Using ODB in ALADIN

Kertész Sándor
kertesz.s@met.hu
Hungarian Meteorological Service

22th November 2002

Contents

1 Introduction

ODB (Observational DataBase) is a newly developed database system to store and handle all kind of observations. It allows fast data extraction by data query language ODB/SQL. ODB concept has been proposed due to the growing number of observations (satellite data mainly) used in the data assimilation. It is supposed to allow an easier management of the data, with a quicker access and a more flexible organisation. ODB is developing by Sami Saarinen at ECMWF.

2 Basic features of ODB

2.1 Data structures and types

ODB consists of uniquely named data structures called *tables*. Each table is made up of several uniquely named data entries called *columns*. The notation for a given column is: column_name@table_name. ODB provides several data types for columns:

- int, real4, real8: Unpacked types.
- pk5int, pk9real etc.: Packed types, packing for 2, 5 or 9 bits both for integer and real.
- string: 8 bytes of character data.
- YYYYMMDD and HHMMSS: Types to store date.
- @LINK: Type to ensure relation between tables.
- OBSCHAR_t, STATUS_t etc.: Compound data types that are used for bitfields (with maximum of 32 bit-members). The notation for a given member in a bitfield type column is: column_name.member_name.
- Arrays can be composed of any data type and an array with n element can be defined as column_name[n]. The notation for bitfield members of arrays of compound types is: column_name[n]→member_name.

Let's see for example how table *hdr* is defined in ODB:

```
CREATE TABLE hdr AS (
    seqno int, // OBSERVATION SEQUENCE
    obstype pk5int, // OBSERVATION TYPE
    obschar OBSCHAR_t, // OBSERVATION CHARACTERISTICS
    ...
    date YYYYMMDD, // OBS. DATE
    time HHMMSS, // OBS. EXACT TIME
    rdbflag REPORT_RDBFLAG_t, // REPORT'S FLAGS
    ....
    ident pk5int,  // Integer equivalentof STATION ID (or zero)
    lat real8,   // LATITUDE
    lon real8,  // LONGITUDE
    ...
```

This table describes report headers and its columns are the type (*obstype*), the place (*ident, lat, lon*), the time (*date, time*) etc. of the report. At the moment 23 tables are defined in ODB. Table 1 gives a brief overview of them:

table name	description	table name	description
desc	analysis date and time	**ssmi_body**	ssmi report body
ddrs	DDR headers	**atovs**	atovs
index	timeslots, sets etc.	**atovs_pred**	atovs predictors
hdr	report header	**rtovs**	rtovs
sat	satellite master table	**rtovs_pred**	rtovs predictors
reo3	reo3	**rtovs_slev**	rtovs surface level info
satob	satob	**rtovs_mlev**	rtovs model level info
scatt	scatterometer	**rtovs_body**	rtovs report body
scatt_body	scatt. report body	**body**	report body
ssmi	ssmi	**errstat**	error statistics
ssmi_slev	ssmi surface info	**update**	update values in 3dvar
ssmi_mlev	ssmi model level info		

Table 1: Data tables of ODB

2.2 Links

Data in ODB can be imagined as arrays of tables interconnected by *@LINK* type variables. Let's visualize this arrangement with the case of hdr and body tables. Each observation report contains a header and a body part. The header gives general information about the observation and the body contains the very observed values. While in ODB one report header is represented by one hdr table, each observed value in the report body has its own body table. These tables are organized into individual arrays, i.e. one array for hdr and another one for body. For a certain hdr table its *body@LINK* variable identifies the corresponding body tables. This variable has two integer components: the first one specifies the number of the corresponding tables and the second one specifies the position of the first corresponding table in the body table array. This solution implies that body tables that belong to the same observation report are strictly one after the other in the body table array (Figure 1).

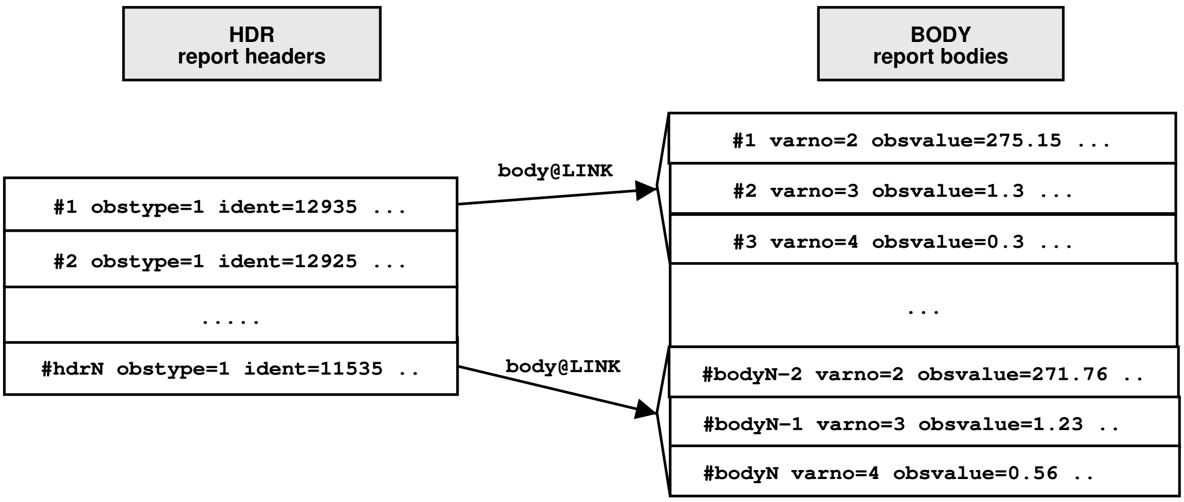

Figure 1: Relation between tables hdr and body (supposing we have only SYNOP reports with only T, u and v observations)

2.3 Database types

Tables can be used as building bricks to create more difficult database structures. A set of tables forms a *complete data definition* (this will be called *database type* in this document). At present several types exist. The most important ones for ALADIN are as follows:

- ECMA: The name stands for extended CMA, and it consists of all the tables needed by screening.
- ECMASCR: It is basically another name to an ECMA ODB. It could be used for special ODB related programs.
- CCMA: The name stands for compressed CMA, and it consists of all the tables needed by configuration e131. Usually CCMA is created from ECMA after the screening using only the active data.
- PREODB: This datatabse type is used for creation of ODB from the OBSOUL ascii file.

The graphs below show the hierarchy in the ECMA, ECMASCR and CCMA database types (the horizontal arrows refer to the @LINKs between the different tables):

```
ECMA/ECMASCR table hierarchy:          CCMA table hierarchy:

desc                                   desc
+--> ddrs                              +--> ddrs
+--> index                             +--> index
|    +--> hdr                          |    +--> hdr
|    |    +--> sat                     |    |    +--> sat
|    |    |    +--> satob              |    |    |    +--> atovs
|    |    |    +--> scatt              |    |    |    |    +--> atovs_pred
|    |    |    |    +---> scatt_body   |    |    |    +--> rtovs
|    |    |    +--> ssmi               |    |    |    |    +--> rtovs_pred
v    v    v    v    +---> ssmi_slev    v    v    v    v    +--> rtovs_slev
|    |    |    |    +---> ssmi_mlev    |    |    |    |    +--> rtovs_mlev
|    |    |    |    +---> ssmi_body    |    |    +--> body
|    |    |    +--> atovs              |    |    +--> errstat
|    |    |    |    +---> atovs_pred   |    |    +--> update
|    |    |    +--> rtovs
|    |    |    |    +---> rtovs_pred
|    |    |    |    +---> rtovs_slev
|    |    |    |    +---> rtovs_mlev
|    |    |    |    +---> rtovs_body
v    v    +--> body
|    |    +--> errstat
|    |    +--> update
```

2.4 Aligned and one-looper tables

ODB ensures scpecial relations between tables that are not connected with a @LINK type variable. These tables are enforced to be *aligned* or *one-looper*.

Aligned tables

Let's imagine ODB again as arrays of tables. If tables are enforced to be aligned then there must be a one to one correspondency between their elements. For example in the ECMA database body and errstat are aligned tables, thus the n-th body table and the n-th errstat table refer to the same observed value. The list below sets out all the aligned realtions in ODB:

1. body → (update, errstat, rtovs_body, scatt_body, ssmi_body)
2. rtovs → (rtovs_slev,rtovs_pred)
3. atovs → atovs_pred
4. ssmi → ssmi_slev

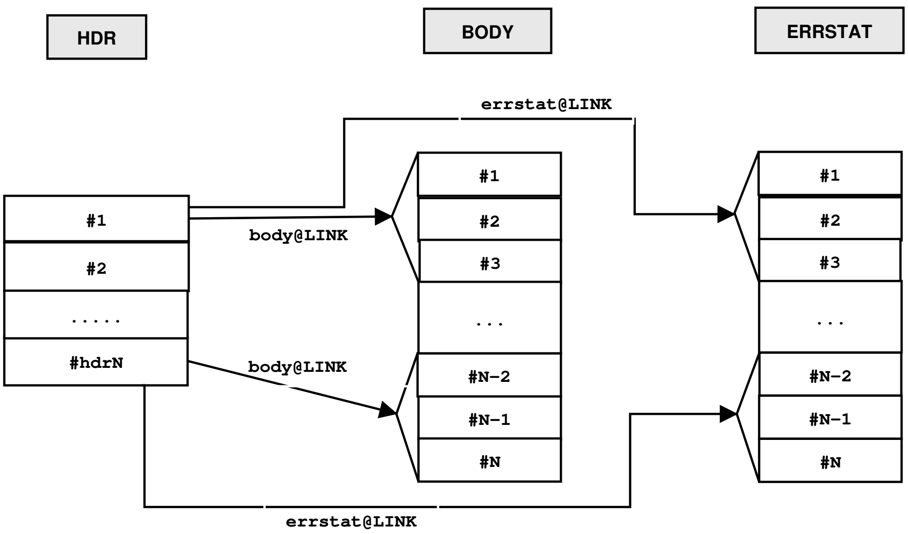

Figure 2: The hierarchy between aligned tables body and errstat

One-looper tables

If tables are enforced to be one-loopers then a master table acts like a middle-man between slave tables. In this case the length of @LINK (from the master table to a given slave table) must be 0 or 1. One example for one-looper tables are sat (master) and atovs, rtovs, ssmi, scatt, satob (slaves).

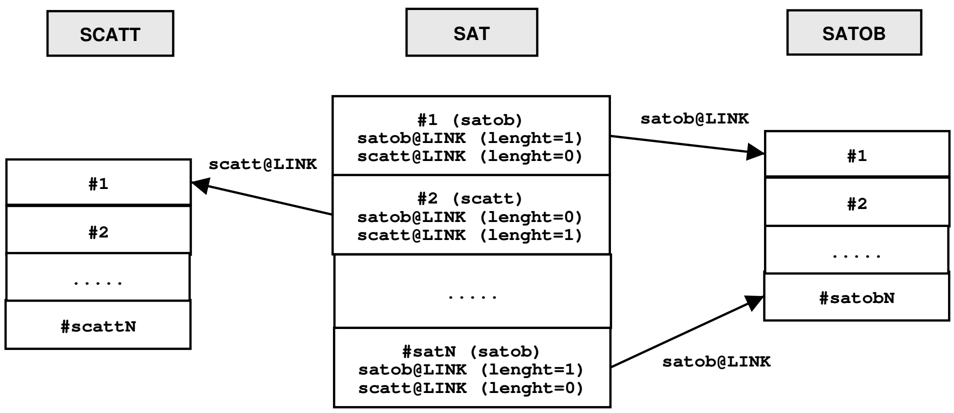

Figure 3: The hierarchy between one-looper tables satob and scatt (only @LINKs length of 1 are shown).

The list below sets out all the one-looper relations in ODB (master tables followed by the slaves):

1. index → hdr
2. hdr → sat
3. sat → (atovs,rtovs,rtovs,ssmi,scatt,satob)

2.5 Pools

Data partitioning for multi-processor runs can be realised inside ODB via the so-called *pools*. Each pool contains only a certain part of the database. Partitioning can be done with respect to latitude-bands, or time-slots or other ways. For multi-processor runs the number of pools must be greater or equal than the number of PEs.

2.6 What does ODB look like?

ODB data for a given analysis date is stored in a certain directory sturture. In the top directory there are two ascii files and one or more subdirectories. The top directory can have an arbitrary name and it can be anywhere in the filesystem. The two ascii files are as follows:

- *db_type*.dd (ECMA.dd etc.): It contains the default values for all the columns and variables (for parametrized queries) in the database.
- *db_type*.sch (ECMA.sch etc.): It contains the scheme to create the database.

The subdirectories represent the different pools. The name of such a directory is the number of the given pool (pool numbering starts at 1). A pool contains one file for each table of the given database. Basically these files store the data of the database.

3 Views

A subset of ODB data can be extracted by data query language ODB/SQL. This is done via the so-called *views*. A view can be defined as follows:

```
CREATE VIEW test1 AS
SELECT
  obstype,ident,
  lat,lon,
  varno
FROM hdr, body
```

Keywords of OSB/SQL were written in capital letters in the example. Obviously the *CREATE VIEW* ... line specifies the name of the view (*test1*). The *SELECT* block sets out the columns to be retrieved and the *FROM* keyword specifies the tables to be searched for. When more than one table is used we have to be careful because different tables can have columns of the same name. For example both table hdr and body have a column called status, so in this case we should use status@hdr and status@body. During the data retrieval the whole database graph that was presented in Chapter 2.3. is searched and the result is put into a matrix structure. The output format of the retrieval will be explained in detail in the next chapter. Now the most impotrant is that we can imagine it as a matrix. In our case this matrix will have 5 columns (corresponding to the 5 columns given via SELECT) and as many rows as the number of the observed values in the actual database, like this:

```
obstype@hdr ident@hdr   lat@hdr     lon@hdr    varno@body
  1           12925       0.25       0.11          2
  1           12925       0.25       0.11          3
  1           12925       0.25       0.11          4
  1           12935       0.26       0.23          2
  .........
```

A view can contain additional retrieval conditions, as well:

- With the *WHERE* keyword we can specifiy logical query conditions. The logical statements can contain the *OR* and *AND* logical operators. Statement grouping with the usage of parentheses is allowed.
- With the *ORDERBY* keyword we can specify the order of the output rows. The column given after this keyword will be used in the sorting. The sorting can be descending (*DESC*) or ascending (*ASC*). Sorting can be based on more columns, as well.

Now let's add new query options to view tets1. We will select only the active synop (obstype=1) and temp (obstype=5) reports and we will sort the output with respect to the station ID (ident) in a decending order. The corresponding view is:

```
CREATE VIEW test2 AS
SELECT
  obstype,ident,
  lat,lon,
  varno
FROM hdr, body
WHERE (obstype == 1 OR obstype == 5)
  AND status.active@body
ORDERBY ident DESC
```

Views should be placed into ascii files with a *.sql* extension. However, this format cannot be used directly in ODB. At first the SQL file has to be compiled with the ODB/SQL compliler into a C-language code. Then this C program is compiled with a standard C complier to get an object file that is suitable to be linked with the given ODB application. For each database type there are several predefined views that are located in the ODB system directories (see next chapter). These predefined views are already compiled and the .o files are put into the proper library. So the usage of these views does not require an extra compilation.

4 ODB files and directories

Source

At HMS the ODB system was built on AL15 (version export03). In what follows the directory of ODB source will be referred as *odb_scr*. *Odb_src* consists of several subdirectories, the most important one is *ddl*. This directory contains the basic ODB definitions, namely:

- all the different predefined views (in .sql files)
- file CMA.inc that contains the definition of the compound data types and the tables
- files mdi.inc, obstype.inc etc. that contain constant ODB variables (missing value, observation types, etc.)

For each database type there is an individual directory in *odb_ src* with the name of ddl.*db_ type* (ddl.ECMA etc.) In each ddl.*dbtype* directory there is a file called *db_ type*.dep that contains all the tables and views of which the given database is made up. These ddl.*db_ type* directories also contain all the necessary files from directory ddl to build the database (there are links to files in directory ddl). If we look into ddl.ECMA and ddl.ECMASCR then we can see that the major difference between ECMA and ECMASCR is that for the former database type more views are defined.

The installed system

The installation procedure will be create a new directory structure at the specified location. In what follows the home directory of the installed files will be referred as *odb_ sys*. The most important sub-directories are as follows:

- bin: ODB related executables. The most important one is odbtools.x (for detals see Chapter 6.).
- bintmp: The ODB/SQL compiler (obd98.x) is located here.
- *dbtype*: Each database type has its own directory containing fil*e dbtype*.IOASSIGN that is quete important for ODB related programs (for detals see Section 6.).
- include: Header files needed for ODB related compilation.
- module: Modules needed for ODB related compilation.

Odb_ sys contains also several libraries that are needed for the compilation of ODB related programs:

- lib*dbtype*.a: Each database type has its own library. Such a library contains all the compiled views of the given database.
- libodb.a: Modules and low level utils.
- libodbmain.a: Objects of executables.
- libodbport.a: Main interface to the model.
- libodbdummy.a: It contains only dummy routines. This library must be the last one in the library order at linking.

5 ODB in ARPEGE/IFS

5.1 Basic subroutines

ARPEGE/IFS can reach ODB through an intermediate layer of subroutines. The highest-level subroutines are *OPENDB*, *CLOSEDB*, *GETDB* and *PUTDB* (their sources can be found in *odb_ src*/cma2odb). The usage of ODB in the model starts with OPENDB which opens and initialize the database. After that subroutines working with observations have to call GETDB to extract data from ODB into a set of temporally allocated matrices. Subroutines then work with these matrices and in the end they call PUTDB to deallocate them and put the data back into ODB. Thus, observations are available only between these GETDB-PUTDB calls. Several GETDB-PUTDB calls can follow each other while in the end CLOSEDB closes the database and updates the modified data (as requested).

OPENDB

As it was mentioned before usage of ODB starts with OPENDB which is used to connect to the database (it is called in subroutine READOBA).

Arg.	Type	Description
1.	character	Type of ODB ('ECMA', 'CCMA' etc.)
2.	character	Status of ODB: • for new databases 'NEW' • for existing, but updateable databases 'OLD' • for existing, but read-only databases 'READONLY'
3.	integer array	Information array. Its first element gives back the number of the pools.
4.	integer	Number of the active elements in the previous array. It can be even zero.

Table 2: Arguments of subroutine OPENDB

GETDB

After the OPENDB call the database is accessible, but no data has been read yet. To extract data from ODB subroutine GETDB must be called. This subroutine reads data - the way of reading is specified in the arguments, mainly by the so-called context - into matrices defined in module *YOMDB*. These matrices can be as follows:

- MOBHDR: report header data (integer4)
- MOBODY: report body data (integer4)
- MOBSU: setup related data (integer4)
- ROBHDR: report header data (real8)
- ROBODY: report body data (real8)
- ROBSU: setup related data (real8)
- ROBDDR: DDR related data (real8)
- SATHDR: satellite specific header data (real8)
- SATBODY: satellite specific body data (real8)
- SATPRED, SATSLEV, SATMLEV: misc. satellite specific data

These matrices are allocated dynamically. The data extraction is not based on single views but on special group of views called contexts (see Chapter 5.2. for details on contexts). A context for each of its views determines the target matrix (one from the list above) and specifies those columns in the view which should be updated. While the first dimension of the matrices refers to the rows in the retrieval, the second dimension refers to the different columns in the given view. Each column name in ODB - except for the links - has an integer counterpart in YOMDB. The GETDB call sets these variables and they can be used as column indices for the matrices. For example, the corresponding value for lat@hdr in YOMDB is MDBLAT. So if a view containing lat@hdr is read into ROBHDR, then ROBHDR(*,MDBLAT) will refer to the retrieved lat@HDR values. The exact relation between database columns and YOMDB variables is defined *odb_src*/cma2odb/initmdb.F90.

Arg.	Type	Description
1.	character	Name of the context.
2.	integer	Version of the context .
3.	integer	Return value: number of the rows in the extracted data.
4.	integer arr.	Information array.
5.	integer	Number of the active elements in the previous array.
6.	real arr.	Information array. Not used now.
7.	integer	Number of active elements in the previous array (=0 now).
8.	integer	Number of the desired set or 0 if all the sets are needed.
9.	integer	Number of the desired timeslot or -1 if the value defined in the view should be used. In the latter case if the timeslot is not set in the view then all the timeslots will be loaded.
10.	integer	Number of the desired pool or -1 if the value defined in the view should be used. In the latter case if the pool is not set in the view then all the possible pools will be loaded.
11.	integer	The desired observation type (obstype@hdr) or -1 if the value defined in the view should be used. In the latter case if observation type is not set in the view then all the available observation types will be loaded.
12.	integer	The desired observation code type (obschar.codetype@hdr) or -1 if the value defined in the view should be used. In the latter case if observation code type is not set in the view then all the possible codetypes will be read.

Table 3: Arguments of subroutine GETDB

Another important variable in YOMDB is array *MLNKH2B*. This array - if it is ensured in the given context - corresponds to the link between table hdr and table body. Supposing that report headers are read to MOBHDR and report bodies to MOBODY, then MOBHDR(n,*) will contain the header of the n-th report, while the observed values (etc.) connected to this report will be in MOBODY(i,*) where i=MLNKH2B(n),...,MLNKH2B(n+1)-1. The piece of code below from subroutine VERCO illustrates this:

```
!2. LOOP OVER THE OBS HEADERS IN THE GIVEN SET
-----------------------------------------------
OBS_LOOP: DO JOBS = 1, MLNSET(KSET)
  ...
  ...
  !3.0   FIRST LOOP OVER THE REPORT
  FIRST_LOOP: DO JNLV = MLNKH2B(JOBS),MLNKH2B(JOBS+1)-1
    ILEV = JNLV
    !3.1   VARIABLE AND THE MAXIMUM FLAGS
    ICMVNM = MOBODY(ILEV,MDBVNM)
    ...
  ENDDO FIRST_LOOP
  ...
ENDDO OBS_LOOP
```

PUTDB

After finishing the work with the observations in the given subroutine PUTDB must be called. This subroutine deallocates the matrices and puts back the modified data into ODB. Its arguments are as follows:

Arg.	Type	Description
1.	character	Name of the context.
2.	integer	Version of the context.
3.	integer	Return value.
4.	integer arr.	Information array. Not used now.
5.	integer	Number of the active elements in the previous array (=0 now).
6.	real arr.	Information array. Not used now.
7.	integer	Number of the active elements in the previous array (=0 now).

Table 4: Arguments of subroutine PUTDB

CLOSEDB

Finally, when all the work with observations has already done CLOSEDB deconnects from the database. It is very important to set its argument carefully. Its argument is as follows:

Arg.	Type	Description
1.	logical	If it is .TRUE. then those columns that are marked for update in the context will be saved from memory into the database files. Otherwise no modification will be put into ODB!!

Table 5: Arguments of subroutine CLOSEDB

5.2 Contexts

Contexts contain most of the information needed for GETDB to perform data extraction. A context basically is a group of views. For each of its views it determines the target matrix in YOMDB (the result of the data query will be put into it) and specifies the columns that have to be updated. The definition of contexts can be found in *odb_src*/cma2odb/ctxinitdb.F90. In this chapter a basic introduction will be given for contexts via the definition of context SUFGER. The piece of code below has been taken from ctxinitdb.F90:

```
do j=1,size(sufger_nviews)
  CALL next_context('SUFGER', version=j, nviews=sufger_nviews(j))

  write(ctx(idctx)%view(1)%name,'(a,i1)') 'sufger_robhdr_',j
  ctx(idctx)%view(1)%ncase = JPCASE_ROBHDR

  write(ctx(idctx)%view(2)%name,'(a,i1)') 'sufger_mobhdr_',j
  ctx(idctx)%view(2)%ncase = JPCASE_MOBHDR
  ctx(idctx)%view(2)%execute = .FALSE.
  ctx(idctx)%view(2)%shared_index = 1
  ctx(idctx)%view(2)%mlnkh2b = 4

  write(ctx(idctx)%view(3)%name,'(a,i1)') 'sufger_robody_',j
  ctx(idctx)%view(3)%ncase = JPCASE_ROBODY

  write(ctx(idctx)%view(4)%name,'(a,i1)') 'sufger_mobody_',j
  ctx(idctx)%view(4)%ncase = JPCASE_MOBODY
```

```
    ctx(idctx)%view(4)%execute = .FALSE.
    ctx(idctx)%view(4)%shared_index = 3

    if (j == 1) then
      isize = 2
      allocate(ctx(idctx)%view(3)%update_col(isize))
      ctx(idctx)%view(3)%update_col(1) = 'fg_depar@update'
      ctx(idctx)%view(3)%update_col(2) = 'fg_error@errstat'
    else
      ctx(idctx)%view(1:sufger_nviews(j))%cancel = .TRUE.
      ctx(idctx)%view(1:sufger_nviews(j))%putdb = .FALSE.
    endif
  enddo
```

We can read out from this code segment that context SUFGER has several versions and each of them consists of four views. The do loop goes through the definitions of the different versions. The actual version of the context is represented by the *ctx(idctx)* element of the *ctx* array. The i-th view is represented by the *ctx(idctx)%view(i)* field. The data retrieval itself is defined via the elements of this field. These elements can be as follows:

- name: Name of the view.
- ncase: Specifies the target matrix (in YOMDB) for the view. JPCASE_ROBHDR means ROBHDR etc.
- execute, shared index: If execute=.FALSE. it indicates that it needn't extract data for this view from ODB. In this case the matrix given in *ncase* will contain the same data as the matrix of the view specified with *shared_index*. This solution reduces the data query time. It is important to mention that for execute=.FALSE. the view itself needn't exist physically as an sql file!
- mlnkh2b: Ensures the creation of array MLNKH2B that will describe the relation between the matrix of the actual view and the view given with *mlnkh2b*.

Now, using this description we can find out how data is retrieved for the different view in context SUFGER (j stands for the context version)

- sufger_robhdr_j: Data connected to this view will be extracted into ROBHDR.
- sufger_mobhdr_j: This view does not exist as an sql file. Its target matrix MOBHDR will be filled directly with data from ROBHDR. So MOBHDR will be the integer counterpart of ROBHDR. Array MLNKH2B will be created between MOBHDR and MOBODY!!
- sufger_robody_j: Data connected to this view will be extracted into ROBODY.
- sufger_mobody_j: This view does not exist as an sql file. Its target matrix MOBODY will be filled directly with data from ROBODY. So MOBODY will be the integer counterpart of ROBODY.

The last part of the context definition specifies the columns to be updated. The modified values of these columns will be saved into the database.

6 ODB related programs

6.1 General remarks

Programs

As for the code management ODB programs can be devided into two groups. There are programs provided with the ODB package and there are programs which have to be installed separately from ODB. In the ODB package the most important program is *odbtools* that is nothing else than an interface to the 3 main ODB handling programs: *to_odb*, *to_ecma* and *shuffle*. The actual task of odbtools depends on its actual name, so on execution it has to be linked or renamed to the proper name.

Parallelization

ODB uses MPI for multiprocessor runs. The actual number of pools in the processed ODB must be greater or equal to the number of processors.

Directory creation

When an ODB program creates a new ODB (to_odb or shuffle) the directory tree is created by the program itself. However, on SX4 (in Prague) it is not working, so the complete directory structure must be created in advance.

Ioassign

ODB handling programs requires a scpecific *ioassign* file that describes the directory structure of the database. Each database type has its own ioassign file which is located in the ODB system directories. For example, this file is *odb_sys*/ECMA/ECMA.IOASSIGN for the ECMA database type. The programs reads the location of this file from environmental variable *IOASSIGN*. Because some programs modifies the ioassign file it is not recommended to set IOASSIGN directly to the system files. Much safer to copy the system ioassign file to the working directory and set IOASSIGN to this local file.

Environmental variables

ODB is working with a lot of environmental variables that should be exported before execution. The most important variables are as follows:

- IOASSIGN: Specifies the ioassign file.
- ODB_ANALYSIS_DATE: Specifies the day of the analysis in *yyyymmdd* format for shuffle.
- ODB_ANALYSIS_TIME: Specifies the time of the analysis in *hhmmss* format for shuffle.
- ODB_STATIC_LINKING: Indicates that the views used by the program are loaded dinamically (0) or they are already linked statically to the executable (1).
- ODB_TRACE_FILE: Specifies the log-file for ODB related trace information.
- ODB_TRACE_PROC: Indicates the number of PE for trace. If it is set to **-1** information from all processors will be gathered.
- ODB_CMA: Type of the ODB database in the CMA $\leftrightarrow$ ODB conversion and in screening, minimization and canari.
- ODB_DATAPATH_*dbtype*: Specifies the location of the pools of the given *dbtype* ODB.
- ODB_SRCPATH_*dbtype*: Specifies the location of the *.dd* and *.sch* files of the given *dbtype* ODB.
- TO_ODB_DEBUG: Prints debugging information to the standard error in CMA $\leftrightarrow$ ODB conversion (0 or 1).
- TO_ODB_ECMWF: Outside ECMWF this variable always should be set to 1.

6.2 To_odb

This program creates an ODB from a CMA file. The database type can be ECMA, ECMASCR or CCMA. The program is able to handle more CMA files, as well. In this case the resulted ODB will contain as many pools as the number of the CMA files. But we have to take care of the number of the PE: for a parallel run we have to specify at least as many input CMA files as the number of the PE. The argument of the program is the input CMA file (or files). The example below shows the most necessary settings to run to_odb to create an ECMA ODB:

```
export TO_ODB_ECMWF=0
export ODB_STATIC_LINKING=1
export ODB_CMA=ECMA
export ODB_DATAPATH_ECMA=$tmpdir/ecma_db
export ODB_SRCPATH_ECMA=$tmpdir/ecma_db
export IOASSIGN=$tmpdir/ioassign

#Copy ioassign file
cp $odb_sys/ECMA/ECMA.IOASSIGN $IOASSIGN

#Run conversion
to_odb cmafile
```

6.3 To_ecma

This program creates a CMA file from an ODB. The type of the ODB can be ECMA, ECMASCR or CCMA. The result is one CMA file for each pool. The only argument for the program is the prefix of the name of the output CMA file(s). The results will be named as *prefix*.1, .., *prefix*.N, where N is the number of the pools. The example below shows the minimum configuration to run to_ecma in order to create a CMA file from an ECMA ODB:

```
export TO_ODB_ECMWF=0
export ODB_STATIC_LINKING=1
export ODB_CMA=ECMA
export ODB_DATAPATH_ECMA=$tmpdir/ecma_db
export ODB_SRCPATH_ECMA=$tmpdir/ecma_db
export IOASSIGN=$tmpdir/ioassign

#Copy ioassign file
cp $odb_sys/ECMA/ECMA.IOASSIGN $IOASSIGN

#Run conversion
to_ecma cmafile
```

6.4 Shuffle

Shuffle extracts data from a given type of ODB database and writes it into another type of ODB database. The distribution of data into the desired number of pools and timeslots is also ensured. Shuffle can perform three different database conversions:

ECMASCR→ECMA: In this case shuffle creates an ECMA database using all data in the ECMASCR database. This conversion should be used before screening to create an input ECMA database with the given number of pools.

ECMA→CCMA: In this case shuffle creates a CCMA database using only the *active* data in the ECMA database. This conversion should be used after screening to create an input CCMA database for minimization.

CCMA→ECMA: This conversion should be used after the 3D/4DVAR analysis for matchup. So the ECMA ODB will be updated with the minimization results stored in the CCMA ODB.

Argument list of shuffle consists of several flags:

-iidbtype **-o**odbtype **-a**ipn **-b**fp **-B**andate **-L**tstartdep **-R**tenddep **-T**tsnum **-F**ftr **-M**

where:

- idb: Input database type.
- odb: Output database type.
- ipn: Number of pools in the input database.
- fp: Multiplication factor for the number of pools. The output database will contain ipn·fp pools. If it is set to 0 the output pool number will be one (for this the oroginal source of shuffle must be modified).
- andate: Analysis date in yyyymmddhh format.
- tstartdep, tenddep: The departure between the edges of the assimilation time window and the analysis date (in minutes).
- tsnum: Number of timeslots in the output database (1 for 3DVAR).
- ftr: Database type used in final trajectory in minimisation (ECMA or CCMA). It is an optional flag for the matchup.
- **-M** is an optional argument; if it is given then the input database will be opened in READ-ONLY mode.

It can be seen that the actual mode of shuffle is completely defined by the input and output database types. The example below illustrates how to run shuffle to create a CCMA ODB from an ECMA ODB (with 1 pool):

```
export TO_ODB_ECMWF=0
export ODB_STATIC_LINKING=1
export ODB_DATAPATH_ECMA=$tmpdir/ecma_db
export ODB_SRCPATH_ECMA=$tmpdir/ecma_db
export ODB_DATAPATH_CCMA=$tmpdir/ccma_db
export ODB_SRCPATH_CCMA=$tmpdir/ccma_db
export IOASSIGN=$tmpdir/ioassign


#Prepare ioassign file
cat $odb_sys/ECMA/ECMA.IOASSIGN > $IOASSIGN
cat $odb_sys/CCMA/CCMA.IOASSIGN >> $IOASSIGN

#Run shuffle
shuffle -iECMA -oCCMA -a1 -b1 -B2000072100 -L-180 -R180 -T1
```

It is worthy to note that ioassign in this case must contain the ioassign files for both database type. Shuffle is used to reduce the content of the ODB during the ECMASCR→ ECMA conversion. If environmental variable *TO_ODB_REDUC* is set to 1 shuffle will write only reports with subtype@hdr=1 into the target ODB (to do this the original source of shuffle must be modified).

6.5 Lamflag

For ALADIN 3DVAR screening the input ECMA ODB must contain reports only in the C+I zone. At present, the creation of such an ODB is done in two steps. In the first step program *lamflag* scans through the whole ODB and sets column subtype@hdr to 1 for reports inside the C+I zone (and to 0 otherwise). Then in the second step with shuffle only reports with subtype@hdr=1 are selected and written out into a new ODB which is now suitable for ALADIN purposes.

6.6 Viewer

In *odb_sys*/bin there is a program called *viewer* that is able to retrieve data from ODB in ascii format. The example script to run viewer can be found in *odb_src*/tools. With the viewer we can use our own views (SQL queries) for data extraction. During the retrieval the view is compiled into a shared object file that is loaded dynamically (by shl_load) by the viewer. So this method is quite fast because only the given view must be compiled (and we do not have to link a new executable).

6.7 Mandaodb

The content of a CMA file can be easily visualized in many different ways with Mandalay. As for ODB there is a program called *mandaodb* that can perform similar tasks. Mandaodb can use both pre-defined and user-defined views for data extraction. It is executed via a specific script. The retieval in each case is based on specific sql commands (i.e. on views). For the pre-defined retieval modes these views are already complied and linked to the executable. However, a user-defined view must be compiled into a C program with the ODB provided SQL-compiler, then this C program must be compiled to get static objects files and finally a new (temporary) executable must be generated. For the linking a static library containing all the pre-compiled sources is used. The compilation and linking make the execution of the script slower in this case. After the executable is ready in each retrieval mode the last step is the same: the script generates another script for the retrieval and launches it.

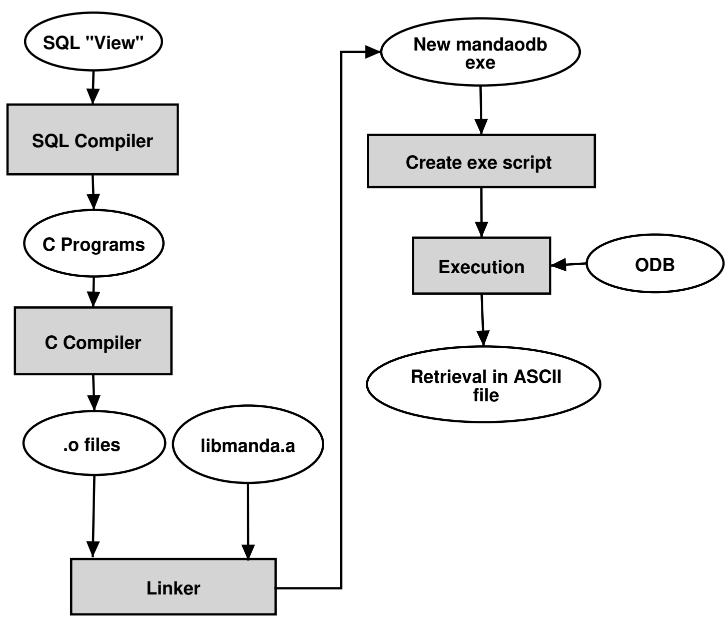

Figure 4: Execution steps of mandaodb for user-defined views

6.8 Batodb

ODB can be created directly from the OBSOUL file using program batodb. At HMS batodb could not run on more than one processors.

7 Working with ODB

7.1 Creating ODB

Getting ODB from Meto France

The simplest way to get an ODB is to download an existing one from Meteo France. At Meteo France observations in ODB format are available operatively since September 1, 2001. For date yyyymmddhh (hh without leading zero) a global ECMASCR ODB with one pool can be found on **delage** as:

```
/chaine/mxpt/mxpt001/arpege/oper/assim/yyyy/mm/dd/rhh/ecmascr.tar
```

This tar-file contains the .dd and .sch files and the pool (its name is 1). For former dates OBSOUL or OBSOB observation files should be converted into CMA file and then into ODB.

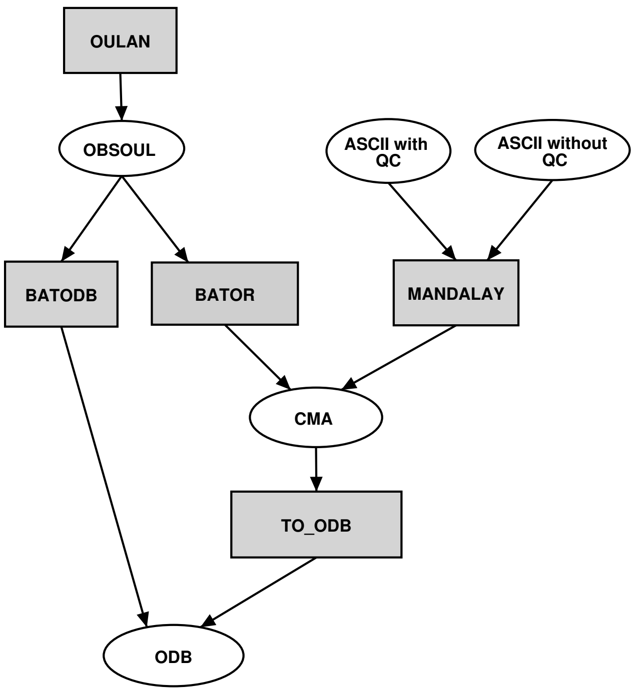

Figure 5: Different ways to create ODB

Conversion from CMA

ODB can be created from a CMA file with program to_odb. The CMA file itself can be generated with bator or with mandalay. The input of bator is an ascii file in OBSOUL format. As far as I know, mandalay can convert two kind of ascii files into CMA:

- Files containing with quality flags can be converted into CMA with setting LAERCQR=.T. in the namelist.
- Files without quality flags can be converted into CMA with setting LAESIM=.T in the namelist.

The problem of mandalay is that if the CMA file is converted into ODB, then this ODB cannot be converted back to CMA. It is true for both starting ascii format. This problem does not occur with bator. Another problem with the ascii→mandalay→...→ODB conversion is that the ODB cannot be used in canari because some tovs related (!) information is missing from the DDR header. The problem did not occure when I converted the ASCII file into CMA at HMS with mandalay07, so maybe the soultion is the usage of this version of mandalay.

Batodb

ODB can be created directly from the OBSOUL file using program batodb. Another advantage of this application that the resulted ODB can be converted back to CMA readily.

7.2 Dumping of ODB

There are many programs that are working with ascii files created from CMA files with mandalay. These inputs can be also generated in the ODB environment by converting ODB into CMA with to_ecma. However, as it was described in the previous section this conversion is not working if we create the ODB with the ascii→mandalay→CMA→ODB procedure. The usage of bator or batodb can solve this problem. As for the CMA files it is suggested to use them as rare as possible. In the ODB creation process the skipping of CMA files is already possible with batodb. In the ODB dumping mandaodb can create ascii output directly from ODB. At present this dump is only available in a few format but in the future mandaodb will integrate many tasks of mandalay.

7.3 3DVAR

In this chapter the structure of an AL15 3DVAR run with screening/minimization/surface canari will be described.

Creation of ODB

The first step of the process is the creation of an ODB that is suitable for LAM data assimailation. The preparation of such an ODB requires a special treatment. Assuming that we have an OBSOUL observation file there are two different ways to generate ODB:

- We can use bator to convert the OBSOUL file into CMA. In the next step *obsort* has to be applied to this CMA file. We have to use obsort to put some important timslot information into the DDR header of the CMA file. In the abscence of this information the generated ODB (with CMA→ODB conversion) won't be used in screening (it will crash). After running obsort we can convert the CMA file into an ECMA ODB with to_odb. In this method we can replace bator with mandalay, but in this case the input ascii format is not OBSOUL, of course.
- We can apply batodb directly to the OBSOUL file. The resulted ODB will contain all the important timslot information.

At this point we have to make two copies of this ODB because a different preprocessing and therefore a different input ODB is required for screening/minimization and for canari.

Screening/minimization

Screenig is working with an ECMA ODB that contains reports only in the C+I zone of the LAM domain. This means that we have to create a new ODB with a reduced number of data. The creation of such an ODB is done in two steps. In the first step program lamflag scans through the whole ODB and sets column subtype@hdr to 1 for reports inside the C+I zone (and to 0 otherwise). Then in the second step with shuffle only reports with subtype@hdr=1 are selected and written out into a new ODB. This ODB is now suitable for screening. Screening will put all its decisions into this ODB. After screening we have to run shuffle to convert this ECMA ODB into a CCMA ODB. This conversion will select only the observations that got active flag during screening. The minimization uses this CCMA ODB as an input and updates it with new information. When the minimization is over it is possible to run shuffle again to put back the new information from the CCMA ODB into the ECMA ODB (i.e. to execute a matchup).

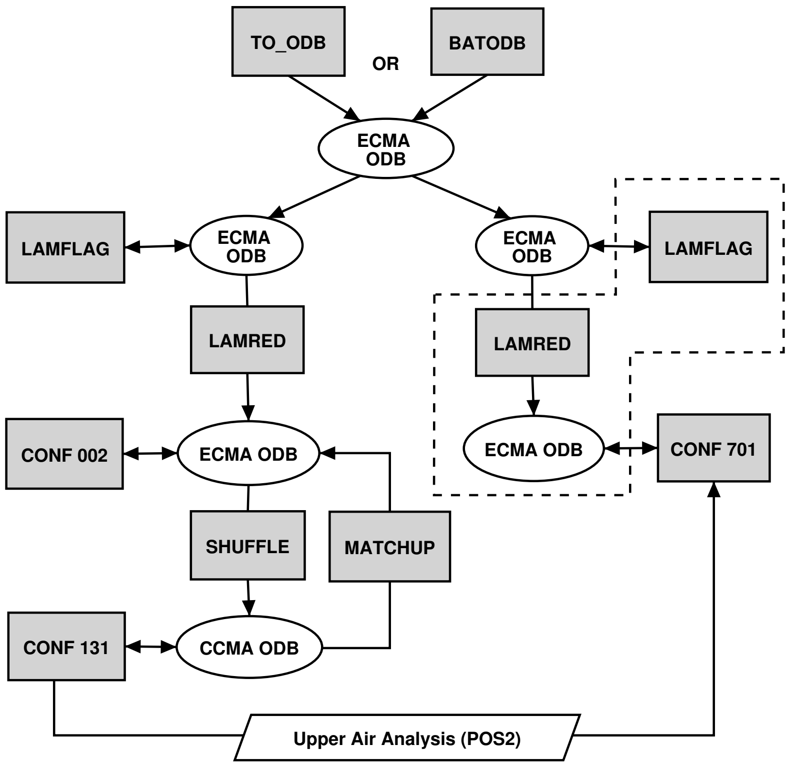

Figure 6: Data flow in AL15 3DVAR (the steps bounded by dashed line are optional)

Canari

Canari is used in this process for the analysis of surface fields. Canari can work with observations outside the C+I zone, so in this case we do not have to run lamflag and shuffle and we can use directly the "raw" ECMA ODB. Canari will put all its decisions into this ODB. We have to remark that this solution means that different observations will be used in screening/minimization and in canari.

7.4 Veral

The *veral* package theoratically can be easily modified in order to be able to run with AL15/ODB. The veral procedure with AL12 can be divided into 3 main parts (represented by 3 scripts):

1. At first ascii observation files that do not contain quality flags are converted into CMA. It is done with mandalay. Then canari with NAEINC=0 is applied to this CMA file to compute and write quality control information into it.

2. In the next step canari has to be run again with NAEINC=2 that prevents the quality control. The input of this run is the CMA file containing the quality control flags. The result of canari is a CMA file, that contains the desired obs-fcast departures for verification. Hereupon mandalay is run to compute spatial averages of scores and write them into an ascii file.

3. The third step is the computation and visualization of the verification scores based on the ascii output of the previous step.

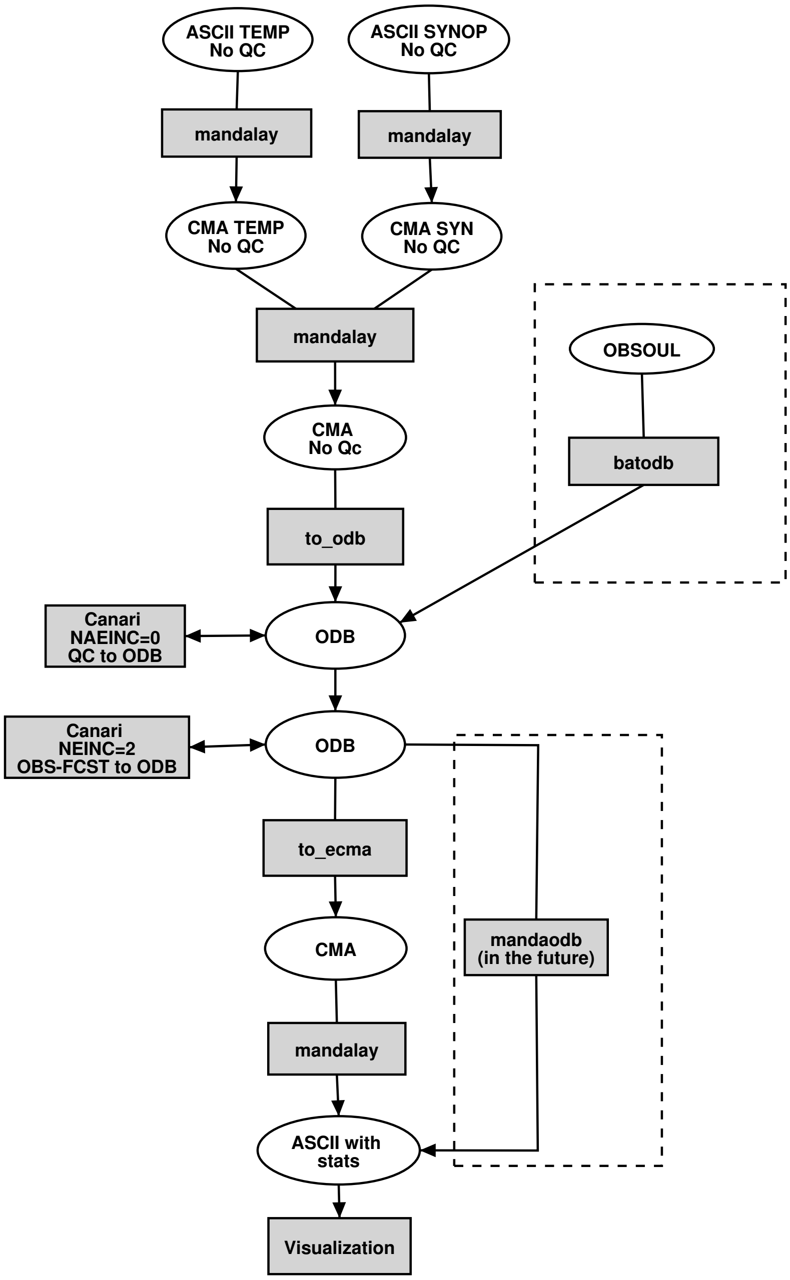

Figure 7: Data flow in VERAL with AL15 (the steps bounded by dashed line are optional)

The simplest way to insert AL15/ODB into this procedure is described step by step:

1. In the first step the CMA output of mandalay should be converted into an ECMA ODB with to_odb. Then a canari run with NAEINC=0 would put the quality flags into this ECMA ODB.

2. In this step another canari run would put the desired departures into the same ECMA ODB. Then this ODB should be converted into a CMA file with to_ecma. At this point mandalay

could be used without any modification.

3. The visualization step needn't be modified.

The only problem with this scheme that to_ecma is not able to convert ODB into CMA if mandalay is used in the ODB creation process. Probably this problem will be solved in the near future, but maybe it is better and simpler to generate the ODB from an OBSOUL observation file with batodb. In this case the ODB→CMA conversion would be working well. Another modification could be the usage of mandaodb to write the desired spatial averages and scores into an ascii file directly from ODB. In the future mandaodb will contain this option.

References

[1] Puech D.: ODB Guide pratique. Users guide to ODB.

[2] Saarinen S.: ODB - Observational DataBase. Presentation for Meteo France, 18-22/09/2000

[3] Puech D.: HTML documentation for ODB (in French).

[4] Kertesz S.: Testing of ODB on NEC SX4. Report on work in Prague July 8-26, 2002