Data Base for the X-ray Tomograph

Yu. Sedykh, Yu. Smirnov

Abstract

We describe the Objectivity C++ data base of the X-ray tomograph which was developed to store the scan parameters and the scan analysis results.

1. Introduction

The X-ray tomograph ([1], [2]) has been created to measure the MDT chambers of the ATLAS Muon Spectrometer ([3]) in order to certify and control the chamber production lines. A measuring rate of the automated tomograph is estimated to be 2 chambers per week, that is about 10 scans per week. During 4 years of chamber production and measurements about 2000 scans will be done. Evidently the scan parameters and results of scan analysis must be put in the data base to provide a standard mechanism to store and retrieve the data.

Amongst the possible choices we considered well known relational data bases (Oracle, MySQL, Access, etc.) and the Object Oriented (OO) Objectivity/C++ data base ([4]). The PC oriented ones (like Access) were rejected because of bad compatibility with UNIX operating system where the X-ray tomograph DAQ system running, and because of instability of the Windows 95 or 98 systems. Comparing other relational data bases and the Objectivity/C++ one we have chosen the latter. The main reasons of this decision are the following.

1. Objectivity/C++ has all main features of the relational data bases. But it has some essential benefits over them.

2. The structure of the data is flexible, that is we can create a data tree with arbitrary number of branches (including empty). It is very convenient because we can not predict how many parameters we will have to put into the data base. It strongly depends on the certification

procedure which has not been designed yet. So, we are able to reprocess any scan, calculating new parameters (decided to be calculated after the data base has been created), put the results into the data base not changing its common structure, for example, searching procedure where the new parameters may be searched for as well as for any old one.

3. Objectivity/C++ provides a natural interface for programmers. We can use data structures similar to the C++ language ones.

The computer code based on C++ language has been developed to write data into the data base. The data flow is organised in the following way. Data acquisition system makes a scan, puts common scan parameters (like scan date and time, chamber name, type, scan section, etc.) into the header file, the slow control parameters (temperature of the air and the chamber in some points, pressure, humidity, high tension of the X-ray sources, etc.) into the corresponding file, starts the "scana" program for data reconstruction ([5]), which will put the results (wire coordinates, standard deviations of the grid fits and many others) into the results data file. After any of these three files has been created the corresponding "DB write" program starts to read the file and puts the data into the data base. If the data base server is not reachable the program is postponed and starts again later for every file which must be put into the data base. As soon as data have been put into the data base successfully the original files are destroyed (more precisely they are copied to some scratch area to keep them for some time for the case of unrecognised problems with the data base) to indicate the success.

2. Structure of the Data Base

As it was mentioned already the structure of the data base must allow to add new parameters not changing the DB structure itself. Another thing is the necessity to reserve a possibility to add additional fields to the structure describing the wire position in space and in chamber for future development of the X-tomograph system. To provide these features the following structure of the Objectivity/C++ data base has been created.

Generally the structure of the data base is present in Fig. 2.1. The data for every scan are put in independent structures. Also there exists a table of names described later. It is also independent on other classes.

The structure of individual scan is shown in Fig. 2.2.

The data base consists of the 9 classes: Scan_Links, Headers, SC_Records, Scan_Results, Wires, Reserv_Wire, Mytable, SC_Parameter and Scan_Wire.

Results and other parameters of the X-Ray Tomograph measurements are presented in the Headers, SC_Records, Scan_Results and Wires classes. The Headers class contains the general parameters of the scan like date, time, scan number, production site name, chamber name, operator name, etc. The Headers parameters are written only once per scan. The SC_Records class contains slow control parameters like temperature, humidity and pressure in the clean room, chamber temperature in some points, electrical parameters of different devices (X-ray tubes, power supply of photo multipliers, power supply for Front End Electronics), gas supply parameters, etc. Those parameters are measured periodically during the scan (1 time per 100 events, i.e. 1 time in 4 seconds) and must be put in the data base after scan finishes. The Wires class contains the parameters of the wires, calculated by “scana” program for every scan. The Class_Results class contains general geometrical parameters of the measured chamber like standard deviation of the wire coordinates relative to the ideal grid corresponding to the nominal values, distances between layers, multilayers, many other parameters. Usually the Wires and Class_Results classes are written one time per scan but there is a possibility to have more versions of the same parameters. This feature has been foreseen for the case if some essential improvement of the “scana” program is made and it is necessary to reanalyse some or all scans with the new version of the program. So, a few versions of the analysis results may be put in the data base. The last one is default.

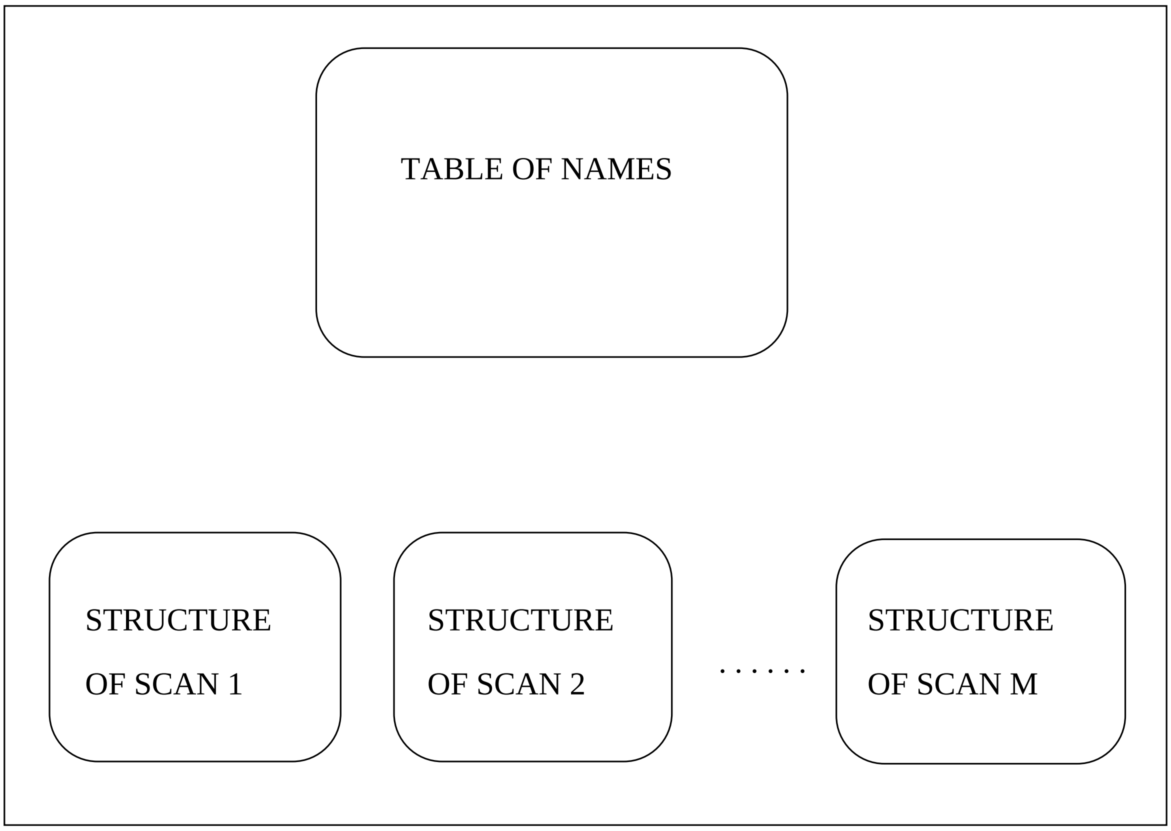


Figure 2.1 General organization of data

Physically every parameter is put in a separate object. Those objects are connected together and associated with a certain scan via the Scan_Links class which contains the association links to the Header, Result and Wire parameter objects directly. The Slow Control parameters are put in the SC_Parameter objects which are combined via VArray Objectivity/ C++ construction into the SC_Records objects and the Scan_Links class has association links to the SC_Records objects.

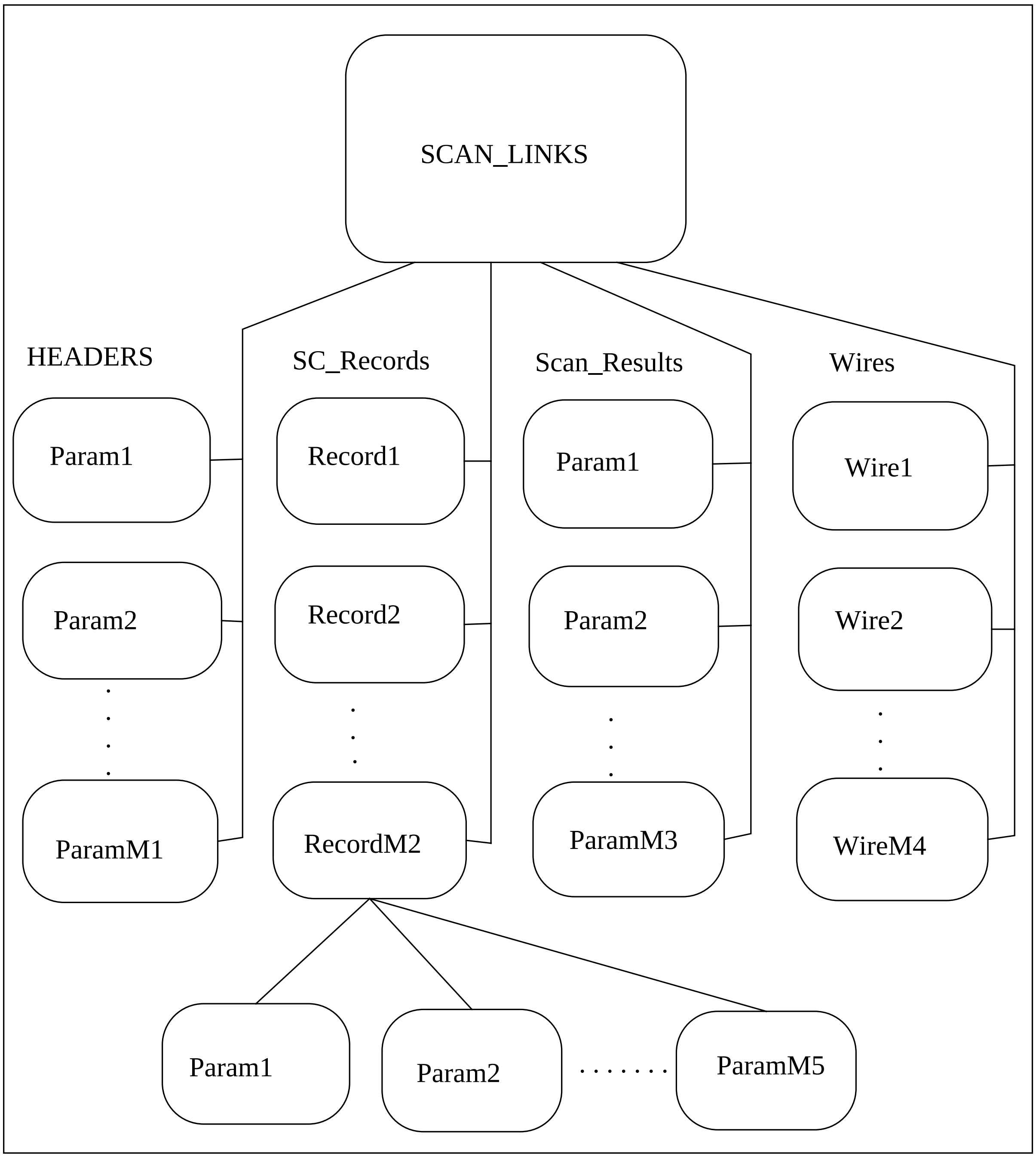


Figure 2.2 Structure of scan

It is evident that at the lower level of work with the data base a use of the numbers is better than the use of parameter names (character strings) from point of view of disk space and data access time (for example, for search purposes). Class Mytable is used to give the correspondence between each parameter name and number. So, at the lower level only numbers of the parameters are used.

The Reserv_Wire class is an auxiliary class which is not used at the moment, but may be used in the future if some additional information must be added for individual wire parameters.

Let us shortly describe the detailed structure of the data base.

To develop the computer code to create and support such a data base we used Objectivity 5.1 and C++ compiler working on SUN stations under Solaris 2.5.1. The ATLOBJ01 node has been used as the lock server, the application programs were compiled and run on nodes ATLAS50 and SUNEPEOS01.

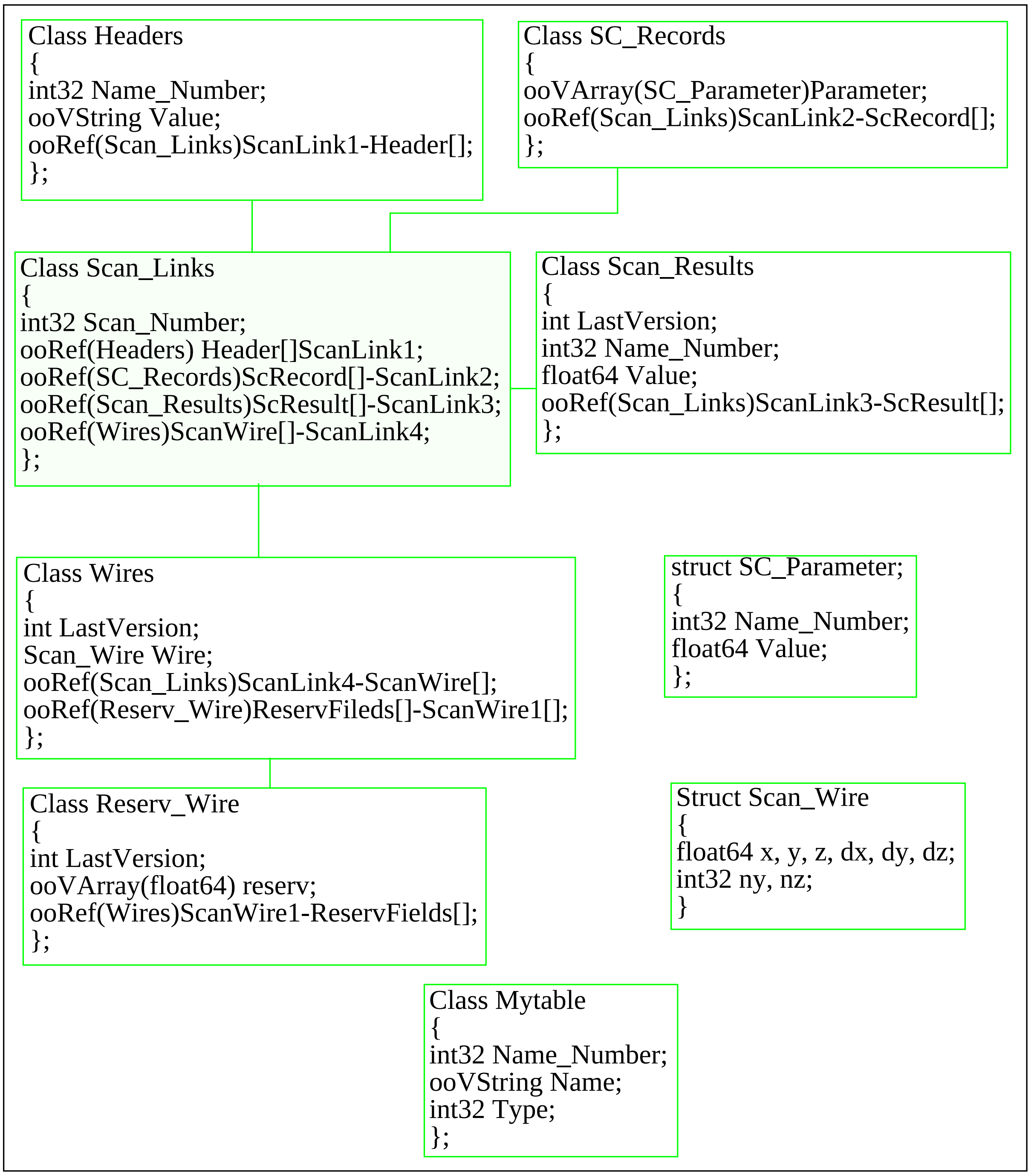
A federated data base named XTOMODB has been created. It contains a data base XTOMO1_db. To store the objects of corresponding classes several containers have been used.

We keep the objects of Mytable class in the container named MytableContainer. Class Mytable consists of three members. For every scan each parameter of measurements has a unique name which is stored in the string “Name” of variable length. The second member of Mytable class is the integer variable “Name_Number”. These two fields establish a correspondence between parameter name and number of the parameter. We will use Name_Number instead of Name of parameter to navigate and organize searching procedure faster and simpler. It is naturally to deal with integers instead of strings, and so we can simplify and accelerate the search of parameters in Mytable, which can be loaded into RAM directly when an application program is working. The third member of that class is the integer “Type”. This is a flag equalled 1 for the Headers class parameter, 2 for parameter of the SC_Records class and 3 for the parameters of Scan_Results or Wires class.

Container ScanLinksContainer is used to store all necessary association links to objects of classes Headers, SC_Records, Scan_Results, and Wires, which are described below (see Fig. 2.3). In this container we keep the class Scan_Links, which consists of 5 members: integer Scan_Number, which is a main input parameter and is an identifier of the current scan, and four association links to objects of Headers, SC_Records, Scan_Results and Wires classes. Here we create a key object and deal with the key search using the field Scan_Number as a key for the class Scanl_Links.

Container HeadersContainer is used to store the objects of Headers class. Headers class consists of the following members: integer Name_Number, the string of variable length Value and the many-to-one association link to an object of the control class Scan_Links. Many-to-one association is used here to link all header parameters within one scan. In the same container we keep an index with two key fields: Name_Number and Value. So we can realise a search within Headers class using both key fields.

Container SlowControlContainer is used to store the records of slow control parameters. The corresponding class SC_Records includes as a member the variable-size array Parameter. Each element of this array belongs to the structure SC_Parameter, which consists of integer Name_Number and float Value and represents one slow control parameter. We use the variable-size array here because the number of parameters in the slow control records may be different in general case. And the second member of the SC_Records class is a many-to-one association link to an object of the control class Scan_Links. Such an association is needed to connect all slow control records obtained within one scan.



```
Class Headers
{
int32 Name_Number;
ooVString Value;
ooRef(Scan_Links)ScanLink1-Header[];
};

Class SC_Records
{
ooVArray(SC_Parameter)Parameter;
ooRef(Scan_Links)ScanLink2-ScRecord[];
};

Class Scan_Links
{
int32 Scan_Number;
ooRef(Headers) Header[]ScanLink1;
ooRef(SC_Records)ScRecord[]-ScanLink2;
ooRef(Scan_Results)ScResult[]-ScanLink3;
ooRef(Wires)ScanWire[]-ScanLink4;
};

Class Scan_Results
{
int LastVersion;
int32 Name_Number;
float64 Value;
ooRef(Scan_Links)ScanLink3-ScResult[];
};

Class Wires
{
int LastVersion;
Scan_Wire Wire;
ooRef(Scan_Links)ScanLink4-ScanWire[];
ooRef(Reserv_Wire)ReservFileds[]-ScanWire1[];
};

struct SC_Parameter;
{
int32 Name_Number;
float64 Value;
};

Class Reserv_Wire
{
int LastVersion;
ooVArray(float64) reserv;
ooRef(Wires)ScanWire1-ReservFields[];
};

Struct Scan_Wire
{
float64 x, y, z, dx, dy, dz;
int32 ny, nz;
}

Class Mytable
{
int32 Name_Number;
ooVString Name;
int32 Type;
};
```

Figure 2.3 Detailed description of classes.

In the ScanResultsContainer we keep objects of both Scan_Results and Wires classes. Class Scan_Results is used to store results of data processing and has the following members: integer Name_Number, float Value and integer LastVersion. Data can be processed using different techniques, methods, tools etc. So it is possible for user to have several versions of results. LastVersion equals to 1 if it represents the last default version of results, for all previous versions this parameter is equal to 0, and it is the user responsibility to differ from such a versions of results, obtained before (for example one can use a parameter ResultsVersionNumber to get a number of corresponding version of results). And as before the last member of class Scan_Results is an association link to the object of the class Scan_Links, representing the given scan. The Wires class is used to store results of measurements of wire positions and consists of integer LastVersion (having the same meaning as for Scan_Results class) and the second member of class Wires is Wire, which has type of structure Scan_Wire, consists of float variables x, y, z, dx, dy, dz, which represent coordinates and errors of measurements of wire positions, ny - number of layer and nz - number of wire within the layer. The third member of the Wires class is an association link to the object of the class Scan_Links, and the last one is an association link to the object of class Reserv_Wire. Class Reserv_Wire can be used to store some additional information about wires in the future. Members of this class are integer LastVersion, variable-size array of float reserv, in which necessary parameter values may be written, and an association link to the object of class Wires. One can recommend to store the objects of the Reserv_wire class in the same container as Wires, i.e. ScanResultsContainer for the easy navigation. In the ScanResults container we store also the index, which helps us to realise the faster access to the objects of Scan_Results class. As the key fields for the index for the Scan_Results we use both Name_Number and Value.

3. Application programs for writing into data base and for formatting of the input data

To write into data base three application programs have been developed. First one called Headerswr.cc is used to write the information about header’s parameters for current scan into the data base. The input file has the following format. Comments (they begin with the # character) are ignored. Each Value of header parameter can occupy a number of lines in the input file. Integer value representing a number of lines for the Value of the parameter is coming first, then the Name of current parameter is read. We use the following conventions: the Name can consist of from one up to 20 words separated by one or several blanks each from another. The program will write into the data base the Name, in which the parts (if there is a number of parts) are separated by one blank, automatically deleting all others. After that the Value of the parameter is read from the following one or several lines in accordance with the number of lines have been read before. The program writes Name and its corresponding Name_Number into the MytableContainer if it is not there yet and Value will be stored in the HeadersContainer. This procedure will be continued till EOF is reached.

To read the information about slow control parameters from the SCParameters.dat file and to write it into the data base the SCrecordswr.cc computer code has been developed. A positive integer number of parameters in the current record is read from the first line of SCParameters.dat file. Then the Value and the Name of parameters has to be read for every parameter in the record and written into corresponding container. The process stops when negative or zero number of parameters in the record is read.

The last application program named Resultswr.cc has been created to read the information about scan results and measurements of wires from the Results.dat file and put it into the data base. Positive integer number of result parameters is read first. The next line of Results.dat file consists of the Value and Name of result parameter, which are read and stored in the corresponding containers. After a cycle for result parameters is over, positive integer number of wires has to be read and the cycle for wires begins. Parameters x, y, z, dx, dy, dz, ny, nz for every wire are read from the input file and are written into the data base consequently.

4. Acknowledgements

We would like to thank N. Smirnov and RD Schaffer for very efficient consultations on using Objectivity/C++, S. Goldfarb and D. Lellouch, K. Safarik for fruitful discussions. We are grateful to C. Fabjan and F. Rohrbach for strong support of this activity.

5. Bibliography

[1] J. Berbier et al. X-ray Tomograph Prototype for MDT Quality Control Status of the X-QC project, ATLAS Internal Note MUON-NO-174 (1997)

[2] X_Tomo WWW reference address: http://wwwinfo.cern.ch/~xtomo

[3] ATLAS Muon Spectrometer Technical Design Report, CERN/LHCC/97-22 (1997)

[4] Using Objectivity/C++. Version 4, Release 4.0, 1996. Objectivity, Inc.

[5] E. Gschwendtner, F. Rohrbach, Y. Sedykh. Analysis and Results from Measurements on an X-ray Tomograph of Large Full-Scale MDT Prototypes, ATLAS Internal Note MUON-NO-175 (1997)