Technical note on observation handling in ALADIN cy28t0

Gergely Bölöni (HMS)

12th March 2004

Observation handling for ALADIN data assimilation purposes was described in general by Sàndor Kertész (2001) for example. Here, the chain of procedures to prepare the observations will be briefly described. The emphasis will be put on the new features appearing in the recent cycles (cy27, cy28) of ODB (D. Puech personal communication), that were obviously leading to several changes in the observation preparations for ALADIN data assimilation too. The text below might be easier to follow having a look at the script ~mrpe703/3dvar/2004/scr/Obs2LamOdb_28.ksh on tora, or vice versa, the script might be easier to understand while reading this text.

1 The new structure of ECMA ODB

The new structure of ODB enables the preparation of ECMA type data bases separately for each observation type, that after all, still can be handled as one common "virtual" data base while using the ODB related programs (shuffle, lamflag) in the most of the cases. This gives more flexibility in assimilation experiments where the user does not want to use all the data available for a given analysis time, but just a subset of them. (However, this flexibility can be important mostly using huge global data sets, to save memory and CPU, the new structure is a technical constraint also in the ALADIN observation preparations.) In the new structure of ODB, the ECMA data base consists of subset of data bases for each observation type and one virtual base that describes the structure of all the sub-bases together as a common base. It is important to point out, that each sub-base has in fact the same structure as the ECMA type of base "well-known" from previous cycles up to know (Puech or Kertész, 2003), so it contains the descriptor files ECMA.dd, ECMA.sch, and the pools containing approximately the same tables as up to now, etc. On the other hand the virtual base contains nothing else, than descriptors pointing on the different sub-bases and their contents. These descriptors are the header file of the whole data base (ECMA.dd), the IOASSIGN file, describing the structure

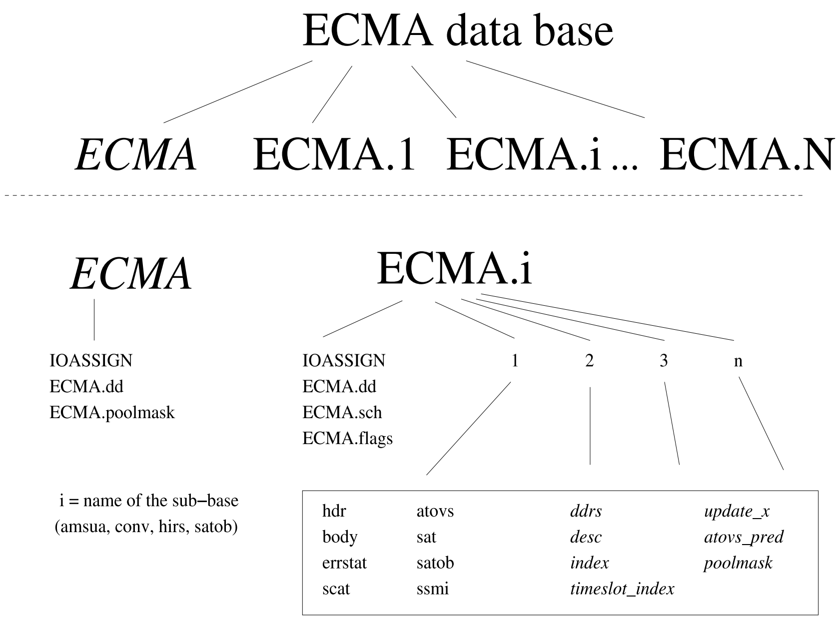

Figure 1: The new structure of ECMA ODB

of each sub-base, and a poolmask file (ECMA.poolmask). The new structure of ODB is illustrated on Figure 1. As a consequence of the new structure, a few new environmental variable have been introduced. These are as follows:

* ODB_IO_METHOD

* BATODB_NBPOOL

* ODB_CCMA_POOLMASK_FILE

* ODB_CCMA_CREATE_POOLMASK

Here we will just explain the variable ODB_IO_METHOD as it is a general one, the others will be described later on in this note. The ODB_IO_METHOD variable can have a value of 1 or 4. These values are both corresponding to the same structure explained here above, but the distribution of data to pools is different. In case of ODB_IO_METHOD=1 the user always has to give explicitly the number of pools to be created. On the other hand, if ODB_IO_METHOD=4, the number of pools will be automatically specified by the software itself, depending on the size of the input data files. This latter is not planned to be used in ARPEGE/ALADIN, so far, so ODB_IO_METHOD should always be set to 1. This value has to be set by an export, just like other environmental variables of ODB, before running any ODB related program.

2 Observation preparations for ALADIN

Due to the new structure of ECMA type ODB, the observation preparations for ALADIN data assimilation have been also slightly changed. So, here we give an explanation of the chain of procedures to create a LAM ODB data base in cycle cy28t0.

2.1 Creating odb

As we know, from earlier documentations, to create an ODB data base, one has to run the program batodb with an ascii input file (OBSOUL). First of all, the BATODB_NBPOOL environmental variable has to be set. It has to be exported with the value of the number of pools we want to create. To prepare an ECMA data base in the new structure explained above, batodb has to be run separately for each data type (i.e. amsua, conv, hirs, satob). To be able to do this, of course, separate OBSOUL files for each data type have to be available for the user (OBS.amsua, OBS.conv, OBS.hirs, OBS.satob). We will not detail here, how such OBSOUL files can be created (see an example on andante: ~mrpa660/WORK/CY27/OULAN/EXE_oulan.ksh). Before running batodb to create each ECMA sub-base, one has to ensure the presence of the IOASSIGN file in the actual working directory, that will determine the structure of the data base. In earlier cycles the IOASSIGN file was always simply copied to the actual working directory, from the ODB system directory. This is a simple solution and it works well, but it assumes that we always create the data base in the same structure, that is on the same number of pools. To prepare the IOASSIGN file in a more intelligent way, one can use in the new cycle the procedure prepared by D. Puech: tora: ~mrpa660/util/ODB_28R0/PROC_EXEC/create_ioassign. This procedure is a shell script with two input arguments. The first argument specifies the type of ODB (ECMASCR, ECMA, CCMA) for which we create the IOASSIGN file and it should be given after a flag "-l". The second argument specifies the number of pools on which we create the data base, which should be given after a flag "-n". For example the command:

```
~mrpa660/util/ODB_28R0/PROC_EXEC/create_ioassign -l ECMA -n 4
```

executed on tora, creates an IOASSIGN file for an ECMA type of data base, with 4 pools. Once the IOASSIGN file is created, and is present in the actual working directory, batodb can be run sub-base by sub-base. Here, it is important to mention that batodb does not create all the ECMA tables in the recent cycles any more. Namely, the tables *update_x, atovs_pred, index, timeslot_index, desc* and *poolmask* are created at a later stage, when merging the data sub-bases into the virtual base. The merge of sub-bases can be done once they have been created with batodb. The merge consists of the creation of the IOASSIGN file of the whole virtual base and of a shuffle run, which creates the header and poolmask files (ECMA.dd, ECMA.poolmask) of the virtual base, and also, completes the sub-bases with the missing tables mentioned above. The IOASSIGN file here has to describe the structure of all sub-bases, so in fact it will be a merge of the IOASSIGN files of each sub-base. This merge can be done using the prepared procedure of D. Puech again:tora: ~mrpa660/util/ODB_28R0/PROC_EXEC/merge_ioassign. This procedure is a shell script too, with two input arguments. As first argument the path of the ECMA sub-bases should be given with a flag -d and as second argument the name of the base should be specified with a flag -t. So, to merge, for example the IOASSIGN files of the sub-bases ECMA.conv and ECMA.amsua located in $workdir, one has to execute the following command:

```
~mrpa660/util/ODB_28R0/PROC_EXEC/merge_ioassign -d $workdir -t conv -t amsua
```

Of course, the IOASSIGN file should exist before running shuffle, and again it has to be copied into the actual working directory. It is well known, that while running shuffle, the input and output ODB type must be specified (flags -i and -o). To perform a merge "ECMA" should be given both for input and output odb types. After the shuffle run an ODB base is created with a structure as shown on fig.1.

2.2 Lamflag

Lamflag is working on the new structured data base without any problem. All we have to do is to specify the ECMA_ODB_DATAPATH and ECMA_ODB_SRCPATH variables properly to point on the virtual ECMA base. There is also a minor change in the lamflag namelist, the variable CANZONE has changed to Z_CANZONE.

2.3 Shuffle (C+I reduction and split to more pools)

This configuration of shuffle is not working yet on the virtual base directly, so it has to be run sub-base by sub-base. A straightforward solution is to prepare a loop for the sub-bases and perform the reduction for each, separately. To run shuffle in this configuration (reduction of observations outside the C+I zone and distribution of the observations to several pools) the input ODB type must be specified as "ECMASCR". As the output of the previously performed lamflag is an ECMA type of base, one has to transform the actually concerned ECMA sub-base into an ECMASCR sub-base. More simply saying (as the ECMA and ECMASCR type of bases are the same in their content), the ECMA.* files should be renamed to ECMASCR.* inside the concerned base. To run shuffle, a proper IOASSIGN file has to be provided again (and of course, copied into the actual working directory!). This means, that it should describe both an ECMASCR (input) and an ECMA (output) type of base. Consequently, the procedure of D. Puech to create IOASSIGN should be used again (-l "ECMASCR ECMA"). After running shuffle subsequently on each sub-base, a new merge of their IOASSIGN files is needed to be able to use further their corresponding virtual ECMA base. This re-merge is important if the number of pools was changed in shuffle. The data base we have at this stage is ready for screening.

2.4 Screening

The ALADIN screening code will run without any problem with a new structured ODB base. All we have to do is to copy the merged IOASSIGN file of the data base into the actual working directory.

2.5 Shuffle (reduction after screening)

To create the CCMA ODB after screening one has to run again shuffle. To do this the output data base type should be specified as CCMA and the environmental variables ODB_CCMA_CREATE_POOLMASK and ODB_CCMA_POOLMASK_FILE have to be set. The first has to be exported with a value 1, the latter has to specify the path of the CCMA poolmask file (in fact it has to be exported with a value $ODB_CCMA_SRCPATH/CCMA.poolmask). Also before running the shuffle executable, the already well known procedure of D. Puech should be run to create an IOASSIGN file describing the CCMA base to be created (-l CCMA). The created CCMA base will be just like in previous cycles, the sub-bases are disappearing at this stage, so 3dvar will use it as before.

2.6 3dvar minimization

However, it's not about the observation preparation itself, it is not neglegible that the 3dvar code of ALADIN was running using a CCMA base prepared as it is described above. The correctness of the minimization in cy28t0 has been also validated, via comparison with cy26t1. (The code to provide compactly supported background error correlations has been revalidated too.)

References

Kertész, S. 2001: Observation management for ALADIN. *ALADIN Technical Report,* 30 pp.
Kertész, S. 2003: Using ODB in ALADIN, *LACE lecture note,* 21 pp.
Puech, D.: ODB Guide Pratique. *ODB users guide (on the GMAP web page f.e.)*