

PAPER • OPEN ACCESS

## A new type of smart pointer for data object reference both in memory and in root files

To cite this article: T Li and X T Huang 2016 *J. Phys.: Conf. Ser.* **762** 012001

View the [article online](#) for updates and enhancements.

### Related content

- [Monitoring of quantum mode correlation functions in the presence of pointer state phase relaxation](#)  
E Trifanova and A Trifanov
- [Next-Generation Navigational Infrastructure and the ATLAS Event Store](#)  
P van Gemmeren, D Malon, M Nowak et al.
- [Purity sieve for models with factorizable interactions](#)  
Oleg Lychkovskiy

### Recent citations

- [Design and development of JUNO event data model](#)  
Teng Li *et al*
- [Application of SNiPER framework to BESIII physics analysis](#)  
Xin Xia *et al*

# A new type of smart pointer for data object reference both in memory and in root files

T Li<sup>1</sup> and X T Huang<sup>1</sup>

<sup>1</sup>School of Physics, Shandong University, No.27 Shanda South Road, Jinan, Shandong, 250100, China

liteng@hepg.sdu.edu.cn, huangxt@sdu.edu.cn

**Abstract.** Based on ROOT framework, we develop a new mechanism named as SmartRef for event data correlations in offline data processing of High Energy Physics experiments. SmartRef uses the Universally Unique Identifier in ROOT to handle correlations between event data objects, both in memory and in ROOT files. It also provides a lazy-loading functionality to speed up event selection processes and simplify data input system. We have applied SmartRef to the JUNO offline software and used it as a test-bed. The test results show that SmartRef provides a significant improvement in event selection processes.

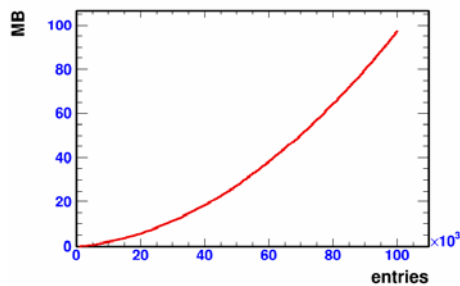
## 1. Introduction

In offline data processing applications of High Energy Physics experiments, the event data model defines entities of the event data management, file input/output (I/O), event data storage and physics analysis. Therefore, the design of event data model plays an important role for functionalities and performances of the whole offline software.

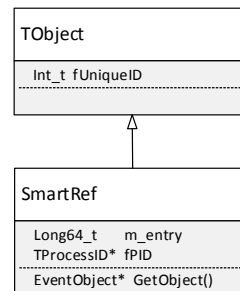
From the physics analysis point of view, neutrino physics experiments are different from accelerator physics experiments. The formers such as Daya Bay [1] and JUNO [2, 3] have two special requirements. Firstly, time relations between signal events must be supported due to the working principle of the Liquid Scintillator detector [4]. Secondly, high efficient data access and storage capabilities must be implemented due to the rare neutrino signal events compared with the backgrounds.

In order to meet the requirements mentioned above, both pointers of C++ and TRef of ROOT are investigated. Plain C++ pointers or smart pointers within the standard C++ library can only handle the relationship of objects in memory, but the correlated objects cannot be saved into different files. TRef is provided by ROOT [5] as a mechanism for the correlation of TObjects, but does have some limitations. Firstly, the lazy-loading mechanism of TRef requires correlated objects to be saved within TTrees with the same entry number, and it is designed for object correlation within one event. If TRef is used for object correlations across different events, the TBranchRef, which holds extra information for the lazy-loading mechanism, takes too much space. Figure 1 shows that the size of TBranchRef rapidly increases as a function of the number of correlated objects saved into a TTree.





**Figure 1.** Space usage of TBranchRef



**Figure 2.** Design of SmartRef

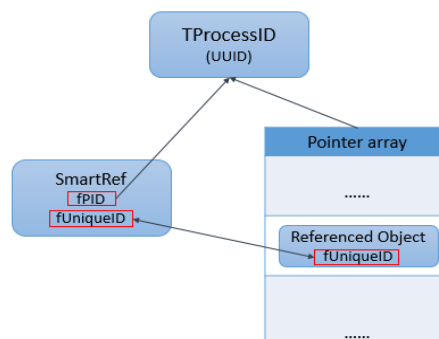
## 2. SmartRef

### 2.1. Design of SmartRef

SmartRef is derived from TObject of ROOT and its key members are listed in Figure 2. The fUniqueID variable of type integer and the fPID, a pointer to the TProcessID object make use of the UUID (Universally Unique Identifier) tool [6] in ROOT to build correlations between SmartRef and its referenced object (RO). The m\_entry variable of type long integer records location of the RO when written into a ROOT file in term of TTree. The GetObject() function is designed to retrieve the RO and directly returns a pointer to it if the RO has been already loaded into memory. And if not, SmartRef firstly invokes the lazy-loading mechanism to load the RO from ROOT files and then returns a pointer to it.

### 2.2. Working principle

UUIDs are deployed to build correlations between SmartRef and its ROs. In a ROOT process, one or several instances of TProcessID are automatically created by the global TROOT, each of which keeps a UUID and one global integer. Associated with the TProcessID instance, an array of pointers is created simultaneously. When a SmartRef refers to a TObject, the value of the global integer is added by 1 and assigned to the fUniqueID member of both the SmartRef and the referenced TObject. Meanwhile, the fPID member of the SmartRef is set to the address of the current TProcessID and the RO is put into the n-th (n equals to the fUniqueID) slot of the array. The reference between SmartRef and its RO is constructed with the UUID and fUniqueID as showing in figure 3. In reverse, when users retrieve the RO, the SmartRef underlyingly queries its fPID and fUniqueID to get the array holding ROs and the position of the RO in the array.



**Figure 3.** Working principle of SmartRef

When a SmartRef and its RO are written into a ROOT file, the corresponding TProcessID are also written into the file at the same time. When the ROOT file is opened, TProcessIDs are automatically

loaded into memory, and the corresponding array is created simultaneously. Once a RO is loaded into memory, it is put into that array accordingly.

### 2.3. Lazy loading

For neutrino experiments like Daya Bay and JUNO, one signal event consists of a very large load of data. It is not necessary and inefficient to read the whole event into memory at one blow during the event selection process because only partial of the event data is enough to perform fast event selection. And it adds burden to I/O operations if more redundant information is read. In this case, the lazy-loading mechanism is very essential.

We design the SmartRef and an I/O manager module, InputElementKeeper, to achieve the lazy-loading functionality. During an output process, when a TTree holding ROs is created, a special table, TablePerTree, which keeps integer values, is also created. Whenever a RO is written into a TTree, its fUniqueID member is saved into the TablePerTree too. Before a ROOT file is closed, all objects of type TablePerTree are written into this file associated with the TTrees. At the same time, the corresponding UUID of the TProcessID are also written into the file as the meta-data.

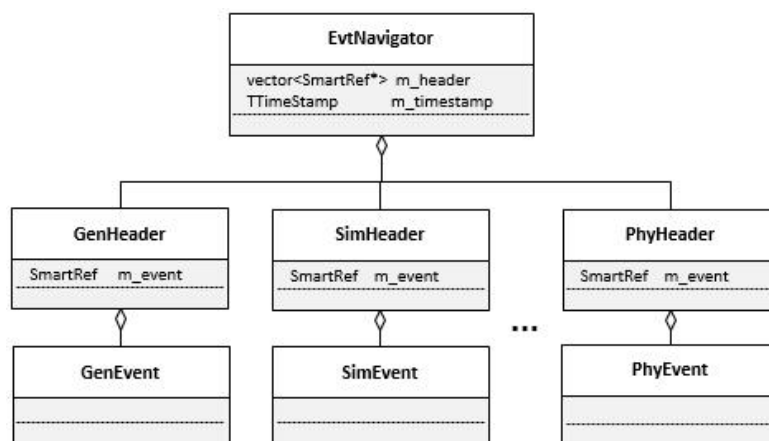
When the input system is initializing, the InputElementKeeper module scans all input files, analyze the metadata of these files and then construct a table, which keeps the mapping relationship between UUIDs and files. When users access an object referenced by a SmartRef instance, the SmartRef instance firstly looks it up in the memory to check if the object has been loaded. If not, it will query the InputElementKeeper and search for the file that holds the RO. Once the file is found, InputElementKeeper will search the TablePerTree and load the RO into memory.

Comparing with the relatively large event data object, the metadata of files and TablePerTrees only take a small space, about 4 bytes per event object, and it takes  $O(1)$  time complexity to search the TablePerTree. So the efficiency of lazy loading is high, and the extra space needed is relatively small.

## 3. Application in JUNO

The SmartRef mechanism is successfully deployed in the offline software of JUNO, and it plays an important role in the event data model.

### 3.1. Design of JUNO event data model



**Figure 4.** Design of JUNO Event Data Model

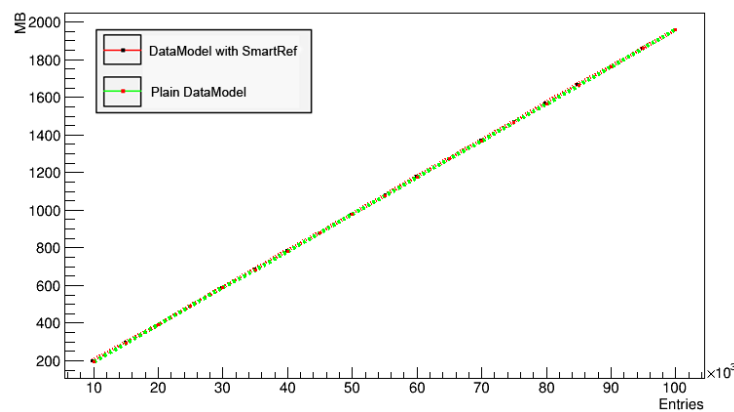
Figure 4 shows the design schema of JUNO event data model. All the event classes are derived from TObject directly or indirectly in order to take advantage of functionalities of data management provided by ROOT. To achieve quick event selection, the event data objects are divided into two levels: header and event. They keep the meta-information and full-information, respectively. In each header class, one or more SmartRefs are deployed to correlate its event objects. In this way, users are

able to get the header of one event quickly for the meta-data and make decisions whether or not to load the whole event object to perform further analysis.

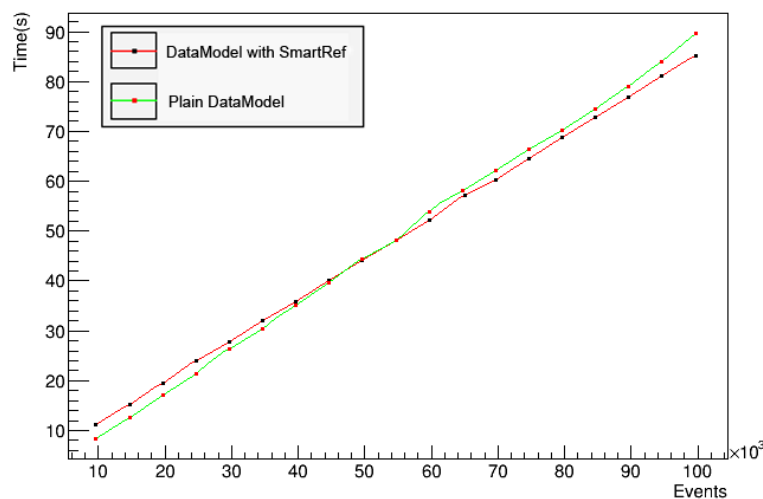
The EvtNavigator is another application of SmartRef. It keeps a list of SmartRef instances that refers event headers of different data processing stages (including generator, simulation, reconstruction, calibration, etc.) of one event and is used to easily navigate objects between different processing stages in one event. It turns out that SmartRef makes the correlation analyses of the inverse beta decay very conveniently and efficiently.

### 3.2. Performance test

To test the performance of SmartRef mechanism, we use JUNO offline software framework based on SNiPER [7] as a test-bed and generate two large statistical data samples with and without SmartRef, respectively. Figure 5 and Figure 6 show the comparison of the space usage of ROOT files and the time consuming of loading the data from ROOT files in two cases. The result shows that the extra space needed by SmartRef is very small, and the speed of lazy-loading and normal-loading is almost the same.

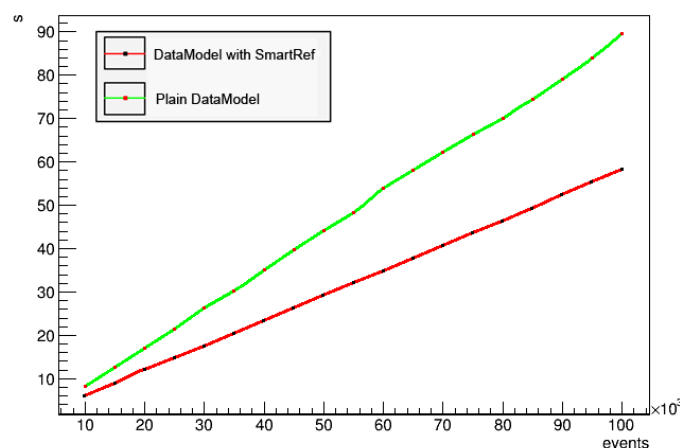


**Figure 5.** Comparison of space usage between plain data and data with SmartRef.



**Figure 6.** Comparison of the loading-speed between plain data and data with SmartRef.

We also test the event selection process by randomly selecting about 50% of the generated event data and comparing the time consuming of two cases. The result shows that the lazy-loading speeds up the event-selection process by a factor of 35%, as shown in Figure 7.



**Figure 4.** Comparison of time consuming of event-selection between plain data and data with SmartRef

#### 4. Conclusion

We design and develop a smart pointer mechanism, called SmartRef, based on ROOT. The mechanism supports lazy-loading from persistent ROOT files and correlations between objects within one event and across events. The testing shows that it works very well in JUNO and might be used by other experiments too.

#### Acknowledgements

We acknowledge the supports from the Joint Large-Scale Scientific Facility Funds of the NSFC and CAS (U1532258), the Program for New Century Excellent Talents in University (NCET-13-0342) and the Shandong Natural Science Funds for Distinguished Young Scholar (JQ201402).

#### References

- [1] F. P. An et al. (Daya Bay Collaboration), A Precision Measurement of the Neutrino Mixing Angle  $\theta_{13}$  using Reactor Antineutrinos at Daya Bay, arXiv:1202.6181
- [2] Z. Djurcic et al. (JUNO Collaboration), JUNO Conceptual Design Report, arXiv:1508.07166
- [3] F.P. An et al. (JUNO Collaboration), Neutrino Physics with JUNO, arXiv:1507.05613
- [4] F. P. An et al. (Daya Bay Collaboration), A side-by-side comparison of Daya Bay antineutrino detectors, Nucl. Instrum. and Meth. A 685 78 (2012).
- [5] R. Brun, F. Rademakers, ROOT — An object oriented data analysis framework, Nucl. Inst. Meth. in Phys. Res. A 389 (1997) 81–86.
- [6] Balkić Z, Š o š tarić D, Horvat G. Geohash and uuid identifier for multi-agent system, Technologies and Applications Lecture Notes in Computer Science Volume 7327, 2012, pp 290-298
- [7] J. H. Zou et al. SNiPER: an offline software framework for non-collider physics experiments, J. Phys. Conf. Ser., 664(7): 072053 (2015)