Manual for the extended ConditionsDB interface for ATLAS

Author: A.Amorim, N.Barros, D. Klose, J.Lima, C.Oliveira, L.Pedro
Date: May 7, 2003

Abstract

This document reports the evolution of the Conditions Database (`CondDB`) interface for the ATLAS experiment. A new `CondDB` with extended features have been developed during the last moths and this document is the first step that will allow its usage. The extended version of `CondDB` is based on the `MySQL` inplementation [1] and it is his natural sucessor. In this document will be presented the notes for the extended `CondDB - MySQL` implementation (0.3 version). It will provide a guide line to the interface, including a detailed description of the **major changes**, the necessary documentation to start using the **new features** and also an enumeration of **bug fixes**.

Contents

1 Introduction

This document will present the changes accomplished in the `CondDB` API for ATLAS. Although we are still trying to understand with the users what will be the final requirements[1], it was understood that a new version of the interface was needed. This new version will provide a set of bug fixes found, both from user reports and from more exhaustive tests performed on the implementation. To cope with the user requirements collected so far[3], we made very important changes in the implementation, mainly providing new tools and features. The main new feature that is included in this version of the API is the support for tiny objects which will allow the storage of `PVSS`-related data into the `CondDB`.

Other enhancements of the API were also implemented, such as a `Tcl/TK` wrapper, new examples, or the support for new compilers. These changes will be described in detail in the appropriate section.

This document is the first one that provides information about an **extended** `CondDB` API. At the time it was written this extention does not apply to any other implementation of the API (neither ORACLE nor Objectivity).

The document will be structured in 9 different sections:

Introduction - this one.

Bug fixes - a description of the bugs that were found and that should now be fixed.

Major changes - includes a description of the new database schema; the support for tiny objects, etc.

New tools - a description of new tools implemented that will help in the usage of `CondDB` API.

New examples - new source code examples that make usage of the new features are distributed.

Other changes - minor but also important ones.

To do - a list of things that should be done in the near future.

General informations - other informations that might be relevant

Acknoledgements -

[1] http://atlobk01.cern.ch/ConditionsDB/requirements/

2 Bug fixes

In this section we will present all the bugs found during testing and reported by users. All the bugs that were reported during the time between the release of version 0.2.6-b and this one are fixed.

Data storage stop when using multiple clients. When more than one client were used simultaneously storing data, the process stopped until only one client was using the database. An exception was generated and the process stooped until only one client was using the database. This happens because the API provides methods to ensure the database integrity and, with multiple clients, this was not possible in the previous version. In version 0.3 table locking mechanism is used during INSERT and UPDATE querys. The problem is fixed.

Error while storing and retrieving data. All the queries for storage and retrieving where performed in order to the localhost instead of the server passed from the init() method. This error was found while running remote tests and it's fixed in this version. All the necessary methods were re-written using the variable `srvname`.

find() generates an exception. The method `find()` threw exception when point was exactly the frontier between two objects. This bug was fixed by substituting `BETWEEN` by `>= AND <` in the appropriated queries.

Problems after `cvs co`.

- Missing of the `lib` directory causes an error in compilation after checking out from the CVS repository. Users needed to create this directory by their own. In this version, if the directory does not exist it will be created from the `Makefile`.
- After `cvs co` users needed to create a softlink in directory `ConditionsDB-MySQL/include/` in order to ensure that the compilation of `CondDB` library proceeds. This problem was solved by removing from the necessary source code files all references to directory `ConditionsDB` in `#include<xxxxx.h>` lines.

`db_id` error in table `folders_tbl`. In the previous version of `CondDB` interface, the field `db_id` was not properly filled. If it was a `folderRoot` or a `folderSet` the field was stored with the value `0`, otherwise it would be stored the value `1` and this was hard-coded. After the necessary corrections, the interface retrieves the proper value performing a query in the table `databases_tbl` giving the `dbname` and `srvname`. After retrieving the result from the query it stores the correct value in table `folders_tbl`.

Corrected function `getAllCondDBFolderBeneath()`. This function didn't make any distinction between folder and folderset. This bug is corrected.

3 Major changes

In this version of the API a lot of changes were introduced, some of them very deep, what causes the incompatibility between this version and the previous one. To cope with the new enhancements introduced, as well as for simplicity, a new databases schema was achieved. This means that databases from the previous versions of the API will not work with the 0.3 version (neither storage, nor retrieving data). Figure 1 shows schematically the database design of `CondDB` API including the relationship between the tables. In this version we also introduce the support for tiny objects. This means that using a proper manager connected to `PVSS` the API can now handle very granular objects coming from DCS.

3.1 New database schema

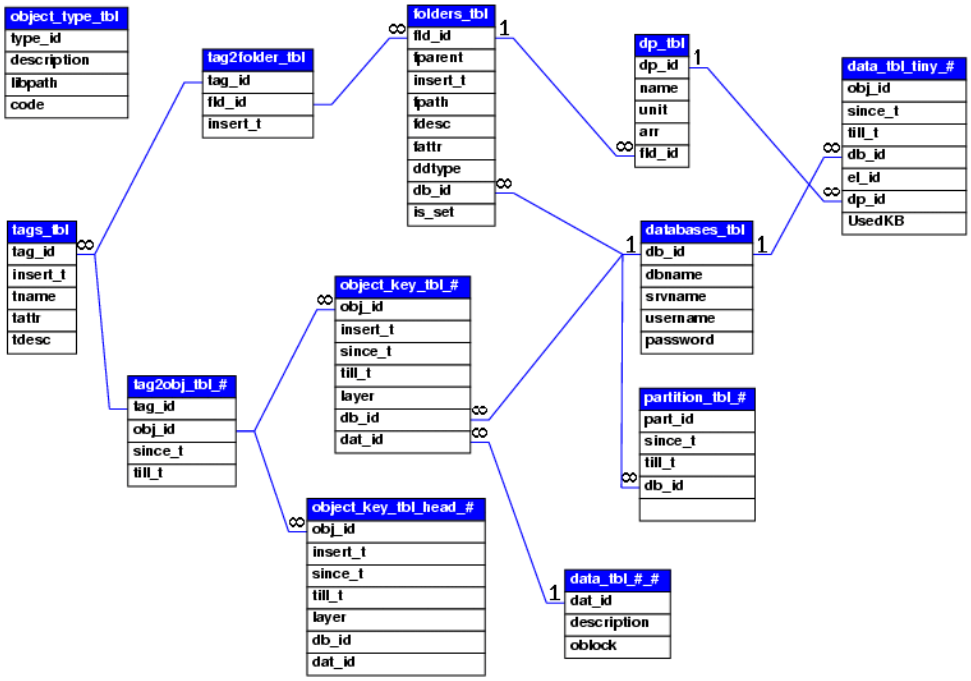

Figure 1: `CondDB` API database schema.

The new database schema was designed in order to allow extention of the API to support new features, also trying to think in future improvements that can be made (e.g. Database partitioning) as well as to make it more

simple than the previous one. Some of the changes in the schema were intruduced, also due to the results from performance tests[2] that we've been seting and implementing.

Here is the list of tables together with a small description of each one:

- `databases_tbl`

Field	Type	Null	Key	Default	Extra
db_id	int(11)		PRI	NULL	auto_increment
dbname	char(64)	YES		NULL	
srvname	char(64)	YES		NULL	
username	char(16)	YES		NULL	
password	char(16)	YES		NULL	

  In this table will be stored information about all the databases used to store conditions data. This table will be used in the condDB master servers in order to cope with scalability issues. One row will be added if a new slave server or database will be used.

- `folders_tbl`

Field	Type	Null	Key	Default	Extra
fld_id	int(11)		PRI	NULL	auto_increment
fparent	int(11)			0	
insert_t	timestamp(14)	YES		NULL	
fpath	varchar(255)				
fdesc	varchar(255)				
fattr	varchar(255)				
ddtype	int(11)	YES		0	
db_id	int(11)	YES		NULL	
is_set	tinyint(1)	YES		0	

  All the information about folders, folderSets and folderRoots will be stored in this table. The table includes the `db_id` field that will make the relationship between folders and the description of the database in which they will be stored.

- tag2folder_tbl

```
+----------+---------------+------+-----+---------+-------+
| Field    | Type          | Null | Key | Default | Extra |
+----------+---------------+------+-----+---------+-------+
| tag_id   | int(11)       |      |     | 0       |       |
| fld_id   | int(11)       |      |     | 0       |       |
| insert_t | timestamp(14) | YES  |     | NULL    |       |
+----------+---------------+------+-----+---------+-------+
```

  This is an auxiliary table used to create a many-to-many relationship between tables `tags_tbl` and `folders_tbl`.

- tags_tbl

```
+----------+---------------+------+-----+---------+----------------+
| Field    | Type          | Null | Key | Default | Extra          |
+----------+---------------+------+-----+---------+----------------+
| tag_id   | int(11)       |      | PRI | NULL    | auto_increment |
| insert_t | timestamp(14) | YES  |     | NULL    |                |
| tname    | varchar(64)   | YES  |     | NULL    |                |
| tattr    | varchar(64)   | YES  |     | NULL    |                |
| tdesc    | varchar(255)  | YES  |     | NULL    |                |
+----------+---------------+------+-----+---------+----------------+
```

  This table is used to store information about tags, including the insertion time, the tag name and the tag description.

- partition_tbl_#

```
+---------+------------+------+-----+---------+----------------+
| Field   | Type       | Null | Key | Default | Extra          |
+---------+------------+------+-----+---------+----------------+
| part_id | int(11)    |      | PRI | NULL    | auto_increment |
| since_t | bigint(20) | YES  | MUL | NULL    |                |
| till_t  | bigint(20) | YES  | MUL | NULL    |                |
| db_id   | int(11)    |      |     | 0       |                |
+---------+------------+------+-----+---------+----------------+
```

  This table will be used in the future to allow table partitioning. This feature is very important because we can ensure the database scalability. Once again the # character goes for the `id` of the corresponding folder. For each folder there will be a new partition_tbl. This table also contains the server coordinates because it uses the field `db_id`.

- `object_key_tbl_#`

Field	Type	Null	Key	Default	Extra
obj_id	int(11)		PRI	NULL	auto_increment
insert_t	timestamp(14)	YES		NULL	
since_t	bigint(20)	YES	MUL	NULL	
till_t	bigint(20)	YES	MUL	NULL	
layer	int(11)	YES		NULL	
db_id	int(11)	YES		NULL	
dat_id	int(11)	YES		NULL	

  In this table is stored all the information that allow the full description of each object. Each one of the fields is of extreme importance for the object qualification. It includes the time of insertion, the validity time, the reference to the description of the database and the version of the object. The # character represents the id of the folder in which the objects are assign. It also includes a field that points to the data it self (data_id).

- `object_key_tbl_head_#`

Field	Type	Null	Key	Default	Extra
obj_id	int(11)		PRI	NULL	auto_increment
insert_t	timestamp(14)	YES		NULL	
since_t	bigint(20)	YES		NULL	
till_t	bigint(20)	YES		NULL	
layer	int(11)	YES		NULL	
db_id	int(11)	YES		NULL	
dat_id	int(11)	YES		NULL	

  This table records the same information as the previous one with the difference that the objects are associated with the `head` tag. Description of objects which tag is different from `head` are stored in the previous table.

- `tag2obj_tbl_#`

Field	Type	Null	Key	Default	Extra
tag_id	int(11)			0	
obj_id	int(11)			0	

  Another table used to create a many-to-many relationship. This one creates a relationship between `tags_tbl` and, `object_key_tbl_#` and `object_key_tbl_head_#`.

- data_tbl_#_#

```
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| dat_id      | int(11)      |      | PRI | NULL    | auto_increment |
| description | varchar(255) | YES  |     | NULL    |                |
| oblock      | mediumblob   | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
```

This is the table where the real data is stored. The objects are stored in BLOBs (Binary Large Object)

- dp_tbl

```
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| dp_id  | int(11)      |      | PRI | NULL    | auto_increment |
| name   | varchar(255) |      |     |         |                |
| unit   | varchar(10)  | YES  |     | NULL    |                |
| arr    | int(11)      |      |     | 0       |                |
| fld_id | int(11)      | YES  |     | NULL    |                |
+--------+--------------+------+-----+---------+----------------+
```

Information regarding the description of PVSS datapoints are stored in this table, namely the datapoint name and the array size.

- data_tbl_tiny_#

```
+---------+------------+------+-----+---------+----------------+
| Field   | Type       | Null | Key | Default | Extra          |
+---------+------------+------+-----+---------+----------------+
| obj_id  | int(11)    |      | PRI | NULL    | auto_increment |
| since_t | bigint(20) | YES  |     | NULL    |                |
| till_t  | bigint(20) | YES  |     | NULL    |                |
| db_id   | int(11)    | YES  |     | NULL    |                |
| el_id   | int(11)    | YES  |     | NULL    |                |
| dp_id   | int(11)    | YES  |     | NULL    |                |
| Value   | _________  | YES  |     | NULL    |                |
+---------+------------+------+-----+---------+----------------+
```

Values coming from the DCS scada system that reflect the state of the detector are stored in this table. Because this type of data does not need neither versioning nor tagging mechanism, only the validity time is stored. The field dp_id is used as a reference to the datapoint that each element corresponds. The type of the field designated here by the name of Value will change depending on the type of what comes with the datapoint. # goes for id of the corresponding folder.

3.2 Support for tiny objects

As written in the previous section, the API provides, starting from version 0.3, features that allow storage and retrieving of tiny objects. These features match the requirement of some users that would like to store objects with simple structures but with a huge granularity. The best example of objects of this kind are the data coming from the DCS part of the detector. To deal with this type of objects the API was extended in order to handle the `PVSS` interface. New methods implemented:

Storage:

- `createCondDBFolderPvss (fullpathname, attributes, description, parents, dptype, dpname, unit, name, elem)`

  This function creates a set of tables (`partition_tbl_n`, `data_tbl_tiny_n`), the necessary, folders and makes an entry in the tables `folders_tbl` and `dp_tbl` for the given `dpname`. `fullpathname` is the full pathname of the folder. `dptype` is needed to define the type of field for the values (`INT`, `FLOAT`, etc). `dpname` is the datapoint name unit, the unit associated to the dp name is the name of the values column elem specifies how many elements an array has (if it is not an array elem = 0).

- `storeCondDBPvss (parentfolder, dpname, unit, values, since, name)`

  This function inserts a new value (or a set of new values in case of an array) in the `data_tbl_tiny_n` table, closing the interval of validity of the previous value. `parentfolder` is the name of the parentfolder dpname the datapoint name unit the actual unit associated to the dp values: the values to be stored (1 for normal dp, n for dyn dp) since, the actual time name, the name of the values column

Retrieval:

- `findCondDBPvss (string dpname, CondDBKey time, vector<string> value, string unit)`

  This function searches for the value(s) stored at time and stores the found value(s) in the vector<string> value and the associated unit in string unit. time can be in SimpleTime format which makes it easier to use this function. dpname, the datapoint name whose value we want. In the case of a dyn dp value will contain as many values as have been stored at time.

- `browseCondDBPvss (string dpname, CondDBKey since, CondDBKey till, vector<string> unit, vector < vector<string> > data, vector <SimpleTime> time)`

  This function searches for all the values contained in the time interval defined by since and till. It stores the result in vector< vector<string> > data, vector<string> unit and vector <CondDBKey> time, meaning the values found in data, associated unit in unit and the associated time in time. data

is a twodimensional vector where each row contains an entry for the respective time. This entry is a vector which contains as many values as the dp contained at the time (1 for normal dp, n for dyn dp). unit and time are onedimensional where each element corresponds to one time. This means:

data[n][m] returns he mth element of the nth row

time[n] returns the time for the nth row

unit[n] returns the unit for the nth row

data[n].size() returns the number of elements in row n

data.size(), time.size(), unit.size() returns the number of rows since and till can be in SimpleTime format dpname the name of the datapoint

4 New tools

4.1 Tcl/tk wrapper

4.1.1 General informations

The wrapper makes the ConditionsDB API available under TCL. The API is available as a binary module that can be easily loaded into a running interpreter. When properly installed, the following line will suffice to automaticaly load the ConditionsDB module.

```
package require conddb 1.0
```

For further details about the installation read the install item listed bellow.

The TCL API mimics the C++ API as much as possible. So, if you are already familiar with the C++ API it should not be difficult to start using the TCL API right away.

Appart from the sintactic differences, most API caracteristics were preserved. Even the object oriented nature of the API is present in the TCL version. The following facts can be taken as granted when learning the TCL's API.

- For every class, all method names are the same as the C++ couterpart;
- The argument order for each method is the same as the C++ counterpart;
- The return value for methods is the same in TCL and C++;
- When a reference or a pointer is passed as argument to a C++ method, The TCL's version takes a variable name;
- When an object is taken as argument or a pointer to it is returned from a C++ method, TCL takes or returns the name of the object.
- When a value (integer, float or string) is passed to or returned from a C++ method, The TCL's counterpart also expects or returns a value.
- The instances of the C++ API classes are all supported by the TCL API. They are represented by TCL procedures that are actually bound to a C++ object.
- The first argument for each object bound procedure is the method that one wishes to invoke. The remaining arguments are, y the order in which they appear, the corresponding arguments for the C++ method for that object.

- The exception to the above statement is the case of the factory classes methods. However, this is an exception even n the case of the C++ API as these methods are not bound to any object, they are pure static methods. On the TCL side these methods are defined inside a namespace with the name of the corresponding class.

An interesting feature provided with the TCL API that is not available in the C++ API is the performance measurement facility. This facility is controlled through to TCL global variables:
`conddb_perf_measurement`
`conddb_elapsed_time`
The first one holds a boolean value stating if the performance measurement is active (1) of inactive (0).
The second global variable holds a floating point representation for the time elapsed during the last API operation in seconds. It is important to evaluate this variable immediately after the operation you are interested in, since every subsequent operation will change its value.

4.1.2 How to install

Follow these steps when installing the CONDDB module for tcl.

1. Compile the ConditionsDB from the toplevel directory
   (a) Copy the tclconddb.so to the standard tcl modules directory in your system. This might vary from system to system. Contact you system administrator.
   (b) Alternatively, you may choose whatever directory you wish and append that directory path to the system variable `TCLLIBPATH`. Pay attention that, unlike other system vars like `PATH` or `LD_LIBRARY_PATH`, the path elements in `TCLLIBPATH` are not separated by colons ":" but by spaces " ".
2. Invoke a TCL shell (can be tclsh, wish, or any other TCL shell) and run the command `pkg_mkIndex` with the following arguments
   `pkg_mkIndex -verbose <conddb_module_dir_path> tclconddb.so`
   The ¡conddb_module_dir_path¿ stands for the directory where the tclconddb.so file resides (the one chosen in step 2). The path can be relative or absolute. The `-verbose` option is very importante since it lets you know if something goes wrong.
   On success, a file named `pkgIndex.tcl` will be created in ¡conddb_module_dir_path¿ which contains the command that will provide automatic loading when the package is required.

4.2 The PVSS Manager

4.2.1 General information

A PVSS manager was created to store PVSS values in the Conditions Database as tiny objects. This Manager works only in Linux since Windows is not yet supported by the Conditions Database.

The PVSS Manager for the conditions database connects to a PVSS system and reads the contents of the datapoint defined in the config file. This datapoint has to be of type dynamic string and is supposed to contain the datapoint names of the datapoints whose values we want to store online.

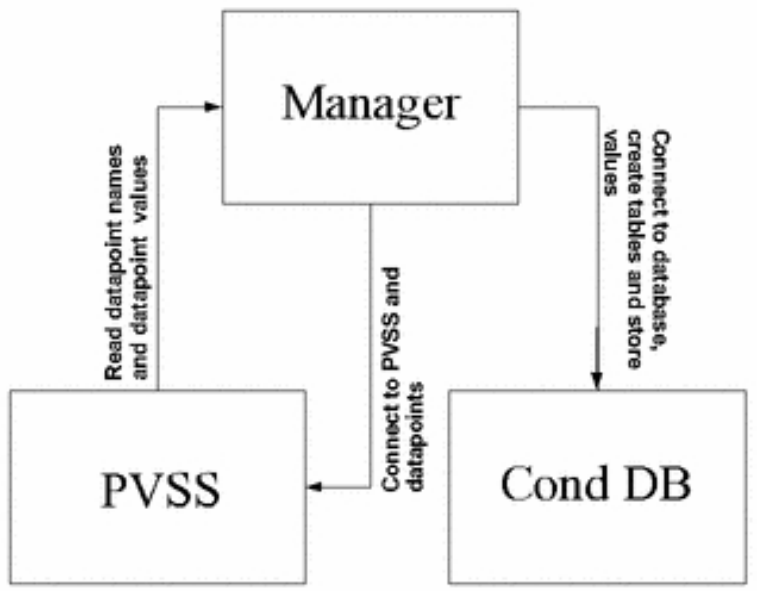

Figure 2: Interface between PVSS and CondDB.

All the necessary configuration information is defined in the config file.

The PVSS manager reads the list of datapoint names and connects to each one. At the time of the connect, the manager checks if the database exists and if the folders exist (one folder per datapoint). If they don't exist, the manager creates them, making usage of the CondDB API. After that the initial value of each datapoint is stored. Every time a value changes the storing function is called. Figure 2 shows a very general schema on how the Manager interfaces PVSS and the CondDB.

At the moment single value datapoints and dynamic datapoints are supported, their types being: float, int, bool, string, char and time. Structures are not supported in this version.

The functions used and the table structure are described in section 3 of this document.

Files

ApiManRunTime.zip CDBManager.hxx CDBResources.hxx addVerInfo.cxx CDBMain.cxx CDBManager.zip Makefile.api CDBManager.cxx CDBResources.cxx README

4.2.2 How to install

`CDBManager.zip` contains the executable version of the Manager for Linux. The source code files are also available including the Makefile.

To get the manager running you have to do the following:

1. Compile the Conditions Database API
2. unzip the manager CDBManager.zip
3. unzip APIManRunTime.zip
4. change config file in APIManRunTime/config (see README)
5. define environment variables (see README)
6. execute the Manager

4.3 Backup tool

There's also available a tool to backup `CondDB` databases. In directory `tools` there is a `perl` script that makes the backup automatically. The syntax to use this script is the following:

```
cond_db_backup.pl database_name backup_location mysql_user [mysql_host]
```

If you don't specify a `mysql_host` by default it will try to connect to localhost machine. The `mysql_passwd` will be asked while trying to connect to the server. Note that if `backup_location` starts by `/castor/cern.ch` the backup file is then moved to the specified location at CERN's CASTOR.

To use this aplication, you'll need to have installed in your system the `MySQL` aplication named `mysqldump`.

5 New examples

A set of new examples have been developed. It is intended to exemplify the usage of the new features available for the `CondDB` API. In this section some examples will be presented along with a short description. These examples, as well as others, can be found in directory `CondDB-MySQL-extended-xx-xx-xx/implementationMySQL/examples/` of the implementation.

5.1 browsepvss.cxx

Example application to illustrate the use of the function `browseCondDBPvss (const string& dpname, CondDBKey since, CondDBKey till, vector<string>& unit, vector < vector<string> >& data, vector <SimpleTime>& time)`

It returns all the values found for the given time interval, the corresponding unit and the since time of each value. In the case of an ordinary dp there is only one column (second index=0) for the vector< vector<string> data. In the case of a dyn dp there are as many columns to each row as there are values for each time.

```
#include "ICondDBMgr.h"
#include "CondDBMySQLMgrFactory.h"

#include <string>
#include <iostream>
#include <vector>

using namespace std;

int main ( int argc, char* argv[] )
{
  try {

    ICondDBMgr* condDBmgr = CondDBMySQLMgrFactory::createCondDBMgr();
    condDBmgr->init();

    condDBmgr->startRead();
    condDBmgr->openDatabase();
    condDBmgr->commit();

    ICondDBDataAccess* condDataAccess = condDBmgr->getCondDBDataAccess();

    condDBmgr->startRead();

 // start browse
    vector< vector<string> > values;
    string dpname;
    vector <string> unit;
```

```
    vector <SimpleTime> time;

    SimpleTime since, till;
    since = SimpleTime(2003,3,13,16,45,0);// start time
    till = SimpleTime(2003,4,13,16,50,0); // end time
    cout << "Start time: " << since << endl;
    cout << "End time  : " << till << endl << endl;

 // Single value:

     cout << "Single value data point" << endl << endl;
     dpname = "System1:_MemoryCheck.UsedKB:_online.._value";
     condDataAccess->browseCondDBPvss(dpname, since, till, unit, values, time);

     if (values.size() > 0){
       cout << "Value(s) found for " << dpname << ": " << endl;
       cout << "Value" << "\t\t" << "Unit" << endl;
       for (unsigned int i=0; i<values.size(); i++)
          cout << values[i][0] << "\t\t" << unit[i] << "\t" << time[i]<< endl;

     }
     else
        cout << "No value found" << endl << endl;
// Array:
     cout << endl << endl << "Dynamic data point" << endl << endl;
     values.clear();

     dpname = "System1:test.dynfloat:_online.._value";
     condDataAccess->browseCondDBPvss(dpname, since, till, unit, values, time);

     if (values.size()>0){
        cout << "Data found for " << dpname << endl << endl;
        cout << "Value" << "\t\t" << "Unit" << endl;
        for (unsigned int i=0; i<values.size(); i++){
           for (unsigned int j=0; j<values[i].size();j++)
              cout << values[i][j] << "\t\t" << unit[i] << "\t"
                   << time[i] << endl;
           cout << endl;
         }
     }
     else
        cout << "No value found" << endl;

// end browse

     condDBmgr->commit();
     CondDBMySQLMgrFactory::destroyCondDBMgr( condDBmgr );
     return 0; // return success
 }
```

```
 catch (CondDBException &e){
    cerr << "*** ConditionsDB exception caught: " << e.getMessage() << "\n"
         << "***   error code: " << e.getErrorCode() << endl;
    return 1;  // return failure
  }
}
```

5.2 findpvss.cxx

Example application to illustrate the use of the function `findCondDBPvss (string dpname, CondDBKey time, vector<string> value, string unit)`.

Searches the value of the given dp for the given time and returns de value into the vector<string>value and the unit into string unit. In the case of an ordinary dp the value will be unique and therefore vector<string> will contain exactly one entry. In the case of a dyn dp the value found will be an array therefore vector<string> will contain as many entries as values that have been stored at the given time.

```
#include "ICondDBMgr.h"
#include "CondDBMySQLMgrFactory.h"

#include <string>
#include <iostream>
#include <vector>

using namespace std;

int main ( int argc, char* argv[] )
{
    try {

        ICondDBMgr* condDBmgr = CondDBMySQLMgrFactory::createCondDBMgr();
        condDBmgr->init();
        condDBmgr->startRead();
        condDBmgr->openDatabase();
        condDBmgr->commit();

        ICondDBDataAccess* condDataAccess = condDBmgr->getCondDBDataAccess();

        condDBmgr->startRead();

// start find

        vector<string> value;
        string dpname,unit;

        SimpleTime time = SimpleTime(2003,4,15,16,45,0);// search time
        cout << "time: " << time << endl << endl;

// Single value:
```

```
        cout << "Single value data point" << endl << endl;
        dpname = "System1:_MemoryCheck.UsedKB:_online.._value";

        condDataAccess->findCondDBPvss(dpname, time, value, unit);

        if (value.size() > 0)
            cout << "Value found for " << dpname << ": " << value[0]
                 << " " << unit << endl;
        else
            cout << "No value found for time " << time << endl << endl;
// Array:
        cout << endl << endl << "Dynamic data point" << endl << endl;
        value.clear();

        dpname = "System1:test.dynfloat:_online.._value";
        condDataAccess->findCondDBPvss(dpname, time, value, unit);

        if (value.size()>0){
            cout << "Data found for " << dpname << endl << endl;
            cout << "Value" << "\t\t" << "Unit" << endl;
            for (unsigned int i=0; i<value.size(); i++)
                cout << value[i] << "\t\t" << unit << endl;
        }
        else
            cout << "No value found for time " << time << endl;

// end find

        condDBmgr->commit();
        CondDBMySQLMgrFactory::destroyCondDBMgr( condDBmgr );
        return 0; // return success
    }
    catch (CondDBException &e){
        cerr << "*** ConditionsDB exception caught: " << e.getMessage() << "\n"
             << "***   error code: " << e.getErrorCode() << endl;
        return 1;  // return failure
    }
}
```

6 Other changes

6.1 GCC 3.xx support

After some code clean up and some re-implementation ConditionsDB `MySQL` implementation it compiles and works fine with GCC compilers with versions > than 3.0. Here is a list of things that were changed:

- using include file <sstream> instead of <strstream>
- using ostringstream instead of strstream
- added using namespace std to some implementation files - mainly the examples source code
- added #include <iostream> to some source examples in which it was missing

6.2 TIMETESTS flag

It was added a **Macro** to perform time tests while querying. Use the flag -DTIMETESTS in compilation and then set up the conditionsDB debug environment variable to export COND_DB_DEBUG=all to allow storage of the time spent in each query to be written in log file. This time is written in microseconds. Please note that this procedure will leave to a loss of performance of the API.

7 To do...

The CondDB interface is far from being complete. A lot other features that are not yet available are desired from the users. To deal with this we will continue to try to understand the user requirements and then develop and include them in a future version. Some of these new features have been already isolated. This is a list of features that should be included in the next version.

7.1 Graphical user interface

A graphical user interface (GUI) it's one of the next steps for this API. We're still evaluating technologies to ensure that a proper solutions is chosed. Athought already exists a GUI based on Tcl/TK, it's oriented for developers and for debug purposes. The condDB GUI should provide the functionalities of displaying the relevant information stored in the database and should also be able to display the data related with the objects whenever this could be possible.

7.2 Support for other types of objects

A set of other types of objects should be supported by the API in one of the next versions. The ideas presented below are still in an immature stage and it's up to the users to help to understand their needs.

- XML The capability of storing XML objects as well as the feature of being able to serialise and deserialise them should be implemented. XML objects are possible to store for the current implementation, but the API doesn't know anything about it's schema. A more sofisticated mechanism is desired by the users and should be provided in the future.
- ROOT objects Also, the capability of storing ROOT objects can be one of the features to implemented in the future. This will allow the API to be able to understand ROOT objects and use them.

7.3 Other platforms support

In the next version of the CondDB API Windows platform should be provided. This will allow users that are developing applications in Windows operating systems to use the CondDB facilities to store data in the database. This could be very interesting in particular for CERN users that are using the PVSS scada system.

Other platforms should also be supported if this becomes a user requirement.

7.4 New browsing functions

New browsing functions should be implemented in order to make it easy to select from a set of objects. A function that finds objects from a since_t to a till_t time is already desirable by the users and should be available in the next version.

7.5 New tools

A set of new tools should also be available in a future release of `CondDB` API. In this section it can be included tools to import `CondDB` databses that are stored for backup.

8 General informations

Some general and practical informations will be described in this section.

8.1 How to download

Please refere to the web page https://savannah.cern.ch/projects/conddb-mysql/ to get all the necessary links for downloading the different versions. There you'll find all the informations concerning the `CondDB` API subject including the link to the project home page: http://kdataserv.fis.fc.ul.pt/ATLAS/.

If you want to download the development version please follow the instructions below. You can also find this information in the project web page mentioned above.

Download ConditionsDB via CVS repository

- Define the environment variables:
  `export CVSROOT=:ext:conddb@kdataserv.fis.fc.ul.pt:/usr/local/cvsroot`
  `export CVS_RSH=ssh`
  or `setenv CVSROOT :ext:conddb@kdataserv.fis.fc.ul.pt:/usr/local/cvsroot`
  `setenv CVS_RSH ssh`
  Depending on what shell are you using.
- Execute the command:
  `cvs co CondDB-MySQL-extended`
- passord = `conditions`

8.2 Compiling

The `MySQL` version of `CondDB` should compile in any UNIX platform. If you find a problem compiling the library, please inform us as soon as you can.

If you've downloaded the .tgz file from the web page, start by unpacking the distribution in youDirectory:

tar zxf CondDB-MySQL-extended-xx.xx.xx.tgz

You'll get a directory scructure very similar to this one:

```
CondDB-MySQL-extended-xx-xx-xx/
|-- docs-MySQL
|-- implementationMySQL
|   |-- examples
|   |   |-- PVSSManager
|   |   |-- deprecated
|   |   `-- future
|   |-- include
|   |-- lib
|   |-- src
|   |-- tcl
|   `-- tools
`-- include
```

Change to the directory youDirectory/CondDB-MySQL-extended-xx-xx-xx/implementationMySQL. Edit the toplevel of the `Makefile` to match to your own settings. The `Makefile` is heavily commented, so you just need to follow the instructions. In the Makefile, you can set the default INIT_STRING.

Finnaly, type: make depend (skeep this step if you're using GCC >=3.0
make

If you want to compile the `Tcl/TK` wraper type:
make tcl
Do not try to use the make tcl command before you compile the `CondDB` library.

8.2.1 Requirements

To compile the `CondDB` API make sure that:

- GCC version 2.95.3 or above with C++ support
- `MySQL` development packages (header files and `MySQL` client library) version 3.23.41 or above
- in order to compile the `Tcl/TK` wraper you need the `Tcl/TK` developement package
- to run the `Tcl/TK` browser or the `Tcl/TK` examples you need a tcl shell installed

8.3 Installing

The installation settings for the API are also defined in the toplevel of the `Makefile`. Please change the line of the TARGETDIR = /usr/local/lib if you want to customize your installation path. Please check if you have the necessary privileges and the type:

make install

This will install the library in the defined path. Run the `ldconfig` command. If your directory of installation is not a standard one defined for libraries, you'll have to define the environment variable `LD_LIBRARY_PATH`. Type:

export LD_LIBRARY_PATH=your_installation_path
or
setenv CVSROOT your_installation_path

8.4 Reporting bugs

For the `CondDB` API package we're using the `LCG Software development portal`[2]. You can find the web page for the CondDB project in:

https://savannah.cern.ch/projects/conddb-mysql/

[2]https://savannah.cern.ch/

There you can find an easy to use tool for bug repporting. Please use it to repport any kind of problem that you find wile using `CondDB - MySQL`. The team of developers will be automatically notified every time that a bug is reported.

8.5 Running the examples

The main `Makefile` also builds a set of example programs that are distributed with the source code of the package. To use any of theses example programs change to the directory yourDirectory/CondDB-MySQL-DCS/implementationMySQL/examples and run the tests you want. Note that some of the example programs depend on others, and other may fail if executed twice. The list of examples that are provided is the following:

- **basicSession** - Creates the database basic structure.
- **createFolderx** - Creates a folder and tests it.
- **storeData** - Stores a simple object.
- **readData** - Reads bask the object.
- **genericObjectStore** - Stores a vector object.
- **genericObjectRead** - Reads the vector.
- **storeDatax** - Stores multiple objects (up to 1 milion).
- **readDatax** - Reads multiple objects.
- **storeDatay** - Stores multiple objects with diferent intervals of validity.
- **readDatay** - Reads back the objects created by storeDatay.
- **createTags** - Creates some tags.
- **testTags** - Tag objects.

For more information on how to create you own objects please refer to[1] section 10.4.

8.6 The log file

If you want to know what is hapning behind the API, you can set an environment variable (`COND_DB_DEBUG`) that will log in a file diferent informations deppending of the value that takes. Use the comand export COND_DB_DEBUG = verbosityLevel or setenv COND_DB_DEBUG verbosityLevel. `verbosityLevel` can take the following values:

- **none**: no message is printed (not to be used when defining a message)
- **prof**: only those messages are printed which do not disturb profiling measurements
- **user**: only those messages are printed which are of "general" interest
- **devl**: also those messages are printed which are only of interest for the package developer

- **more**: more paranoid messages are printed
- **all**: any debug message will be printed. You can also get the time that takes to perform each query if you compile with the -DTIMETESTS option.

9 Acknolegments

Thank you for those who have been involved in the last months with this project. Also thanks for their sugestions and pragmatism that helped us to develop this new version of the `CondDB`.

References

[1] J. Lima, A MySQL based ConditionsDB interface,
http://kdataserv.fis.fc.ul.pt/ATLAS/, November, 2002.

[2] A. Amorim et all, Evaluation of the Relational implementation of Conditions Database
http://kdataserv.fis.fc.ul.pt/ATLAS/, February, 2003.

[3] A. Amorim et all, Requirements on the Conditions Database interface to TDAQ
http://atlobk01.cern.ch/ConditionsDB/requirements/