topmost layers are clients, both general and specialized.

FIGURE 1. Software layers in the RDB Editor.

The `rdbComm` layer is responsible for the lowest level communication with the database. It provides methods for accessing the RDB for both reading and writing. There is also a possibility to read the schema and data separately.

There are also several helper classes defined in this layer. Their full description, including description of their members and methods, can be found in the Javadoc documentation[2]. Among the most important ones are `Attribute`, which encapsulates the properties of particular attribute, and `AttributeValue`, which represents name and value of a particular attribute.

The `confCore` layer just adds some functionality to the underlying layer, otherwise it just forwards the database access requests to the `rdbComm`. This added functionality is, for example, support for bookmarks, history of visited objects, and navigation track.

¹ <http://www.eclipse.org>

There is the GUI appearance and functionality defined in the `rdbMng`. The main class here contains the main method and logic responsible for program running, the rest of classes defined in this layer are just helper GUI classes.

For example, there are classes that provide the additional functionality of the `ComboJTable`, like the hypertext nature of object identifiers in the table (in relationship lines) or the ability of editing the values.

The GUI defined by this set of classes is able to browse through RDB database containing any set of objects. The easy-to-use navigation through the database is supported for all objects defining the relationships and the histories are available all the time. It is universal RDB viewer and writer.

The application also supports addition of custom graphical viewers for displaying specific object relationships which are not easily understandable with the default viewer. As an example, the viewer for `EFD_Configuration` instances was implemented. The objects of this class represent the configuration of Event Filter Dataflow, the way the event data are routed between the SFIs and SFOs. Many different tasks may deal with the event in-between, they may for example log some data, make some computation, move the data to the trash, etc.

Browsing of the `EFD_Configuration` in the textual view can be quite complicated, as loops in the event dataflow may occur or the net of the processing tasks may be very large. Therefore, the graphical plug-in may help with understanding of the dataflow net.

When the user displays an object of the `EFD_Configuration` class, the second panel in the attribute-area appears, showing the graphical representation of the object showing the user the configuration information he needs in a comprehensive way.

The configuration graph is implemented using the `JGraph[3]` component. This is a framework for creating Java applications working with various graphs or diagrams.

Graphical User Interface

As one can see in Figures 2 and 3, the application window can be divided into three main sections. There is a detachable toolbar on the top, tabbed histories display on the left, and potentially tabbed properties viewer which occupies the rest of the window. There are also two separate windows designed to provide the GUI for bookmarks managing and preferences setting.

The menu in the main window of the application provides access to preference settings, database connection command, and Bookmarks options. In the preferences, the user can store the value used for 'Home' object and the database name used for connecting to the `rdb_server` (other possibility is the command-line parameter). The bookmarks submenu offers organizing and saving/loading of the bookmarks.

The detachable toolbar under the menu contains navigation buttons, object selection items, and 'Show' and 'Write' buttons. The object selection comboboxes allow the user to select the class and consequently an object of that class. The 'Show' button actually loads the selected object into the properties panels and the 'Write' button writes the changes made to the attributes or relationships to the database.

The screenshot shows the 'RDB Management Application' window. The 'Table view' tab is active, displaying the properties of the 'EFD_Configuration.efd-cfg-in-ext-out' object. The left sidebar shows a list of objects with their IDs and timestamps. The main panel is divided into 'Relationships' and 'Attributes' sections.

Name	Value
Relationships:	
log	EFD_LogFilter.efd-cfg-fltr-normal
logEvent	EFD_LogFilter.efd-cfg-fltr-normal
logWork	EFD_LogFilter.efd-cfg-fltr-normal
logWorker	EFD_LogFilter.efd-cfg-fltr-normal
ctlBarrier	EFD_Barrier.efd-cfg-barrier-in
inputTasks	EFD_InputTask.efd-cfg-task-in-3
sharedHeap	EFD_SharedHeap.efd-cfg-sharedheap-256mb
Attributes:	
name	in-ext-out
desc	input->extPT->output
nbrWorker	1
stopTimeout	20
unconfigTimeout	20
statePoll	2
statUpdateRate	2

FIGURE 2. Textual view on the EFD_Configuration object.

FIGURE 3. Graphical view on EFD_Configuration object.

There are several views on the history of visited objects in the program. The default one, 'History', logs the name of the visited object and the time it was visited. This panel allows the user to see what object was displayed at any point in the history.

The 'Last Visited' tab works as sort of live bookmarks. One can see the list of 15 last visited objects there with no duplicate records and no timestamps. There is also a possibility of searching and sorting the histories.

The largest part of the application window is occupied by the properties view. Figure 2

shows the textual view on the properties of the `EFD_Configuration` object. The relationships values act as hypertext links.

The graphical plug-in for displaying the objects of the `EFD_configuration` class is shown in Figure 3. The displayed tasks are also hypertext links; hovering over an object shows detailed information in a tooltip.

Scalability

The component `EFDCnfGraph` is self-contained part of the program. Similar functional blocks can be developed by members of any other group which is storing its configuration data in the `rdB` database.

The interconnection of the newly developed specialized part and the core program involves some minor changes of the program source code and a recompilation. The developer of such a plug-in (graphical or not) may and should use the infrastructure provided by lower levels of the `RDB Editor` application; namely the standardized access to the database via the `confCore` interface is a must to keep the histories and track of visited objects up to date.

SUMMARY

The application described here is meant to help people using the `RDB` database for storage of huge numbers of object to stay oriented. It offers comprehensive way of navigation and the possibility of developing specialized viewers.

ACKNOWLEDGMENTS

The work was done during Marie-Curie fellowship of Jan Valenta at CPPM Marseille. Thanks go also to ATLAS T/DAQ collaboration for developing the whole Trigger and Data Acquisition system and maintaining great documentation.

REFERENCES

1. I. Soloviev: *Configuration Databases User's Guide*, ATLAS DAQ Technical Note 135².
2. C. Meessen, F. Touchard, J. Valenta: *Java application for rdb management*, Javadoc documentation³.
3. JGraph, Ltd.: *The Java Graph Visualization Library*⁴.

² <http://atddoc.cern.ch/Atlas/DaqSoft/components/configdb/docs/confdb-ug/2.3/pdf/ConfDB.pdf>

³ <http://cern.ch/jvalenta/rdbMng>

⁴ <http://www.jgraph.com>