DCS Database for the ALICE TRD

T. Dietel, K. Oyama

October 1, 2006

1 Overview

Information about the ALICE TRD is stored in two databases: gateDB[1] and wingDB[2]. The gateDB is located in Heidelberg and tracks the integration of the TRD modules. It contains the full history of the location and status of all components, including test results and the possibility to attach plain-text comments and files of arbitrary type to each component.

Operation of the TRD requires a second database, called wingDB, that will at CERN be part of the Configuration DB of the DCS. Other sites like Heidelberg, Münster and Frankfurt will also have a wingDB to operate parts of the TRD. The table structure of the wingDB will be described in section 2.

The retrieval of configuration files from the database is done by the Command Coder, which is part of the InterCom Layer. The basic concepts and database queries are explained in section 3.

Each wingDB will have to be synchronized with the gateDB at some time. For the DCS Configuration DB at CERN, it will be sufficient to load the data from the gateDB once for every Supermodule. Afterwards, no further access to the gateDB is necessary. The other sites will require more frequent updates, because the integration will be recorded in the gateDB, and all changes will have to be propagated to the local wingDB before operation of the newly installed components is possible.

2 wingDB

The data required for the operation of the TRD can be grouped into four classes: information about the components and their locations in the TRD, standard configuration files to be downloaded to the detector, software patches to account for broken hardware or special configuration requirements, and calibration information like gain correction tables. For each of these four classes, the data is contained in one or more tables in the wingDB.

2.1 Detector Components

The DB structure for the components follows the physical structure of the TRD: the TRD is divided into 18 Supermodules that contain 5 stacks of 6 read-out chambers (ROCs) each. Each ROC is instrumented with 6 or 8 read-out boards (ROBs), a DCS Board for slow control purposes and 2 optical read-out interfaces (ORIs). The ROBs are equipped with 17 or 18 multi-chip modules (MCMs). The wingDB contains tables for Supermodules (listing 2.1), ROCs (listing 2.2) and ROBs (listing 2.3). DCS boards and ORI's are not represented by separate tables in the wingDB, but only as fields in the table for ROCs. It is foreseen to also include a table for MCMs or to include the list of mounted MCMs in the ROB table, but this has not yet been implemented. The information in this section is referenced by other sections of the wingDB to identify the parts of the detector a patch or calibration applies to.

For each component, the location is given by a foreign key referencing the superordinate component and the location therein, e.g. location of a ROC is given by the Supermodule ID and the stack and layer coordinates within the SM.

The data volume is dominated by the smallest and most frequent componentent, i.e. in the current setup by 4 000-5 000 ROBs. If a separate MCM table is included, there will be about 70 000 entries in that table.

2.2 Configuration Files

From a users point of view, the TRD is controlled by the means of tags. A tag is an integer number that identifies an action to be performed by the detector, usually a configuration or a self-test. In the database, each tag is mapped to a configuration that is identified by a name and a revi-

[1]**G**lobal **A**LICE **T**RD **E**lectronics **D**ata**b**ase
[2]**W**ingDB **i**s **n**ot **g**ate**DB**

Listing 2.1 Supermodule table: a SM is identified by its serial number (`sm_id`), and can sit in one of the 18 positions in the space-frame (`sm_slot`).

```
CREATE TABLE sm(
    sm_id    NUMBER(2) PRIMARY KEY,
    sm_slot  NUMBER
)
```

Listing 2.2 Read-Out Chamber table: a ROC is identified by its type (`roc_type`) and serial number (`roc_serial`). Its location is given by the SM (`sm_id`) and position inside the SM (`sm_stack`,`sm_layer`). The DCS board and ORIs mounted on this ROC are also listed here (`dcs_id`, `ori_a_id`, `ori_b_id`), while the information about ROBs on a ROC is held in the table "rob".

```
CREATE TABLE roc(
    roc_type   CHAR(4),
    roc_serial NUMBER,

    sm_id      NUMBER,
    sm_stack   NUMBER,
    sm_layer   NUMBER,
    dcs_id     NUMBER,
    ori_a_id   NUMBER,
    ori_b_id   NUMBER,

    PRIMARY KEY (roc_type,roc_serial),
    FOREIGN KEY (sm_id)
        REFERENCES sm(sm_id),
    UNIQUE (dcs_id),
    UNIQUE (sm_id,sm_stack,sm_layer)
)
```

sion (usually the release number in the Subversion repository). The configuration determines a set of scripts that are executed in a defined order. Each script contains a sequence of commands that are the atomic unit of the FEE configuration.

The TRD front-end electronics is configured via the proprietary SCSN bus, that transmits packages containing one configuration command at a time, consisting of a destination designating either a single or all MCMs, a 5-bit command, a 16-bit address and 32 bits of data. Commands can for example be a write to or a read from an adress in memory, sleep commands etc. Address and data are not used for all commands.

It was a natural choice to use the contents of these packages as the basic structure for the configuration of the TRD. Listing 2.4 shows the representation of this command in the database.

Listing 2.3 Read-Out Board table: a ROB is identified by its serial number (`rob_id`), the type defines the positions on a ROC where this ROB can be mounted. The location of a ROB is given by the ROC (`roc_type`,`roc_serial`) and the position on the ROC (`roc_pos`).

```
CREATE TABLE rob(
    rob_id  NUMBER,

    type    CHAR(2),

    roc_type   CHAR(4),
    roc_serial NUMBER,
    roc_pos    NUMBER,

    PRIMARY KEY (rob_id),
    FOREIGN KEY (roc_type,roc_serial)
        REFERENCES roc,

    UNIQUE(roc_type,roc_serial,roc_pos)
)
```

Apart from command, destination, address and data it also holds a reference to a script and a sequence number within the script to model the ordered 1:n relation between `cfdat_command` and `cfdat_script`.

Scripts are building blocks for configurations, containing groups of related commands. The table `cfdat_script` (listing 2.5) contains just the name and the release that identify a script and an artificial primary key. To allow for reuse of the building blocks, each script can be used in several configurations, requiring an ordered n:m relation that is implemented in the table `cfdat_config_scripts` (listing 2.6), linking the scripts with entries in the table `cfdat_config` (listing 2.7) that holds the name and release of every configuration.

The translation between tags from the user interface and a configuration is done with the table `cfdat_tag` (listing 2.8) which assigns each integer tag an entry from `cfdat_config`.

The data is dominated by the configuration commands, all other tables will hold only small amounts of data. For one full configuration, a few thousand commands are needed, and there will be on the order of ten full configurations for one set of configuration files, that corresponds to one upload of all files from one revision of the subversion repository. Over the lifetime of the detector, we expect to accumulate several hundred sets of configurations, so that we would need 1—10 million

Listing 2.4 Table for configuration commands. Each command sent to the FERO electronics is represented by one entry in this table. The fields cmd, dest, addr and data are directly sent over the SCSN bus, while the reference to a script and the sequence number are used for organization of the commands. A trigger is supplied to automatically assign a sequence number.

```
CREATE TABLE cfdat_command (
        PRIMARY KEY (script_id, seq),
        script_id       NUMBER(8)
            NOT NULL
            REFERENCES cfdat_script,

        seq             NUMBER(6),
        cmd             NUMBER(2),
        dest            NUMBER(4),
        addr            NUMBER(6),
        data            NUMBER(10)
);

CREATE SEQUENCE cfdat_config_id_seq
   INCREMENT BY 1 START WITH 1 CACHE 2;

CREATE TRIGGER trigger_config_id_pk
   BEFORE INSERT ON cfdat_config
   REFERENCING NEW AS NEW OLD AS OLD
   FOR EACH ROW
Begin
   SELECT cfdat_config_id_seq.nextval
      INTO :NEW.cfg_id from DUAL;
End;
```

entries in that table.

2.3 Patches

In some cases, faults in the electronics require special configuration of some parts of the detector, e.g. disabling of some MCMs or the use of spare data lines. The database holds information about all of these problems, with one table for each type of problem. Listings 2.9, 2.10 and 2.11 give some examples of these tables: usually, the faulty component is identified by a foreign key referencing one of the component tables, possibly some additional information about the fault and a validity period, which by default is long enough to contain the lifetime of the TRD (Dec 31, 2099).

It is expected to have very few patches per read-out chamber, so that there will be at most a few thousand entries in these tables.

Listing 2.5 Table for holding information about configurations scripts. Each script is identified by a name and a version number. The script_id is introduced as an artificial primary key, that is generated by a trigger.

```
CREATE TABLE cfdat_script (
    PRIMARY KEY (script_id),
    script_id   NUMBER(8) NOT NULL,
    name        VARCHAR(100) NOT NULL,
    svn_rel     NUMBER(6) NOT NULL
);

CREATE SEQUENCE cfdat_script_id_seq
   INCREMENT BY 1 START WITH 1 CACHE 2;

CREATE TRIGGER trigger_script_id_pk
   BEFORE INSERT ON cfdat_script
   REFERENCING NEW AS NEW OLD AS OLD
   FOR EACH ROW
Begin
   SELECT cfdat_script_id_seq.nextval
      INTO :NEW.script_id from DUAL;
End;
```

Listing 2.6 n:m relation of configurations and scripts. seq gives the order of scripts for a configuration.

```
CREATE TABLE cfdat_config_scripts (
    PRIMARY KEY (cfg_id,seq),
    cfg_id      NUMBER(10) NOT NULL
        REFERENCES cfdat_config,

    seq         NUMBER(6) NOT NULL,

    script_id   NUMBER(8) NOT NULL
        REFERENCES cfdat_script
);
```

2.4 Calibration Data

Th largest fraction of the data that will be required by the TRD are the gain settings for each of approximately 1.4 million channels. Each channel requires a 9 bit gain factor and 5 bit offset, the two values can be combined into a single 16-bit integer. Updates to the gain tables will be rare, with probably only a few changes per year.

Due to the amount of data it is necessary to put the gain parameters for several channels into one row in the database. It is foreseen to group the 336 channels of one ROB into one row, requiring 4104 rows to store the information for the full

Listing 2.7 Table for FEE configurations. Each configuration is identified by a name and a release, an artificial primary key is used as a reference in other tables.

```
CREATE TABLE cfdat_config (
    PRIMARY KEY (cfg_id),
    cfg_id      NUMBER(10) NOT NULL,
    name        VARCHAR(100) NOT NULL,
    svn_rel     NUMBER(6) NOT NULL
);
```

Listing 2.8 `cfdat_tag` translates between the user-accessible tags and configurations stored in the database.

```
CREATE TABLE cfdat_tag (
    PRIMARY KEY(tag),
    tag     NUMBER(10) NOT NULL,
    cfg_id  NUMBER(10) NOT NULL
        REFERENCES cfdat_config
);
```

TRD.

3 Command Coder

Configuration data for the TRD is created by the Command Coder (CoCo), that retrieves all relevant information from the configuration database and assembles one configuration data block per ROC of the TRD. The CoCo is compiled into the InterCom Layer and is executed for every configuration request for a ROC, i.e. 540 times to configure the full TRD at the beginning of a run.

To assemble the configuration file, the CoCo queries the four parts of the wingDB and combines the data into one configuration block, that is then sent to the feeservers running on the DCS boards. The configuration blocks are in binary format, which is defined by structs in the C language as shown in listing 3.1.

3.1 Configuration file format

The header of a configuration block contains general information about the block, information about the ROC to be configured and pointers to the base configuration data and — in the future — to patches and calibration data. The general section consists of a start marker which is always "CFDAT\0\0\0", the size of the header, the version of the file format, one dword to check the endianness of the structure, and the two parameters that the CoCo has been called with: target name and tag.

Information about the ROC that is contained in the header includes the serial number of the DCS board, the type and serial number of the ROC and the types and serial numbers of all ROBs on the ROC. It is also foreseen to also send information about all MCMs mounted on the ROBs, but the data structures are not yet filled.

The remainder of the header consists of offset and number of SCSN commands contained in the base configuration of the, the version number of this base configuration and a checksum, that is however not yet used. Similar contributions are planned for patches and calibration data.

Listing 2.9 Table: Network Interface Errors

```
CREATE TABLE err_rob_ni (
    rob_id  NUMBER NOT NULL
        REFERENCES rob,
    port    NUMBER,
    pattern NUMBER,

    start_time TIMESTAMP NOT NULL,
    end_time   TIMESTAMP
        DEFAULT TO_DATE('31-DEC-2099',
                        'DD-MON-YYYY')
        NOT NULL
);
```

Listing 2.10 Table: General MCM Errors, that require disabling of the MCM

```
CREATE TABLE err_mcm_all (
    rob_id  NUMBER NOT NULL
        REFERENCES rob,
    mcm_pos NUMBER,

    start_time TIMESTAMP NOT NULL,
    end_time   TIMESTAMP
        DEFAULT TO_DATE('31-DEC-2099',
                        'DD-MON-YYYY')
        NOT NULL
);
```

3.2 DB Queries

One of the possible bottlenecks during the configuration of the detectors are the queries to the DCS configuration database. In the case of the TRD, the CoCo is called 540 times, and it has to run several queries to gather the information for the

Listing 2.11 Table: MCM CPU Errors, that invalidate data from the MCM, but leave the network interface intact.

```
CREATE TABLE err_mcm_cpu (
    rob_id  NUMBER NOT NULL
        REFERENCES rob,
    mcm_pos NUMBER,

    start_time TIMESTAMP NOT NULL,
    end_time   TIMESTAMP
        DEFAULT TO_DATE('31-DEC-2099',
                        'DD-MON-YYYY')
        NOT NULL
);
```

configuration files. These queries are described in this section.

The first query simply determines ROC type and serial number and the ID of the DCS board of the. Then, the serial numbers of the ROBs on the chamber are determined. These queries return very little data and should not pose a problem. It is also inteded to write information about all MCM IDs on the ROC into the configuration block, which would require another query, that returns either 17 or 18 additional fields for each row in the ROB table, or an additional query to an MCM table would be necessary, returning 104 or 138 rows per ROC.

The query to the second part of the wingDB, the base configuration is more complex. The statement used to retrieve the base configuration is shown in listing 3.2. The query returns all commands that belong to the scripts for one configuration, the results have to be sorted in the order of scripts and commands, and the data has to be transferred to the CoCo. As in the current implementation this is done for each invocation of the CoCo, it is a potential bottleneck. If this turns out to be true, there are two possibilities to speed this up: by storing assembled script files as large objects in the DB instead of single commands, or by caching the result of the query in the CoCo, so that it can be reused in subsequent invocations. No performance measurements have so far been done to determine if this is necessary.

The queries for patches are not implemented, but as they will only return very few rows per ROC, the load due to this part of the configuration data will be minimal.

The largest part of the configuration data is contained in the gain calibration tables for all channels. To minimize the administative overhead, it

Listing 3.1 Format of configuration files sent from the Command Coder to the feeserver on the DCS boards. For a description see text.

```
struct cfdat_robinfo
{
  signed int  rob_id;
  signed int  rob_type;
  signed int  mcm_id[18];
};

struct cfdat_command
{
  unsigned       cmd  :  5;
  unsigned       dest : 11;
  unsigned short addr;
  signed int     data;
};

struct cfdat_header
{
  char           hd_marker[8];
  signed short   hd_size;
  signed short   cfdat_ver;
  unsigned int   endian_tag;
  unsigned char  target_name[20];
  signed int     tag;

  signed short   dcs_id;
  signed short   roc_type;
  signed short   roc_serial;
  cfdat_robinfo  robinfo[8];

  signed int     svn_ver;
  signed int     n_cmd;
  signed int     offset_cmd;
  signed int     checksum_cmd;

  signed int     checksum_hd;
};
```

is necessary to group gain parameters for several channels into one row. To determine the optimum row size for organizing the data, performance measurements were done with an Oracle 10g database running on an AMD Sempron 2800+ with 1GB of memory, with server and client running on the same machine. A total of 1M fields was inserted into the DB and retrieved from it again. The number of fields per row was varied from 20 to 1000, and the number of rows adjusted accordingly. Above about 100 fields per row, the retrieval time depended only weakly on the number of fields, below this number, the administrative

Listing 3.2 Query for configuration data

```
SELECT cmd,dest,addr,data,
       cfdat_config_scripts.script_id,
       cfdat_config_scripts.seq AS sseq,
       cfdat_command.seq AS cseq
FROM  cfdat_config_scripts,cfdat_command
WHERE cfdat_config_scripts.script_id=
      cfdat_command.script_id AND
      cfdat_config_scripts.cfg_id=
      ( SELECT cfg_id FROM cfdat_tag
        WHERE tag=REQUESTED_TAG)
ORDER BY cfdat_config_scripts.seq,
         cfdat_command.seq
```

overhead seemed to dominate the retrieval time, slowing down the query. The time to retrieve the full dataset in a single query was below 2s for large rows with at least 100 fields.