ODB User Guide

Sami Saarinen

ECMWF, Reading, UK

7th May 2004
OOD User Guide

Executive summary

You are reading the first edition of ODB User Guide, which is related to the Integrated Forecasting System’s (IFS) cycle CY28R2.

OOD stands for Observational DataBase. It is database software to store and retrieve large amounts of meteorological numerical data in an efficient manner while used from within IFS. ODB software mimics relational database queries through its ODB/SQL -compiler and accesses data currently via a Fortran90 library interface.

The document tries to explain as comprehensively as possible the concepts behind ODB and provides many examples of how to use the system. Despite being a large document, the intention was not to go into too small details, since they can be easily found in the source code.

The document is divided into ten main sections. In brief, these sections cover the following major items:

1. ODB at a glance gives a gentle introduction to what ODB is and how a typical observational database looks like.
2. Core ODB explains the core software, and how its data querying, partitioning and packing work.
3. ODB/SQL -language gives detailed introduction how to use the "heart" of ODB – that is: data querying and filtering.
4. ODB Fortran90 access layer gives a tour of the ODB functions i.e. how to manipulate a database and access ODB/SQL -queries from a high level language like Fortran90.
5. Some examples, which it is useful to browse through almost immediately, since it gives an indication about ODB complexity in a form of complete – yet simple – examples.
6. ODB interface to IFS explains how a major ODB application – IFS – is coupled with ODB software.
7. Database related to IFS continues with IFS related aspects including data definition layouts for currently recognized (permanent) databases in IFS like ECMA and CCMA databases.
8. Guidelines for modifications covers most of the aspects that need to be remembered in order to modify something in ODB (or ODB to IFS) environment.
9. Using other database layouts extends the use of ODB by explaining how to use ODB software with non-IFS related databases.
10. *ODB applications* covers a few useful programs that are built on top of ODB, most notably BUFR2ODB and odbviewer.

The main sections are followed by appendices, the contents of which is summarized below:

- **A** appendix covers *Environment variables* i.e. almost all you wanted to know about various Unix environment variables affecting ODB execution.
- **B** *Miscellaneous observation type codes* could perhaps be the most important section in this document. Despite being not directly ODB software related, the observation codes are of major importance for a user in ECMWF environment.
- **C** *Variable numbers* is another important reference in order to find out what kind of observations are in the database.
- **D** *Important files for reference* simply lists most frequently referred files in the ODB framework. Thus, it is useful to have them listed in this document Note, however, that some of them are subject to frequent modifications.
- **E** *ODB source files and binary codes* is a supplementary reading for those who want to understand what kind of object libraries drive ODB software and which source files need to compiled to create ODB software.
- **F** *ODB limitations* briefly explains current ODB software limitations and gives some work-arounds.
- **G** *Data sorting* covers a famous FAQ\(^1\): how could I sort my dataset (possibly quickly)? The answer is to use function KEYSORT also used inside ODB software.
- **H** *Dr.Hook instrumentation tool* is a little bit out of context for this documentation. However, it provides essential information about ODB performance and produces reliable calling trees upon failures.

Finally, the document contains an extensive glossary, bibliographic references and finishes with an index.

**Why have ODB?**

The main reasons to have ODB were as follows:

- Enable large amounts of satellite observations through the ECMWF 4D-Var-system
- Simplify introduction of new observation types
- Manage and control observation processing data flow through data queries
- Have full control over the desired features via source code (ODB/SQL-compiler and Fortran90 library)

\(^1\)Frequently Asked Question
Some of the design goals were as follows:

- Design a relational-like database for MPI-parallel data access
- Implement transparent access to observational data through data definition and query language (i.e. ODB/SQL -language)
- Bind ODB/SQL -compilations into Fortran90 module interface
- Maintain high vectorization efficiency for storing and retrieving observational data
- Introduce well performing and lossless data packing methods
- Enable fast, flexible and configurable I/O management
- Enable MPI-parallel data queries, but also coordinated queries for data shuffling between MPI-tasks
- Allow compatibility with older ODB-databases by re-generating data access libraries directly from database metadata
- Implement such binary format for database files which enables transparent access to data on any (Unix-)platform without explicit or external data conversions

About 95% of the core ODB software is written in C-language, the rest is written in Fortran90. The core library is currently not fully thread safe for OpenMP, but the interface to IFS has been adapted for safe OpenMP-processing via a low-overhead locking mechanism.

The ODB software has purposely been divided into the core part and IFS-interface. The latter is dependent on IFS modules and its environment, whereas the former can also be used for non-observational (but numerical) database accesses.

ODB software itself contains about 3,000 lines of Fortran90 and 45,000 lines of C code. Automatically generated C-code for ECMWF 4D-Var-system from SQL is about 280,000 lines. The separate ODB interface to IFS is about 17,000 lines of Fortran90. Translation from BUFR-format to ODB and back is itself about 65,000 lines of Fortran90.

**A short history**

ODB has been developed at ECMWF since mid-1998. There was a need to replace the CMA-file structure (developed by Drasko Vasiljevic at ECMWF) with a less disk space and memory consuming format and provide more scalable (and parallel) access to ever growing satellite observational data. At the same time an idea of using relational database concepts for easier data selection and filtering was born. The main inspirational spark was absorbed from the reference [B.M91].

A lot of thought was still needed to maintain high vector processing speed, without sacrificing I/O performance, and yet enable high data packing ratios without loss of accuracy.
These special needs in observation handling, software availability and support on various platforms, and high price precluded the use of commercial databases.

The first 4D-Var-system with ODB went into operations at ECMWF with IFS cycle CY22R3 on 27th of June 2000. This system still used CMA-files, but only as intermediate ones. That is to say, input observations BUFR-data format were translated into CMA-files from which it was relatively easy to map observational data into ODB-format, because of the similarities in data structures of CMA-files and ODB data layout.

Before June 2000 the ODB software had already been used for nearly a year in the ERA40 reanalysis-project to detect duplicate conventional observations through a database called PREODB. In this database, some key information was first picked up from BUFR-messages to aid selection and speed-up duplicate detection before observational data was passed to the re-analysis.

A weak OpenMP-support by means of locking mechanism was added to IFS interface to ODB in early 2001. This enabled multiple OpenMP threads to select and update mutually exclusive observational datasets without observational data sharing conflicts. This scheme was considerably enhanced by replacing costly high-level OpenMP-locks with low-overhead locking in Spring 2004 for cycle CY28R2.

The IFS cycle CY26R1 (29th of April 2003) introduced a major technical change to ECMWF’s observation handling system. In this version observations get translated directly from ECMWF BUFR-format into an ODB-database, and after analysis are fed back to feedback-BUFR. This facility was developed jointly with Milan Dragosavac (BUFR2ODB-tools) and Drasko Vasiljevic (MAKECMA-replacement in IFS).

Meanwhile the ODB software and especially its separate IFS interface was enhanced so that separate streams of observational groups can be loaded independently and in (job-)parallel mode. This is one of those unique features of ODB which enables nearly endless amounts of observations through the 4D-Var-system. It is now the backbone in providing high observational data volume into ECMWF 4D-Varsystem.

The new observation handling system has now many levels of parallelism: job level (one stream of observation type per job) and program level (both MPI distributed memory and OpenMP shared memory parallelism is available – simultaneously). That is to say, when there are enough processors available, huge amounts of observational data can indeed be processed through the 4D-Var-system.

A typical size of twelve hour 4D-Var observational database has reached 5GB at ECMWF. The data is heavily packed without loss of accuracy, and represents in fact about 15GB of data, had it not been packed. This is a major step forward from earlier days (ca. 1997), when a six hour 4D-Var-system used less than half a gigabyte of unpacked data in CMA-file format.

You can load the PDF-file or the PostScript-file of this document from ...
Acknowledgements

This document would probably not have ever seen daylight without constant encouragement from various people at ECMWF, especially Adrian Simmons, Jean-Noel Thepaut, Erik Andersson, Sakari Uppala, Mats Hamrud, Drasko Vasiljevic, Milan Dragosavac and Deborah Salmond. They and many other people have kept the candle burning at the end of tunnel!

I am also very grateful to the earlier documentations kindly provided by Sándor Kertész from Hungarian Meteorological Service and Dominique Puech from Metéo-France, which gave a solid starting point and layout to this document.

Typesetting of this book has been done by using LaTeX processing system with a great help from Carsten Maass from ECMWF and using references [Bue90] and [Dal99].

Finally, I would like to thank my family for persistent support in getting this document finished.

Sami Saarinen
ECMWF, Reading, UK
Author of ODB
7th May 2004
Contents

ODB User Guide 3

1 ODB at a glance 13
   1.1 A typical observational database 13
   1.2 Data definition 13
   1.3 Data queries 16
   1.4 Fortran90 access to database 16
   1.5 Data access patterns 17
   1.6 Data partitioning: pool concept 18
   1.7 Availability of the ODB software 19

2 Core ODB 21
   2.1 Introduction to ODB software layers 21
   2.2 ODB/SQL compilation system 22
   2.3 Data partitioning 26
   2.4 Missing data indicator 30
   2.5 Data packing 30
   2.6 Database files 31

3 ODB/SQL-language 35
   3.1 ODB/SQL: language summary 35
   3.2 Data definition layout 39
   3.3 Data queries 41
   3.4 Miscellaneous ODB/SQL language elements 43
   3.5 Built-in functions 44

4 ODB Fortran90 access layer 47
   4.1 Compiling and linking ODB-access programs 47
   4.2 Description of the ODB Fortran90 functions 48

5 Some examples 81
   5.1 Initialization of ODB environment 81
   5.2 Creating a new database 82
   5.3 Populating with data 86
## CONTENTS

5.4 Simple data querying .................................................. 89
5.5 More complicated requests ........................................... 94
5.6 Using parameterized queries ......................................... 97
5.7 Updating database .................................................... 102
5.8 Adding more data ........................................................ 107
5.9 Archiving and removing database .................................. 111

6 ODB interface to IFS .................................................. 113
6.1 Architecture ............................................................ 113
6.2 Observational arrays, MDB-variables and MLNK-pointers .... 115
6.3 The main interface functions ......................................... 116
6.4 Additional interface functions ....................................... 125
6.5 Shuffling observations between two databases ................. 132

7 Databases related to IFS ............................................. 137
7.1 Table hierarchies ...................................................... 137
7.2 User defined datatypes ................................................ 137
7.3 Database ECMA ........................................................ 147
7.4 Database ECMASCR ................................................... 161
7.5 Database CCMA ........................................................ 161

8 Guidelines for modifications .......................................... 163
8.1 A checklist .............................................................. 163
8.2 Important files and directories ...................................... 164
8.3 Adding a column entry ................................................ 165
8.4 Modifying a column entry ............................................ 166
8.5 Removing a column entry ............................................. 167
8.6 Adding an array of column entries ................................ 167
8.7 Modifying an existing data query .................................. 168
8.8 Adding a new data query ............................................. 169
8.9 Adding a new retrieval ................................................ 170
8.10 Modifying an existing retrieval .................................... 173
8.11 Adding a new table ................................................... 173

9 Using other database layouts ......................................... 175
## CONTENTS

9.1 Database PREODB ........................................... 175  
9.2 User defined, temporary databases .................................. 176  
9.3 Adding new databases in permanent basis .................................. 178  

10 ODB applications ........................................... 181  
10.1 BUFR2ODB ........................................... 181  
10.2 ODB2BUFR ........................................... 183  
10.3 ODB-viewer ........................................... 185  
10.4 simulobs2odb ........................................... 185  
10.5 odb2netcdf ........................................... 186  
10.6 odb2mysql ........................................... 187  

A Environment variables ........................................... 189  

B Miscellaneous observation type codes ........................................... 197  
B.1 BUFR observation types and its (ECMWF) subtypes .......................... 197  
B.2 CMA observation types and its codetypes .................................. 198  
B.3 Mapping between CMA and BUFR types .................................. 200  
B.4 Satellite identification codes ........................................... 203  

C Variable numbers ........................................... 205  

D Important files for reference ........................................... 209  
D.1 Database layouts ........................................... 209  
D.2 ODB interface to IFS ........................................... 225  

E ODB source files and binary objects ........................................... 239  
E.1 ODB source file organization and libraries .................................. 239  
E.2 Building ODB binaries ........................................... 241  

F ODB limitations ........................................... 245  
F.1 Column entry size ........................................... 245  
F.2 Maximum number of rows per request .................................. 245  
F.3 Character string limit ........................................... 245  

G Data sorting ........................................... 247
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>H Dr.Hook instrumentation tool</td>
<td>249</td>
</tr>
<tr>
<td>Glossary</td>
<td>259</td>
</tr>
<tr>
<td>Bibliography</td>
<td>269</td>
</tr>
<tr>
<td>Index</td>
<td>275</td>
</tr>
</tbody>
</table>
1 ODB at a glance

This section explains some basic ideas behind the ODB software.

1.1 A typical observational database

A typical observational database contains the following items:

- Observation identification information (date, position, station identification)
- Observed values for different observation types, for example: winds, humidities, temperatures per pressure level or geopotential height; or brightness temperatures per satellite channel number
- Various flags indicating quality and validity of an observation, like whether it is active or rejected, or whether the measurement can be trusted etc.
- Departure from observed value (i.e. observed value minus background field, observed value minus analyzed)
- Bias corrections
- Satellite specific information like zenith angle, field of view and so on
- Other important observational processing and meteorological information

When each of the items have been placed in its appropriate table, the designated columns can be accessed by using their descriptive names. This means that data querying and especially data filtering becomes fairly straightforward. This is where ODB/SQL query language compiler plays an important role.

1.2 Data definition

An ODB database is constructed out of basic building blocks called tables. You can think of them as matrices (2D-arrays) with a number of rows and columns containing numerical data. For data querying purposes each table and its columns have a name. Each column also has a number of attributes (the currently implemented ones are datatype and column packing method number).

Such a table would be described in the ODB-system through its data definition and query language ODB/SQL as follows:
In practice a (relational) database normally consists of many tables, where data is related to each other normally via one-to-many relationships. Say, for every spatial position a number observational measurements (like air pressure, temperature, wind speed) has taken place. It is then advisable to put positional information into a "parent"-table and the actual observed values into a "child" table and bind these tables together. In ODB the table relations are handled via a link concept (datatype @LINK). The @LINK is a special column which is present in the parent table and allows one-way referencing from a parent-table to its child-table. In detail, a link is a pair of numbers containing absolute row-offset to a particular child table and the count of consecutive rows in the child-table "belonging" to the parent starting from the offset position.

In this case the data definition is only marginally more complex than above:
1.2 Data definition

The `@LINK` concept is the main difference between a relational database approach and ODB’s hierarchical approach. The genuine relational database engines handle connections between tables through special identification columns (PRIMARY KEYS, FOREIGN KEYS and REFERENCES) which are present (in one form or another) in both parent and their child tables. Furthermore there is a speed-up in data queries by use of pre-calculated indices. All this is not applicable at present in the ODB-system, where child-information (e.g. measurements) can be located instantly by looking at the data behind the `@LINK-item from its parent-table.`
1.3 Data queries

Data queries are probably one of the most important reasons to have ODB. They are an easy way to select desired information from a database and place it in a data matrix for further examination. A data query that selects all aircraft temperature observations (below $-10^\circ$Celsius) in certain region could be as follows:

```sql
// File query.sql
READONLY; SET $airep = 2; // IFS-tag to denote aircraft obs.
CREATE VIEW query AS
    SELECT lat, lon, obsvalue, an_depar, status.active@body
    FROM hdr, body
    WHERE obstype = $airep
    AND varno@body = $t
    AND obsvalue < -10C // -10C converted to Kelvin by ODB/SQL
    AND abs(rad2deg(lat)) < 20 // IFS has (lat,lon) in radians
    AND rad2deg(lon) BETWEEN 10W AND 10E // Note: West & East
;
```

The language used is ODB/SQL – almost a minimum subset of SQL.

1.4 Fortran90 access to database

At present ODB software provides access to databases through a Fortran90 library interface, to the underlying C-code. The following small program is an example of how to access the data query above from Fortran90:

```fortran
PROGRAM thefirst
!*** thefirst.F90 ***
USE odb_module
implicit none
INTEGER(4) :: npools, h, rc
INTEGER(4) :: numargs, iargc
character(len=64) dbname, queryname
character(len=64), allocatable :: colnames(:)
rc = ODB_init()
umargs = iargc()
if (numargs /= 2) then
    CALL ODB_abort('thefirst',&
        'Usage: thefirst DBname queryname',numargs)
endif
CALL getarg(1,dbname) ! Database name
```
Later in the document you will be guided how to create a generic data query program which accepts any data queries for any ODB-database. All that matters is how the data layouts, queries and application program are compiled and linked together. Typical commands applicable to the examples here are (the DB denotes here any/generic ODB-database name):

```
$ use odb  
$ newodb DB  # or odbcomp DB.ddl  
$ odbcomp -lDB query.sql  
$ odbf90 thefirst.F90 -lDB -o thefirst.x  
$ ./thefirst.x DB query
```

The last line actually executes our first ODB-application executable thefirst.x. Output of the query can be found in file query.rpt. You may want to browse the output with command:

```
$ b4.x query.rpt
```

### 1.5 Data access patterns

In ODB, emphasis has been put on storing data column-wise in order to perform efficient (often stride-one) data extraction. This enables rapid retrieval of consecutive cells of data with genuine stride one access to data column elements. This is also the usual data access pattern in 4D-Var, so it must be efficient. You could also think of these fetches as vectorized database-to-memory -transfers. Column-wise ordering is also a prerequisite for implementing lossless packing methods in ODB, which pack a whole column of data in one go, usually in a vectorized manner.

One drawback in predefined column-wise ordering is that data insertion between two existing rows (or cells, for that matter) can be very inefficient. The same holds for row deletion. Therefore, we usually do not to do that in ODB, but generate a complete database from the input data by appending data to the end of existing tables in one single sweep. However,
data flow in observation handling never requires physical removal of particular data rows (or cells) while performing complex tasks as required by 4D-Var. Unwanted rows are simply explicitly marked as rejected via status-flags introduced in the data layout. Flags must then be used in the data queries to eliminate undesired rows from data retrievals.

Genuine database engines normally store data row-wise rather than column-wise and keep data rows in a kind of tree structure thus enabling flexible insertion and deletion without much trouble. This approach, however, prohibits stride one access to column data and destroy chances for any good vectorization when lots of data cells are requested. These are features, which must not be neglected in modern meteorological software implementations.

1.6 Data partitioning: pool concept

Another thing that makes ODB unique is its suitability for distributed memory (i.e. MPI-) parallelization. This has been achieved by a simple data partitioning strategy. The tables (and associated data in their child and sibling tables) get split horizontally (row-wise) so that (for example) the first 1000 rows belong to partition#1, the next 1000 rows (say) to partition#2 and so on. In ODB terminology such partitions are called pools. Pools are allocated for each parallel MPI-task in a round-robin fashion so that each processor becomes solely responsible for that partition with exclusive access to its own data pools only. That is to say, no other MPI-task can modify data on a pool that it does not own.

Each data pool to be dealt with is a separate, but at the same time a complete chunk of a full observational dataset. Each MPI-task can manipulate its own partition as it likes without knowing what some other MPI-tasks are doing with their own data pools at the same time. Pools have no connections with each other, so for example data queries are truly local on a per pool basis. Furthermore, each MPI-task owns one or more pools and can perform truly local queries over its own datasets without interfering with the neighboring MPI-tasks. In this way there are no worries about global integrity of data and the pool concept actually implements a kind of databases within a database. This is the key factor in order to process the huge volumes of observational data in the current and future 4D-Var-system in a real(istic) time.

The ODB software can also query and deliver data in a parallel manner. This is accomplished by the Fortran90 interface layer, which explicitly communicates (or even replicates) observations from the owner MPI-task into remote task(s), if necessary. If data was replicated in this way (e.g. a flight path of an aircraft), the rule is that only locally owned data can be updated back to database. Whereas if an individual set of observations gets shuffled over to a remote MPI-task (and only one remote task) then this MPI-task is allowed to update data. This is the only exception, where a pool can be by modified remote MPI-task.
1.7 Availability of the ODB software

ODB software can be made available to ECMWF member state users as source code. A user can simply compile and install ODB software from “tarball” i.e. source code distribution tarfile. This procedure is explained more in appendix E.

ODB is now available in both big and little endian versions for Unix-platforms. It has been tested (or is operational) on the following platforms: Fujitsu VPPs, Silicon Graphics, IBM/RS6000 Power3 and Power4 series, Pentium/Linux (with Portland pgf90-compiler). It has also been compiled for the following platforms: NEC SX, HP-PA, Sun Solaris, Compaq.

It is important to note that ODB databases created (say) on big-endian machines can still be read and updated (say) on little-endian machines (and vice versa). This is possible due to a built-in endian-detection procedure in ODB, which flips bytes as necessary upon loading data.
2 Core ODB

This section explains how the core of the ODB software works in greater detail.

2.1 Introduction to ODB software layers

The ODB software is layered as shown in figure 3. It implements embedded access to databases without client-server intervention. Data can currently be accessed only through a Fortran90 interface layer (described more detail in section 4).

Underneath is found a C-software layer which connects to database queries. And they are in fact ODB/SQL -language transformed into C-code. This generated C-code performs the actual data unpacking and (in many cases) calls I/O-routines to download data on-demand.

Data delivery from parallel queries is handled through the ECMWF message passing library (MPL), called by Fortran90 interface layer. The ECMWF MPL-library in turn calls standard MPI message passing interface library, or alternatively an equivalent library meant for non-parallel runs (libmpi_serial.a).

Figure 3: Software layers in ODB

<table>
<thead>
<tr>
<th>Fortran90 application</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ODB_MODULE</strong></td>
</tr>
<tr>
<td>C-interface layer</td>
</tr>
<tr>
<td>C-code from ODB/SQL</td>
</tr>
<tr>
<td>Data packing, I/O in C</td>
</tr>
</tbody>
</table>

Database queries are processed by using ODB/SQL -language. Firstly, the ODB/SQL -compiler translates data layouts into C-code to provide true access to database files. Then it generates C-code for each SQL-query in order to facilitate fast data extraction and update. Queries are bound to C-layer from which data propagates back to the Fortran90 layer and user-space. The C-layer is also responsible for transparent packing and unpacking of column data, where applicable.

All database I/O in ODB is done by using either standard C or Unix system I/O. The I/O can be done on demand i.e. a whole data table is brought into memory only when needed.
and kept there as long as possible\textsuperscript{2}. Each column in a table gets unpacked only when it is referred to for the first time.

The smallest chunk of data that is currently read in is a single table from one pool (I/O-method\#1). That is to say, the whole table of a particular pool (partition) is read into memory regardless of how many columns need to be accessed.

The latest versions of ODB software also support I/O-method\#4, which is designed to have more efficient I/O, but uses more more memory. In this method data is read and distributed from all tables (by default) when opening the database (see function \texttt{ODB\_open}), even if you are not going to access some tables\textsuperscript{3}. Unless you have a lot of memory to spare and/or multiple processors involved, you may have a problem when postprocessing ODB data from 4D-Var-run. However, there are two ways under development to circumvent this "built-it" problem when using I/O-method\#4 on low-memory systems.

Since availability of memory can be an issue, the future releases of ODB software will see an alternative I/O-scheme (called Direct Column Access or DCA-method), which will allow extraction of a particular column through a seek-and-read operation rather than reading the whole table.

\section{2.2 ODB/SQL compilation system}

Observational data (or any numerical data for that matter) needs to be introduced to ODB through its Data Definition Layout or DDL-file. This is a text file containing possible parameter declarations (\texttt{SET}) or datatype (or Bitfield) definitions and most importantly table definitions.

Tables and their named columns i.e. data definition are the cornerstone of databases. It enables us to refer to data items by their "nice" (user-friendly) names and perform data queries in a relatively easy manner. These queries (also known as SQL-queries\textsuperscript{4}), enable us to extract desired data cells from database which satisfy certain conditions. Once extracted, they can be copied into an analysis program and be modified and re-written back to the database. Alternatively an extraction can be input to a graphical user interface program for plotting positions and their values (say) in a different color or marker.

Here is an example of a data definition, say a DDL-file \texttt{DB.ddl}:

\texttt{2
Not applicable to I/O-method\#4, though, see discussion below

\texttt{3To prevent this from happening refer to environment variable ODB\_CONSIDER\_TABLES (see p.190)

\texttt{4SQL stands for Structured Query Language}
2.2 ODB/SQL compilation system

A data query is expressed in English-like text, which clearly asks to SELECT column items FROM a subset of database tables, WHERE certain constraints apply. The outcome of a query is a sub-matrix containing typically much less data (columns) than was in the originating tables. Also, not all data rows are included, since the constraint-filter (WHERE-condition) restricts the extraction size to contain only the desired rows. Optionally the sub-matrix can also be sorted with respect to one or more result columns (ascending or descending order) before it is presented to the analysis program.

Here is an example of a data query, say a SQL-file query.sql:

```sql
CREATE VIEW query AS
  SELECT b[1], x.second_bit, s
  FROM mytable
  WHERE a > 10;
```

2.2.1 How the ODB/SQL- compilation system works?

The user supplies a DDL-file which will be compiled using the ODB/SQL-compiler. This compilation produces a pre-processed DDL-file i.e. ddd1_file (also known as DDL underscore-file). Compilation also produces a bunch of automatically generated C-codes: the main driver code, a header-file, and one C-file for each table. And all these for each particular database.

The next step is to create data queries. This is done by supplying one or more SQL-files, where data queries are expressed through ODB/SQL language. As long as the DDL underscore-file is available, SQL-files can be compiled into C-code. These C-files will then be compiled with a native C-compiler and put into an object library named after the database name (for example the library libDB.a for database DB).
Once an object library for a particular database is present, your application can gain access to database data through the ODB core library. This library connects the application program to database objects that have just been created and its built-in ODB/SQL-query engine to be driven.

A typical outcome of such a process is that a Fortran90 program receives a subset of data (sub-matrix), which can be manipulated and possibly even updated back to the database.

The ODB/SQL-compilation program (executable `odb98.x`) has been wrapped into a script `odbcomp`. Its purpose is to hide all technical details to make compilation to look like an easy task.

When compiling data layouts (for our imaginary database, DB), we need to invoke the compiler as follows:

```bash
$ odbcomp DB.ddl
```

This produces the pre-requisite files, as explained above. Furthermore, the data queries need to be compiled as follows:

```bash
$ odbcomp -lDB query.sql
```

Here the file `query.sql` contains one or more CREATE VIEW-statements. If there is only one data query present, it is preferable to name the query file prefix after the view name (with `.sql`-suffix) and use lowercase letters in the filename.

If you want to compile several queries, you can supply them one after another, as follows:
The command `odbcomp` concatenates them together and then invokes the ODB/SQL - compiler executable.

Figure 5: Using `odbcomp` -command to compile data layouts and SQL-queries

After all compilations have been performed, the library file `libDB.a` is up to date and in principle ready for linking with the application program and its support (core) libraries. However, one thing is not clear. How do the objects from the support libraries link to this particular library (`libDB.a`)? When the core library was created there may not have been any knowledge of this database name and therefore not even dummy calls to this could have been present.

This dilemma is now solved through a special add-on file called `_odb_glue.c`. It is a crucial bridge between the ODB core libraries and the previously unknown database libraries. It is generated automatically from `static.h`-file when compiling our database `DB` by overriding standard `static.c`-file which is part of the core library. Once this object gets "glued" between the application and the database `DB`, all works fine.

If a particular database becomes standard on a particular site, it should be made permanent. Such a database is called a "recognized" or "permanent" database in ODB parlance.

A permanent (or recognized) database can be overridden (possibly) with some other data layout, but still with the same name. This actually happens all the time when dealing with post-processing requests of ODB. A typical situation: 4D-Var has performed its tasks and the user wants to access conventional data. At ECMWF this can be accomplished by restoring the database from the archive into a user disk and re-creating database access libraries specific
Figure 6: Using `odbcompsch`-command to re-create ODB data access library

---

to this database (DB) from its metadata – in particular using files `DB.sch` and `DB.flags`. The latter is optional, but then it is preferred to have access to the standard `ODB_COMPILER_FLAGS` (see p. ??), normally found in file `$ODB_SYSPATH/odb98.flags`. This reconstruction of libraries is very convenient, since our database can be of any age and yet with the latest software release we are able to recover it without any data conversions. The only thing that is needed is to re-create its data access library (like `libDB.a`). To re-create the access library, treat the file `DB.sch` (a schema file) as if that was your data definition `DB.ddl`-file and create DDL underscore-file with:

```
$ odbcomp DB.sch
```

The rest (data queries, linkage with application) goes as explained earlier.

### 2.3 Data partitioning

We have touched on ODB data partitioning earlier in this document. This subsection is explains it more formally and in detail.

#### 2.3.1 TABLE hierarchy and `@LINK`s

There are in fact three forms of data partitioning in ODB. The first one is to split data between tables by means of data normalization.
In the second way, two or more tables can be said to be aligned if they are split vertically. A special form of this alignment is one-to-one (or one-looper) relationship between tables. Finally, the third way is to split tables horizontally and place datasets into separate pools.

### 2.3.2 Aligned tables

![Vertical data partitioning](image)

If you have lots of columns in one table, you may want to move some columns into another table and still make sure that these two tables retain the same order and have the same number of data rows. In order to achieve this, you need to align the newly created table (also known as "slave" table) with the original table (known as "master" table) where the columns came from.

Table alignment is just a soft assumption and is not forced by ODB. That is to say, the user application must ensure that the master and its slave tables have the same amount of data rows and that rows are one-to-one aligned (horizontally thinking) together, before attempting any SQL-requests.

When alignment conditions are fully met, the ODB/SQL query optimizer treats the aligned tables as if they were "glued" together into single quantity, that is – one big super-table with a large number of columns.
2.3.3 One-to-one relationship tables (one-loopers)

Figure 8: One-to-one relationship between two tables, where data rows are not in the same order

<table>
<thead>
<tr>
<th>tslot</th>
<th>date</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20040201</td>
<td>225000</td>
</tr>
<tr>
<td>3</td>
<td>20040202</td>
<td>012510</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>20040202</td>
<td>011500</td>
</tr>
<tr>
<td>6</td>
<td>20040201</td>
<td>235730</td>
</tr>
<tr>
<td>6</td>
<td>20040202</td>
<td>003020</td>
</tr>
</tbody>
</table>

“index” “hdr”

Sometimes there is a need to have two (or more) tables which are meant to be aligned, but without them having the same number of rows and their rows are not necessarily in the same order as the master table. Such tables are called "one-loopers" (one-to-one -link between rows of the tables) in ODB terminology.

These are very useful. Suppose we have a master table containing full information about positions and observation types. Some observation types however have some additional information specific to this type only. Ultimately you want to exclude them from the master table, since these cell entries are present only for a subset of observation types.

Another example is when two tables have the same number of rows, but in different order. Strictly speaking this does not meet the criterion of being fully aligned tables. Say, the first table contains positional information about observations in more or less random order. The second table contains timeslot information about these positions and is sorted with respect to timeslot number. Clearly what is wanted is to be able to load positional information without considering date and time sorting. Using one-loopers will accomplish this.

2.3.4 Data splitting through pools

A table can be split into multiple tables by assigning upon creation a chunk of consecutive rows into separate partitions. These are called pools in ODB terminology and form horizontal data partitioning.
2.3 Data partitioning

Figure 9: One-to-one relationship between tables, where not all tables have matching row present in the "sibling" table(s).

![Diagram of one-to-one relationship between tables]

Figure 10: Horizontal data partitioning

![Diagram of horizontal data partitioning]

This pool concept helps management of large numbers of observations, since a minimum of one pool at a time can exist in memory. Each pool could also be geographically assigned, so that a particular area of interest could be found in one (or a set of) pools. Pools also come to the rescue when it comes to parallel data querying. Each MPI-task is
assigned to its unique set of pools (in a round-robin fashion) and only those tasks can access and modify data on their pools.

2.4 Missing data indicator

When some cells have no value, their data are said to be missing. Unlike commercial database engines, ODB stores a special value ($mdi$) into those cells to indicate no value. Presently, the missing data value is recommended to be $-\$mdi$ for double precision floating point numbers and $+\$mdi$ for integer quantities. However, users must ensure that these values are inserted explicitly in the application program.

You can query data that is not missing by applying the following WHERE-condition in the request:

```sql
SET $mdi = 2147483647;  // The default
SELECT x,y
FROM t
WHERE abs(z) != abs($mdi);
```

This would correspond to the following query in commercial systems and is also available (recommended way) in the latest ODB-release:

```sql
SELECT x,y
FROM t
WHERE z IS NOT NULL;
```

2.5 Data packing

The way data columns are stored in an ODB database makes them suitable for good packing and usually without loss of accuracy\(^5\). Packing ratios of 3 to 4 are not uncommon. This packing is called internal packing and is transparent to the user. When data columns are requested, their tables are first read in as a whole, columns remaining still in a packed form, in memory. Particular columns are then unpacked on demand, as needed. Columns can also be repacked, if requested, but this would lead to unnecessary performance degradation. Constant unpacking and packing may lead into memory fragmentation and in fact higher memory usage – completely opposite effect to what was originally anticipated.

It is easy to suggest packing of a column: just define a column to be one of the packed integral type: normally `pk1int` for integers and `pk9real` 64-bit floating point numbers.

---

\(^5\)There are lossy packing methods available, too
2.6 Database files

Data packing in ODB can also be external. This means that each table file is subject to gzip'ping (or related) when table-files are written out to disk. Reverse (for example gun-zip) is applied automatically when table data is fetched into memory. To control whether to use external packing or not is down to IOASSIGN-file for a particular database.

External packing usually leads another 10% additional packing when it comes to disk space usage. However, the future direct column access (dca) requires no external packing, since backward seek-operations are very difficult and slow to implement, when a file is externally packed.

From cycle CY26R3 onwards all external packing is removed from ECMWF scripts to accommodate future needs with direct column access. Consequently databases are marginally larger, but faster to access.

2.6 Database files

An ODB database contains two types of files: metadata and actual binary data files. Metadata is usually stored in text-format in order to use it for various data filtering (Perl or similar) and to be able to fix unforeseen problems manually. Metadata resides in directory ODB_SRCPATH_dbname.

The actual data is placed into logical files named after database_name.table_name.pool_number. This normally gets translated (mapped) via IOASSIGN into files $DBDIR/pool_number/table_name.

This however has its own problems in terms of number of files. The most recent I/O-optimization in ODB has addressed this issue and by choosing I/O-method#4 via

```
$ export ODB_IO_METHOD=4
```

By default concatenated files of 32MB will be created. Tables are said to be concatenated horizontally (HC32-file format). One or more (up to ODB_IO_GRPSIZE) similar table-files from adjacent pools are concatenated together into a single physical file until 32MB limit (or ODB_IO_FILESIZE) is reached.

The root directory for binary data can be different from metadata. This is because this enables data to be placed on the systems memory resident file system, whereas less frequently accessed metadata is on disk.

Metadata files are as follows:

**DB.dd** Data description file. The main and originally the only metadata file. Can be used to re-create the data layout in absence of DB.sch (a file, which is absent only in very old ODB databases).
**Figure 11: Database files**

```
DB.sch  Schema file. Nearly a carbon copy of original DDL-file, except some missing items and renaming of derived datatypes (Bitfields).

DB.flags  Supplementary information for the ODB/SQL-compiler. This is a relatively new file and introduced because there was a need to propagate database specific ODB_-COMPILER_FLAGS-files in case the default file (odb98.flags) was missing or had changed radically since the database was created.

DB.iomap  Information about horizontal concatenation. Available to I/O-method#4 only.

The latest version of ODB software generates so called direct column access- or dca-files. These allow potentially faster (and direct) access to ODB-column data without need to read (nearly) all files in memory. This could improve dramatically the postprocessing speed from even large ODB databases.

### 2.6.1 File mapping via ioassign

ODB always opens files using their logical names. These names are mapped into physical names via IOASSIGN-file, which also supplies suggestions for I/O-buffering sizes and...```
whether data need to be read/written through Unix-pipe (e.g. gzip).

Here is an example of IOASSIGN-file for database our artificial database DB with two tables:

<table>
<thead>
<tr>
<th>File</th>
<th>Path</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB $ODB_SRCPATH_DB/DB.dd</td>
<td>1 0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>DB.sch $ODB_SRCPATH_DB/DB.sch</td>
<td>1 0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>DB.flags $ODB_SRCPATH_DB/DB.flags</td>
<td>1 0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>DB.iomap $ODB_SRCPATH_DB/DB.iomap</td>
<td>1 0 0 0 0 0 0 0 0 0</td>
<td>8388608 8388608 8388608 8388608 0 0 32</td>
</tr>
<tr>
<td>DB.body.dca $ODB_SRCPATH_DB/dca/body.dca</td>
<td>1 0 0 0 0 0 0 0 0 0</td>
<td>8388608 8388608 8388608 8388608 0 0 32</td>
</tr>
</tbody>
</table>
3  ODB/SQL -language

ODB/SQL is a language to manage observational databases. It is a minimum subset of international standard SQL used to manipulate genuine relational databases. SQL stands for Structured Query Language and is pronounced as *sequel* or *es-kjuu-el*.

Normally ODB/SQL is used in a semi-interactive way, because ODB/SQL is a compiler that translates particular user defined text files into C-code for native compilation and linking with application program. Application programs must currently be written in Fortran90, although low-level access routines are all coded in C.

You can arrange the use of ODB/SQL to look like an interactive session (see odbviewer, p. 185). ODB/SQL is currently capable of handling the following tasks:

- **Data definition**: To define structure and organization of data to be stored and relationships between the items.
- **Data retrieval**: To define data queries in order to retrieve (normally) a subset of data items.

What ODB/SQL is currently not capable of doing is as follows (see also [N.W02]):

- **Data manipulation**: To add, update, remove or modify data one needs to use Fortran90 access layer of ODB. This situation is subject to change in future releases, where more direct manipulation may become available in one form or another.
- **Access control**: To be able to restrict the user’s ability to retrieve, add or modify data by protecting unauthorized access. However, with Fortran90 access layer, an ODB database can be opened in READONLY-mode, which truly protects from accidental loss or damage of data.
- **Data sharing**: Sharing of a database by concurrent users without interfering each other. This is again already possible for READONLY-databases.
- **Data integrity**: To protect the database from corruption due to inconsistent updates or during system failures. In ODB its important to make sure that @LINK’s between tables stay correct and table alignments are not violated, when making data queries.

3.1  ODB/SQL : language summary

The following subsection summarizes ODB/SQL language in BNF (Backus Naur Form). The notational conventions used are as follows:
- SQL keywords appear in **UPPERCASE**, but can now also be supplied in lowercase. (Mixed case keywords are not yet accepted by **ODB/SQL** compiler).
- Syntax elements are specified in *italics*.
- The notation `item_list` denotes a single `item` or a list of comma separated `items`.
- Vertical bars (`|`) indicate a choice between two or more alternative syntax elements.
- Square brackets (`[]`) indicate an optional syntax element enclosed within them.
- Curly brackets (`{}`) indicate a choice among required syntax elements enclosed within them.

The **ODB/SQL**-language is currently divided into two parts: data definition language and SQL-query processor. The former defines data definition layout i.e. what elements (tables, columns, datatypes) comprise a particular database. The latter implements data query motor in order to search in particular columns of a database satisfying certain conditions.

The data definition layout is normally placed in a database specific file with a suffix `.ddl`. It is recommended to put a single data query per file with a suffix `.sql`. To avoid confusion, the query filename should be named after the viewname (the label `CREATE VIEW` keyword).

### 3.1.1 **ODB/SQL** operators and keywords

The following operators are available in **ODB/SQL**-language:

<table>
<thead>
<tr>
<th>Operator or expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@</code> , <code>.</code></td>
<td>When appears after <code>colname</code> (or <code>colname.memname</code>) signifies the specific <code>table_name</code> in concern. List separator.</td>
</tr>
<tr>
<td><code>as</code></td>
<td>One of the following: `{AS</td>
</tr>
<tr>
<td><code>+</code> <code>-</code> <code>*</code></td>
<td>Add Subtract Multiply, except when after the <code>SELECT</code> or <code>FROM</code>-clause, in which case instructs choose all applicable columns or search from all available tables, respectively.</td>
</tr>
</tbody>
</table>

*continued on next page…*
### 3.1 ODB/SQL: Language Summary

Table 1: ODB/SQL operators...continued

<table>
<thead>
<tr>
<th>Operator or expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/</code></td>
<td>Divide</td>
</tr>
<tr>
<td><code>%</code></td>
<td>Modulo operator i.e. $x % y$ or $\text{mod}(x, y)$</td>
</tr>
<tr>
<td><code>**</code>, <code>^</code></td>
<td>Power operator i.e. $x^y$ or $\text{pow}(x, y)$</td>
</tr>
<tr>
<td><code>SET ... = ...</code></td>
<td>Assignment operator (see SET-variables)</td>
</tr>
<tr>
<td><code>=, ==</code></td>
<td>Equal-to</td>
</tr>
<tr>
<td><code>&lt;&gt;</code>, <code>!=</code>, <code>/=</code></td>
<td>Not-equal-to</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>Less-than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>Less-than-or-equal-to</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>Greater-than</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Greater-than-or-equal-to</td>
</tr>
<tr>
<td><code>&lt; ... &lt;</code></td>
<td>Combined greater-than this, but less-than that</td>
</tr>
<tr>
<td><code>&gt; ... &gt;</code></td>
<td>Same as previous, but reversed</td>
</tr>
<tr>
<td><code>&lt;= ... &lt;</code></td>
<td>Combined greater-than-or-equal-to this, but less-than that</td>
</tr>
<tr>
<td><code>&gt;= ... &gt;</code></td>
<td>Same as previous, but reversed</td>
</tr>
<tr>
<td><code>&lt; ... &lt;=</code></td>
<td>Combined greater-than this, but less-than-or-equal-to that</td>
</tr>
<tr>
<td><code>&gt; ... &gt;=</code></td>
<td>Same as previous, but reversed</td>
</tr>
<tr>
<td><code>NOT</code>, <code>!</code></td>
<td>Logical NOT</td>
</tr>
<tr>
<td><code>&amp;&amp;</code>, <code>AND</code></td>
<td>Logical AND</td>
</tr>
<tr>
<td>`</td>
<td></td>
</tr>
<tr>
<td><code>IN</code></td>
<td>Test if a <code>colexpr</code> has values in a <code>list</code></td>
</tr>
<tr>
<td><code>NOT IN</code></td>
<td>Test if a <code>colexpr</code> has no values in a <code>list</code></td>
</tr>
<tr>
<td><code>BETWEEN ... AND ...</code></td>
<td>Test whether <code>colexpr</code> is between two other <code>colexpressions</code> (with border values included)</td>
</tr>
<tr>
<td><code>LIKE</code>, <code>RLIKE</code></td>
<td>Tests whether a <code>string colname</code> contains particular characters in unix-regular expression fashion</td>
</tr>
<tr>
<td><code>NOT LIKE</code>, <code>NOT RLIKE</code></td>
<td>Same as previous, but tests whether doesn’t have any such characters</td>
</tr>
<tr>
<td><code>IS NULL</code></td>
<td>A shorthand to test whether <code>abs(colexpr)</code> is <code>abs($mdi)</code></td>
</tr>
<tr>
<td><code>IS NOT NULL</code></td>
<td>A shorthand to test whether <code>abs(colexpr)</code> is not <code>abs($mdi)</code></td>
</tr>
</tbody>
</table>

There are no reserved keywords in ODB/SQL that could not be used as column or table names.
Table 2: ODB/SQL keywords

<table>
<thead>
<tr>
<th>Keywords</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALIN</td>
<td>Defines table alignment in which two or more tables should have the same number of rows and appear in the same row-order (horizontal data partitioning)</td>
</tr>
<tr>
<td>RESET ALIN</td>
<td>Resets any previous table alignments</td>
</tr>
<tr>
<td>ONELOOPER</td>
<td>Defines tables which have one-to-one relationship</td>
</tr>
<tr>
<td>RESET ONELOO</td>
<td>Resets any previous one-to-one relationship definitions</td>
</tr>
<tr>
<td>SET</td>
<td>Sets initial value for a scalar variable</td>
</tr>
<tr>
<td>AS</td>
<td>See previous table</td>
</tr>
<tr>
<td>CREATE TYPE</td>
<td>Creates user defined data type (merely Bitfield)</td>
</tr>
<tr>
<td>CREATE TABLE</td>
<td>Creates data layout for table_name</td>
</tr>
<tr>
<td>READONLY</td>
<td>Defines that subsequent select_column’s should not be updatable</td>
</tr>
<tr>
<td>UPDATED</td>
<td>Defines that subsequent select_column’s should be updatable (this is the default)</td>
</tr>
<tr>
<td>CREATE VIEW</td>
<td>Defines query_name</td>
</tr>
<tr>
<td>SELECT</td>
<td>Select desired columns</td>
</tr>
<tr>
<td>SELECT ALL</td>
<td>Synonym to previous</td>
</tr>
<tr>
<td>SELECT DISTINCT</td>
<td>Select columns with distinct (unique) values only</td>
</tr>
<tr>
<td>SELECT UNIQUE</td>
<td>Synonym to previous</td>
</tr>
<tr>
<td>UNIQUEBY</td>
<td>Value in selected columns must pass uniqueness criteria</td>
</tr>
<tr>
<td>FROM</td>
<td>Selects table_name’s in concern</td>
</tr>
<tr>
<td>WHERE</td>
<td>Invokes where_condition</td>
</tr>
<tr>
<td>BETWEEN</td>
<td>See previous table</td>
</tr>
<tr>
<td>IN, NOT IN</td>
<td>See previous table</td>
</tr>
<tr>
<td>NOT, AND, OR</td>
<td>See previous table</td>
</tr>
<tr>
<td>IS NULL</td>
<td>See previous table</td>
</tr>
<tr>
<td>IS NOT NULL</td>
<td>See previous table</td>
</tr>
<tr>
<td>ORDERBY, ORDER BY</td>
<td>Sorts select_column’s</td>
</tr>
<tr>
<td>SORTBY, SORT BY</td>
<td>The same as ORDERBY</td>
</tr>
<tr>
<td>ASC</td>
<td>Invokes ascending sorting (the default) for a sort_column</td>
</tr>
<tr>
<td>DESC</td>
<td>Invokes descending sorting for a sort_column</td>
</tr>
<tr>
<td>ABS</td>
<td>Takes an absolute of a sort_column before invoking either ascending or descending sort</td>
</tr>
</tbody>
</table>

Keywords can be typed in UPPERCASE (recommended) or lowercase letters. No mixed case is currently allowed.
3.2 Data definition layout

3.1.2 Language element syntax

The following language element syntax is used in the subsequent subsections:

<table>
<thead>
<tr>
<th>Language element</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial_value</td>
<td>64_bit_floating_point_number</td>
</tr>
<tr>
<td>scalar_expr</td>
<td>...</td>
</tr>
<tr>
<td>regular_expr</td>
<td>&quot;unix_regular_expression&quot;</td>
</tr>
<tr>
<td>user_datatype_t</td>
<td>name</td>
</tr>
<tr>
<td>typedef_item</td>
<td>memname bitinteger_number_1..32</td>
</tr>
<tr>
<td>target_datatype</td>
<td>accepted_datatypes</td>
</tr>
<tr>
<td>accepted_datatypes</td>
<td>...</td>
</tr>
<tr>
<td>table_name</td>
<td>name</td>
</tr>
<tr>
<td>tabledef_item</td>
<td>name datatype [READONLY</td>
</tr>
<tr>
<td>query_name</td>
<td>name</td>
</tr>
<tr>
<td>colname</td>
<td>name</td>
</tr>
<tr>
<td>column_name</td>
<td>{colname</td>
</tr>
<tr>
<td>select_column</td>
<td>{column_name</td>
</tr>
<tr>
<td>memname</td>
<td>name</td>
</tr>
<tr>
<td>bitfield_member</td>
<td>{colname,memname</td>
</tr>
<tr>
<td>column_regular_expr</td>
<td>&quot; /unix_regex_against_column_names/[i]&quot;</td>
</tr>
<tr>
<td>uniqueby_column</td>
<td>column_name</td>
</tr>
<tr>
<td>sort_column</td>
<td>column_name</td>
</tr>
</tbody>
</table>

3.2 Data definition layout

3.2.1 CREATE TYPE

The CREATE TYPE (or deprecated TYPEDEF) creates user defined datatypes. In ODB parlance this means introduction of a new type, which occupies an unsigned integer word and has member fields consisting of varying length of bits. The total bit occupancy cannot therefore exceed 32 bits. In addition each, so called Bitfield-member has its own unique name to make referencing to them possible in data queries.

The idea of having this kind of user defined bit pattern is that we can effectively save space by packing small integer-like numbers into one word. And yet they all have their own name identifier.
Another form of type definition available in ODB/SQL is type aliasing. Sometimes, perhaps to improve readability, a user wants to call some basic type with a user defined name, which can be more descriptive. Type aliasing has nothing to do with bit pattern definition (i.e. you can alias even a floating point type into your own type name) and is accomplished as follows:

```sql
CREATE TYPE [ | TYPEDEF] user_datatype_t as target_datatype ;
```

### 3.2.2 CREATE TABLE

```sql
CREATE TABLE table_name as
( tabledef_item_list )
;
```

Figure 12: Components in CREATE TABLE statement
### 3.3 Data queries

#### 3.3.1 CREATE VIEW

The data query syntax is as follows:

```
[SET $var [= {initial_value | scalar_expr}] ;]

// This is a comment line
-- This is a comment line, too

[READONLY ;] // read-only-mode for subsequent queries
[UPDATED ;] // ... or all columns will be updatable (the default)

[CREATE VIEW query_name as]
SELECT [ALL | DISTINCT]
  {select_column_list | column_regular_expr_list | *}
  [{UNIQUE BY | UNIQUEBY} uniqueby_column_list]
FROM {table_name_list | *}
[WHERE where_condition]
  [{ORDER BY | ORDERBY | SORT BY | SORTBY} sort_column_list]
;  
```

The `CREATE VIEW` can now be omitted and a query can start from `SELECT-clause` directly, but only if it is possible to determine the viewname from the SQL-filename.

#### 3.3.2 SELECT-clause

```
SELECT [ALL | DISTINCT]
  {select_column_list | regular_expr_list | *}
```

#### 3.3.3 UNIQUE BY-clause

```
{UNIQUE BY | UNIQUEBY} uniqueby_column_list
```

---

6Please note that not all scripts in our IFS environment are adapted to cope with this “omission” of `CREATE VIEW`-statement and consequently compilations may fail. It is a good practice to include `CREATE VIEW` since improves readability.
3.3.4 FROM-clause

```sql
FROM {table_name_list | *}
```

3.3.5 WHERE-condition

```sql
WHERE where_condition
```

3.3.6 ORDER BY-clause

```sql
{ORDER BY | ORDERBY | SORT BY | SORTBY} sort_item_list
```

The ORDER BY-clause allows you to sort the query output with respect to one or more columns found in the SELECT-clause. In the latest version a particular sorting column does not have to be present in the SELECT-clause, instead it will be added automatically into the list of SELECT'able-columns.

Output columns can be sorted in ascending or descending order. Ascending is the default and chosen by the keyword `ASC`. Descending is alternative and denoted by the keyword `DESC`. In addition sorting can be based on absolute value (i.e. ignoring sign) of the column. This is useful for instance if you have to sort with respect to analysis departure values, which can be positive or negative and yet the absolute value determines their significance. This sorting can be chosen by supplying function keyword `ABS()` around the column name. See examples below ... which I don’t have yet, ugh

You can also use `SORT BY` as a synonym. It is not necessary to have whitespace before the `BY`-keyword, but you can use `ORDERBY` or `SORTBY` written together.

3.3.7 Parameterized queries

Parameterized queries are ordinary ODB/SQL-queries, where some of the constants in WHERE-condition are replaced by SET-variables. In this way the application program can change the predefined default values and the same query can be re-used over and over again. It is illegal to use undefined parameters in a WHERE-condition.

<table>
<thead>
<tr>
<th>Normal query</th>
<th>Parameterized query</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE VIEW abc AS</td>
<td>SET $value = 1;</td>
</tr>
<tr>
<td>SELECT x, y, z</td>
<td>CREATE VIEW abc AS</td>
</tr>
<tr>
<td>FROM t1, t2</td>
<td>SELECT x, y, z</td>
</tr>
<tr>
<td>WHERE a &gt; 1;</td>
<td>FROM t1, t2</td>
</tr>
<tr>
<td></td>
<td>WHERE a &gt; $value;</td>
</tr>
</tbody>
</table>
There is no limit in number of parameters (i.e. SET-variables) that can be used in a WHERE-condition. One can change the values of parameters through ODB Fortran90 layer by calling function ODB_setval (see page ...).

### 3.3.8 SET-variables

```plaintext
SET $var [= \{initial_value | scalar_expr\}];
```

Note that a SET-variable can also be defined for data queries (see parameterized queries). In that case their viability is limited to subsequent data query (queries) only. In the following, a variable $x$ is set *after* the first data query and is therefore visible (and applicable) to the second query only. Any attempt to use the variable in the first query would lead to SQL compilation failure, unless variable was already defined in the data definition layout phase.

<table>
<thead>
<tr>
<th>Illegal use of $x</th>
<th>Correct use of $x</th>
</tr>
</thead>
</table>
| CREATE VIEW first AS  
  SELECT a, b, c  
  FROM t  
  WHERE z < $x  // Wrong !!!  
  -- $x not yet defined  
  ;  
  SET $x = 1;  
  CREATE VIEW second AS  
  SELECT a, b, c  
  FROM t  
  WHERE z < $x  |
|                   | SET $x = 1; -- Moved here!  
  CREATE VIEW first AS  
  SELECT a, b, c  
  FROM t  
  WHERE z < $x  // Now OK!  
  ;  
  CREATE VIEW second AS  
  SELECT a, b, c  
  FROM t  
  WHERE z < $x  |

### 3.4 Miscellaneous ODB/SQL language elements

#### 3.4.1 RESET ALIGN-statement

```plaintext
RESET ALIGN;
```

#### 3.4.2 RESET ONELOOPER-statement

```plaintext
RESET ONELOOPER;
```
3.4.3 ALIGN-statement

ALIGN( table_name_list );

3.4.4 ONELOOPER-statement

ONELOOPER( table_name_list );

3.4.5 Preprocessing keywords

ODB/SQL has limited capability to emulate the C-preprocessor (CPP). Only the following statements are currently supported:

```
#include "filename"
#include ’filename’
#define NAME
#ifdef NAME
    conditional_ddl_or_sql_line(s)
#endif
```

Please note that the hash-character (#) must start from the first column of a line, but can follow any number of whitespace (TABs or SPACEs) before the preprocessing keyword.

3.4.6 Comment lines

```
// Double-slash starts a comment
-- Double-dash followed by a blank also starts a comment
CREATE TABLE hdr AS //← A comment start and ends here →
(a int, -- also a comment (until end of line)
    -- A blank after the double-dash is crucial !!
```

3.5 Built-in functions

Presently built-in functions can only be used in WHERE-statement. Currently implemented functions are shown in the following table:
### 3.5 Built-in functions

Table 4: ODB/SQL built-in functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tdiff(tgdate, tgtime, andate, antime)</code></td>
<td>Time difference in seconds (<code>tgdate,tgtime</code>) are target times from which the nominal analysis timestamp (<code>andate,antime</code>) is getting subtracted from</td>
</tr>
<tr>
<td><code>twindow(tgdate, tgtime, andate, antime, lmargin, rmargin)</code></td>
<td>Time window (range) selection around the nominal analysis timestamp. Arguments as in <code>tdiff</code>, but in addition (<code>lmargin</code>, <code>rmargin</code>) denote left and right offsets in seconds from the nominal analysis timestamp</td>
</tr>
<tr>
<td><code>cos(x)</code></td>
<td>Cosine (where <code>x</code> is in degrees)</td>
</tr>
<tr>
<td><code>sin(x)</code></td>
<td>Sine (where <code>x</code> is in degrees)</td>
</tr>
<tr>
<td><code>tan(x)</code></td>
<td>Tangent (where <code>x</code> is in degrees)</td>
</tr>
<tr>
<td><code>acos(x)</code></td>
<td>Arc cosine (the result is in degrees)</td>
</tr>
<tr>
<td><code>asin(x)</code></td>
<td>Arc sine (the result is in degrees)</td>
</tr>
<tr>
<td><code>atan(x)</code></td>
<td>Arc tangent (the result is in degrees)</td>
</tr>
<tr>
<td><code>atan2(x,y)</code></td>
<td>Arc tangent of two variables (the result in degrees)</td>
</tr>
<tr>
<td><code>exp(x)</code></td>
<td>Exponent</td>
</tr>
<tr>
<td><code>ldexp(x,n)</code></td>
<td>$x \times 2^n$</td>
</tr>
<tr>
<td><code>cosh(x)</code></td>
<td>Hyperbolic cosine</td>
</tr>
<tr>
<td><code>sinh(x)</code></td>
<td>Hyperbolic sine</td>
</tr>
<tr>
<td><code>tanh(x)</code></td>
<td>Hyperbolic tangent</td>
</tr>
<tr>
<td><code>int(x), trunc(x)</code></td>
<td>Truncate to integer</td>
</tr>
<tr>
<td><code>nint(x)</code></td>
<td>Round-up to integer</td>
</tr>
<tr>
<td><code>uint(x)</code></td>
<td>Convert to unsigned integer</td>
</tr>
<tr>
<td><code>float(x)</code></td>
<td>Convert to 32-bit floating point number</td>
</tr>
<tr>
<td><code>dble(x)</code></td>
<td>Convert to 64-bit floating point number</td>
</tr>
<tr>
<td><code>floor(x)</code></td>
<td>Calculate floor value</td>
</tr>
<tr>
<td><code>ceil(x)</code></td>
<td>Calculate ceiling value</td>
</tr>
<tr>
<td><code>abs(x)</code></td>
<td>Absolute value</td>
</tr>
<tr>
<td><code>max(x,y)</code></td>
<td>Maximum value of two arguments</td>
</tr>
<tr>
<td><code>min(x,y)</code></td>
<td>Minimum value of two arguments</td>
</tr>
<tr>
<td><code>touch(x)</code></td>
<td>No-op (a dummy operation) function on <code>x</code> in order to force loading of the variables found in expression. Always returns 1, though</td>
</tr>
<tr>
<td><code>maxcount(x)</code></td>
<td>Returns 1, when the row count (per pool) does not exceed value of <code>x</code></td>
</tr>
<tr>
<td><code>maxrows(x)</code></td>
<td>Returns the current row count (per pool)</td>
</tr>
<tr>
<td><code>ibits(x,pos,len)</code></td>
<td>Extract <code>len</code> bits from <code>word</code> offset <code>pos</code></td>
</tr>
<tr>
<td><code>mod(x,y)</code> or <code>x\%y</code></td>
<td><code>x</code> modulo <code>y</code> with floating point numbers accepted</td>
</tr>
<tr>
<td><code>pow(x,y)</code> or <code>x**y</code> or <code>x^y</code></td>
<td><code>x</code> power of <code>y</code> (<code>$x^y$</code>)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>sqrt(x)</td>
<td>Square root ($\sqrt{x}$)</td>
</tr>
<tr>
<td>log(x), ln(x)</td>
<td>Natural logarithm</td>
</tr>
<tr>
<td>log10(x), lg(x)</td>
<td>Base-10 logarithm</td>
</tr>
<tr>
<td>deg2rad(x), degrees(x)</td>
<td>Convert degrees to radians</td>
</tr>
<tr>
<td>rad2deg(x), radians(x)</td>
<td>Convert radians to degrees</td>
</tr>
<tr>
<td>circle(x,x0,y,y0,r)</td>
<td>Returns 1, if the point $(x,y)$ is inside (or on) the (planar) circle with origin at $(x0,y0)$ and radius $r$</td>
</tr>
<tr>
<td>rad(lat0, lon0, angle, lat, lon)</td>
<td>Returns 1, if the $(lat,lon)$ is inside the great-circle (of any radius) with origin $(lat0,lon0)$, and view angle (in degrees)</td>
</tr>
<tr>
<td>distance(lat1, lon1, lat2, lon2)</td>
<td>Distance (expressed in kilometres) between two positions $(lat1,lon1)$ and $(lat2,lon2)$ along a great-circle of the Earth ($R = 6371$km)</td>
</tr>
<tr>
<td>km(lat1, lon1, lat2, lon2)</td>
<td>Synonym of distance-function</td>
</tr>
<tr>
<td>dist(lat1, lon1, lat2, lon2)</td>
<td>Synonym of distance-function, but the result expressed in metres</td>
</tr>
<tr>
<td>paral(PEvar, targetPE)</td>
<td>Special shuffle function (more ???)</td>
</tr>
</tbody>
</table>
4 ODB Fortran90 access layer

Unlike in conventional relational databases, the data manipulation (data insertion, update, removal) has currently been organized under the shelter of the Fortran90 layer. This is a set of Fortran90 module based user callable functions. Data layout creation and making queries has been left to the non-Fortran90 layer i.e. ODB/SQL at present.

There are reasons for having the Fortran90 access library. The ODB was originally designed to be used from within IFS 4DVAR system only, which is solely written in Fortran90 language. Therefore it was natural to be able to access database using a Fortran90 subroutine layer rather than something else.

As discussed before the ODB software layers underneath are mostly written in C; also the ODB/SQL compilation produces C-code to be linked with the user application. Therefore it would be rather simple to extend the access ODB from C/C++ in the future. Also implementing a client-server access to the ODB is a feasible option.

4.1 Compiling and linking ODB-access programs

An application using the ODB Fortran90 layer can be compiled into a sequential or parallel program. How it relates to your application is described in the following two subsections.

4.1.1 Creating sequential Fortran90 programs

Figure 13: Using the odbf90 -command to build a sequential executable
In the sequential ODB-access the MPI-library layer has been replaced with a dummy MPI-layer which effectively disables any distributed memory data handling through the message passing interface.

The advantage of using just a single processor executable is that one does not have worry about existence of parallel message passing subsystem nor how it schedules its jobs.

The disadvantage is that multiprocessor access to the partitioned (multipool) database is denied and thus making data queries and extraction potentially much slower than is necessary.

### 4.1.2 Creating MPI-parallel Fortran90 programs

![Diagram: Using the odbf90mp -command to build a MPI-parallel executable](image)

By enabling MPI-parallelism in ODB-access, the application program can simultaneously access different partitions (pools) of data. In this approach the native (and normally system specific high performance) MPI-library takes over. You only need to worry about how your parallel job gets resources and is scheduled. Script `mpirun` helps in this respect. This is the default mode for IFS 4D-Var-runs, and thus ODB takes full benefit of this parallel feature by piggy-backing on top of MPI.

### 4.2 Description of the ODB Fortran90 functions

In order to access ODB Fortran90 from your source code, you need to specify the following USE-statement in your Fortran90 program or subroutine:
4.2 Description of the ODB Fortran90 functions

SUBROUTINE mysub(...)
  ...
  USE odb_module
  ...
  implicit none
  ...
END SUBROUTINE mysub

The **USE odb_module** statement declares all prototypes of ODB Fortran90 layer and thus gives extra protection against possible misuse of arguments.

Thorough description about the functions and their argument lists is given in the following subsections. The source files reside under the directories \$SRC/odb/module or \$SRC/odb/include. To summarize, the most commonly used functions are described in the table 5.

SUBROUTINE mysub(...)
  ...
  USE odb_module
  ...
  implicit none
  ...
  INTEGER(4) :: nrows, ncols, nra
  REAL(8), ALLOCATABLE :: x(:,:)
  character(len=64) queryname
  ...
  queryname = 'query' ! Based on CREATE VIEW query AS SELECT ...
  do jp=myproc,npools,nproc
    rc = ODB_select(h,queryname,nrows,ncols,nra=nra,poolno=jp)
    ALLOCATE(x(nra,0:ncols))
    rc = ODB_get(h,queryname,x,nrows,ncols,poolno=jp)
    ...! Work with "x" (access data, update, etc.)
    rc = ODB_put(h,queryname,x,nrows,ncols,poolno=jp)
    DEALLOCATE(x)
    rc = ODB_cancel(h,queryname,poolno=jp)
    ...
  enddo
  ...
END SUBROUTINE mysub

After **ODB_select**, the user normally allocates memory for a data matrix, where the subsequent **ODB_get** will place the data. After **ODB_put** this space must be released by the user. The following illustrates the usage. It serves also as a skeleton program for all ODB data extractions and updates which involve just one table. Please note that the type of the
Table 5: Most important ODB Fortran90 functions.

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Function name</th>
<th>Source .F90/.h-file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize ODB processing</td>
<td>ODB_init (4.2.10)</td>
<td>odb.F90</td>
</tr>
<tr>
<td>Open database</td>
<td>ODB_open (4.2.11)</td>
<td>odb.F90</td>
</tr>
<tr>
<td>Add more pools</td>
<td>ODB_addpools (4.2.1)</td>
<td>odb.F90</td>
</tr>
<tr>
<td>Register SQL-query</td>
<td>ODB_addview (4.2.2)</td>
<td>odb.F90</td>
</tr>
<tr>
<td>Execute SQL-query</td>
<td>ODB_select (4.2.16)</td>
<td>odb.F90</td>
</tr>
<tr>
<td>Fetch data behind the SQL</td>
<td>ODB_get (4.2.6)</td>
<td>fodb.h</td>
</tr>
<tr>
<td>Update data back to database</td>
<td>ODB_put (4.2.13)</td>
<td>fodb.h</td>
</tr>
<tr>
<td>Unselect (cancel) the query</td>
<td>ODB_cancel (4.2.3)</td>
<td>odb.F90</td>
</tr>
<tr>
<td>Execute SQL and place the result into a file</td>
<td>ODB_print (4.2.12)</td>
<td>odbprint.F90</td>
</tr>
<tr>
<td>Remove table(s) of a pool from memory</td>
<td>ODB_swapout (4.2.18)</td>
<td>odb.F90</td>
</tr>
<tr>
<td>Release the whole pool from memory</td>
<td>ODB_release (4.2.14)</td>
<td>odb.F90</td>
</tr>
<tr>
<td>Get column names/types of view/table</td>
<td>ODB_getnames (4.2.7)</td>
<td>odbutil.F90</td>
</tr>
<tr>
<td>Get column index number for column(s) of view/table</td>
<td>ODB_varindex (4.2.20)</td>
<td>odbutil.F90</td>
</tr>
<tr>
<td>Get view/table size/dimension information</td>
<td>ODB_getsize (4.2.8)</td>
<td>odb.F90</td>
</tr>
<tr>
<td>Get current value of parameterized $-variable</td>
<td>ODB_getval (4.2.9)</td>
<td>odbutil.F90</td>
</tr>
<tr>
<td>Change the value of parameterized $-variable</td>
<td>ODB_setval (4.2.17)</td>
<td>odbutil.F90</td>
</tr>
<tr>
<td>Test existence of parameterized $-variable</td>
<td>ODB_varexist (4.2.19)</td>
<td>odbutil.F90</td>
</tr>
<tr>
<td>Remove data from a table</td>
<td>ODB_remove (4.2.15)</td>
<td>odb.F90</td>
</tr>
<tr>
<td>Close database</td>
<td>ODB_close (4.2.4)</td>
<td>odb.F90</td>
</tr>
<tr>
<td>Finish with ODB</td>
<td>ODB_end (4.2.5)</td>
<td>odb.F90</td>
</tr>
</tbody>
</table>

ALLOCATABLE Fortran90-matrix (variable x, which will contain the data) must be either REAL (8) or INTEGER (4).\(^7\) Call to ODB_get will abort if any columns in the associated SQL-query contain floating point (or string) types, but variable x was declared as INTEGER (4). You can find the datatypes of every column by using function ODB_getnames, but safest and easiest is to use REAL (8) type all the time.

The following sections explain ODB data manipulation functions in greater detail. For easier navigation, the routines are found in alphabetical rather than in functional order.

### 4.2.1 ODB_addpools

Usage of the function ODB_addpools is as follows:

\(^7\)INTEGER(4) will be phased out in the future versions of the ODB software
This function adds more pools (\texttt{pool\_excess}) – globally – to the existing database and returns total number of pools after the addition. The function is global i.e. all participating MPI-tasks must call it. Furthermore, no outstanding data queries must be in progress i.e. they all must have been completed (ODB\_cancel should have been called).

The parameters denote:

\begin{itemize}
  \item \texttt{rc} \hspace{1cm} \texttt{INTEGER(4)}
    \text{Return code, which upon successful completion will be } \geq 0
  \item \texttt{handle} \hspace{1cm} \texttt{INTEGER(4), \text{INTENT(IN)}}
    \text{Database handle as returned by ODB\_open}
  \item \texttt{pool\_excess} \hspace{1cm} \texttt{INTEGER(4), \text{INTENT(IN)}}
    \text{The number of new pools to be added – globally, not per MPI-task}
  \item \texttt{old\_npools} \hspace{1cm} \texttt{INTEGER(4), \text{INTENT(OUT), OPTIONAL}}
    \text{Returns global number of pools before the addition}
  \item \texttt{maxpoolno} \hspace{1cm} \texttt{INTEGER(4), \text{INTENT(OUT), OPTIONAL}}
    \text{Specifies the highest pool number used in data queries. The reason for using this parameter is that we may know the total amount of pools containing meaningful data is (say) \texttt{old\_npools} and we want to stick with this. However, if the number of MPI-tasks used for this run is larger than \texttt{old\_npools}, then we must add \texttt{pool\_excess} number of "phoney" pools (which we may not necessarily contain any data), so that the total number of pools is at least equal to MPI-tasks. Then we set \texttt{maxpoolno} to be equal to \texttt{old\_npools} and ODB will never search (in this run) beyond this pool number (on the database pointed to by the handle). Consequently the "phoney" pools are also never processed}
\end{itemize}

\textbf{Examples:}

\begin{verbatim}
rc = ODB_addpools ( handle, pool_excess & & , [old_npools], [maxpoolno] )
\end{verbatim}

\begin{verbatim}
rc = ODB_addpools ( handle, 0, old_npools=npools )
\end{verbatim}

\begin{verbatim}
rc = ODB_addpools ( handle, 3, maxpoolno=npools )
\end{verbatim}
! Just add 1 pool; no limits for data queries
rc = ODB_addpools(handle, 1)

4.2.2 ODB_addview

Usage of the function ODB_addview is as follows:

```
rc = ODB_addview ( handle, dtname & 
& [viewfile], [select], [uniqueby], [from], [where] & 
& [orderby], [sortby], [query], [set], [abort] )
```

Registers SQL-query dtname into the running ODB application so that subsequent query execution in ODB_select can take place. Note that in the later releases of ODB-software it is not necessary to use ODB_addview anymore, since when ODB_select is applied first time to dtname, it will issue an automatic ODB_addview Note also that this function is applicable straight away to all participating pools i.e. no poolno parameter is available. (Please note that the dynamic linking mode is a deprecated feature and is not actively supported in the ODB software anymore).

The parameters denote:

- **rc**
  - INTEGER(4)
  - Return code. A positive number indicates successful completion. It also defines the internal view number for this data query, which can be used to refer the query in the future ODB software releases instead of character string dtname.

- **handle**
  - INTEGER(4), INTENT(IN)
  - Database handle as returned by ODB_open.

- **dtname**
  - CHARACTER(LEN= *), INTENT(IN)
  - Query name (in lower case letter) as specified in the CREATE VIEW statement. Usually points also to the SQL-query file dtname.sql; sometimes even to database specific file $dbname_dtname.sql.

- **viewfile**
  - CHARACTER(LEN=*), INTENT(IN), OPTIONAL
  - This parameter is applicable in dynamic linking mode only (i.e. when ODB_STATIC_LINKING is set to 0). In dynamic mode you can supply SQL-query file (say dtname.sql) and the system will compile and dynamically link it into the currently running executable by using shareable object linking mechanism.
4.2 Description of the ODB Fortran90 functions

**select**

CHARACTER(LEN=*), INTENT(IN), OPTIONAL

*This parameter is applicable in dynamic linking mode only.* Instead of using SQL-query file, you will be able to supply the **SELECT** ... **FROM** etc. statements directly through this Fortran90 interface and they will be put into file `dbname.sql` and compiled like in the case of parameter **viewfile** above.

**uniqueby**

CHARACTER(LEN=*), INTENT(IN), OPTIONAL

*This parameter is applicable in dynamic linking mode only.* You can supply (ODB/SQL-specific) **UNIQUEBY** keyword with this parameter.

**from**

CHARACTER(LEN=*), INTENT(IN), OPTIONAL

*This parameter is applicable in dynamic linking mode only.* You can supply **FROM** keyword with this parameter. Assumes that **SELECT** is also present.

**where**

CHARACTER(LEN=*), INTENT(IN), OPTIONAL

*This parameter is applicable in dynamic linking mode only.* You can supply (an optional) **WHERE** keyword with this parameter. The routine checks that **SELECT** and **FROM** have also been supplied.

**orderby**

CHARACTER(LEN=*), INTENT(IN), OPTIONAL

*This parameter is applicable in dynamic linking mode only.* You can supply (an optional) **ORDERBY** keyword with this parameter. You cannot use both **ORDERBY** and **SORTBY** simultaneously.

**sortby**

CHARACTER(LEN=*), INTENT(IN), OPTIONAL

*This parameter is applicable in dynamic linking mode only.* You can supply (an optional and ODB/SQL-specific) **SORTBY** keyword with this parameter. This parameter is mutually exclusive with **ORDERBY**.

**query**

CHARACTER(LEN=*), INTENT(IN), OPTIONAL

*This parameter is applicable in dynamic linking mode only.* You can supply the whole SQL-query string (without CREATE VIEW `dbname`) and thus simplify the subroutine call. It will be placed into file `dbname.sql` and compiled similarly as discussed in the context of parameter **viewfile**.

**set(:)**

CHARACTER(LEN=*), INTENT(IN), OPTIONAL

*This parameter is applicable in dynamic linking mode only.* For parameterized queries you can supply an array of **SET**-variable definitions. Due to a Fortran90 restriction each of such character elements in the array must have exactly the same length.

**abort**

LOGICAL, INTENT(IN), OPTIONAL

By default if view registration fails, ODB aborts the program. By setting
this parameter to \texttt{.FALSE.}, abort is prevented. This option is useful to check existence (availability) of a particular query.

\textbf{Examples:}

\begin{verbatim}
! Register view 'queryname' to the running program
! Abort if not found (the default)
INTEGER(4) :: rc, handle
rc = ODB_addview(handle, 'queryname')

! The same as previous, but do not abort if fails to register
rc = ODB_addview(handle, 'queryname', abort=.FALSE.)
\end{verbatim}

\subsection{4.2.3 ODB_cancel}

Usage of the function \texttt{ODB_cancel} is as follows:

\begin{verbatim}
rc = ODB_cancel ( handle, dtname &
& , [poolno] )
\end{verbatim}

This function the opposite of \texttt{ODB_select} i.e. it deallocates all internal index space reserved for the query \texttt{dtname}. After calling this routine you cannot access (fetch through \texttt{ODB_get}) the data belonging to this query unless you re-execute its counterpart \texttt{ODB_select} again. The parameters denote:

\begin{itemize}
  \item \texttt{rc} INTEGER(4) \quad Return code, which upon successful completion will be \texttt{0}
  \item \texttt{handle} INTEGER(4), INTENT(IN) \quad Database handle as returned by \texttt{ODB_open}
  \item \texttt{dtname} CHARACTER(LEN=*), INTENT(IN) \quad Query name (in lower case letter) as specified in the \texttt{CREATE VIEW} statement. Usually points also to the SQL-query file \texttt{dtname.sql}; sometimes even to database specific file \texttt{Sdbname_dname.sql}
  \item \texttt{poolno} INTEGER(4), INTENT(IN), OPTIONAL \quad The pool number. If omitted, the default $= -1$ will be assumed i.e. all applicable pools owned by the particular MPI-task will be scanned
\end{itemize}

\textbf{Examples:}
4.2 Description of the ODB Fortran90 functions

4.2.4 ODB_close

Usage of the function ODB_close is as follows:

```fortran
rc = ODB_close ( handle, [save] )
```

This function closes connection to the opened database and optionally updates associated data into the disk-media. The parameters denote:

- **rc**
  - INTEGER(4)
  - Return code, which upon successful completion will be \( \geq 0 \)

- **handle**
  - INTEGER(4), INTENT(IN)
  - Database handle as returned by ODB_open

- **save**
  - LOGICAL, INTENT(IN), OPTIONAL
  - If this parameter is present and was set to .TRUE., and if database was not opened in 'READONLY'-mode, then the routine will update database tables and associated metadata to disk-media before closing the connection to the database pointed to by the handle. In any other circumstances, or if this option is not present, the database is not be saved to disk. At the end of the call internal data structures related to this database-handle will also be wiped out from memory.

Examples:

```fortran
! Close database WITHOUT saving (the default)
INTEGER(4) :: handle, rc
rc = ODB_close(handle)

! Save changes (assume not a read-only database) and close
rc = ODB_close(handle, save=.TRUE.)
```
4.2.5 ODB_end

Usage of the function ODB_end is as follows:

```fortran
rc = ODB_end()
```

Finishes ODB-processing by closing connections to all databases that are still open and terminates MPI-parallel processing. Because of this, the program may also terminate, but this is a system specific behaviour. Anyway, this should be the last function call in your program.

The parameters denote:

- `rc`: INTEGER(4)
  - Return code, with value of 0 indicating success

Examples:

```fortran
! Finish with ODB and end parallel processing
! Note: This call may terminate your program
! on some platforms, so don’t expect to
! perform any operations after this call!
INTEGER(4) :: rc
rc = ODB_end()
```

4.2.6 ODB_get

Usage of the function ODB_get is as follows:

```fortran
rc = ODB_get ( handle, dtname, d, nrows &
&    , [ncols], [poolno], [colget], [colpack], [index] &
&    , [sorted] )
```

This function copies data from database, SQL-query or tablename dtname, into the user supplied dynamically allocated data array/matrix d. Currently DATATYPE of the array has to be either INTEGER(4) or REAL(8). In the subsequent release we may have to stick with REAL(8) datatype only.

The data is a complete TABLE of a particular pool, or result of SQL-query dtname executed through the function ODB_select.

The parameters denote:

- `rc`: INTEGER(4)
  - Return code, with value of 0 indicating success
- `handle`: INTEGER(4)
  - Database handle
- `dtname`: CHARACTER(100)
  - Database or table name
- `d`: REAL(8,*)
  - User supplied dynamically allocated data array/matrix
- `nrows`: INTEGER(4)
  - Number of rows to read
- `ncols`: INTEGER(4)
  - Number of columns to read
- `poolno`: INTEGER(4)
  - Pool number
- `colget`: INTEGER(4)
  - Column get option
- `colpack`: INTEGER(4)
  - Column pack option
- `index`: INTEGER(4)
  - Index option
- `sorted`: INTEGER(4)
  - Sorted option
4.2 Description of the ODB Fortran90 functions

rc
INTEGER(4)
Return code, which upon successful completion returns the number of rows actually put into array d

handle
INTEGER(4), INTENT(IN)
Database handle as returned by ODB_open

dtname
CHARACTER(LEN=*) , INTENT(IN)
Query name (in lower case letter) as specified in the CREATE VIEW-statement. Usually points also to the SQL-query file dtname.sql; sometimes even to database specific file $dbname_dtname.sql

d(:,0:)
DATATYPE, INTENT(INOUT)
Array or data matrix, where the outcome of the SQL-request (or whole table) dtname will copied. Note that the columns start at zero, since some internal information needs to stored for subsequent ODB_put

nrows
INTEGER(4), INTENT(INOUT)
Upon return indicates number of rows actually copied into the data matrix d. The value can be < the value upon input, depending on the nature of SQL-request; for example some duplicates may have been eliminated due to SELECT DISTINCT-statement in CREATE VIEW

ncols
INTEGER(4), INTENT(INOUT), OPTIONAL
Upon return indicates number of columns actually copied into the data matrix d. The value can be < the value upon input, depending on the nature of SQL-request; for example some ghost-columns may have been eliminated before copying data to the array (not yet implemented)

poolno
INTEGER(4), INTENT(IN), OPTIONAL
The pool number. If omitted, the default = 1 will be assumed i.e. all applicable pools owned by the particular MPI-task will be scanned

colget(:)
LOGICAL, INTENT(IN), OPTIONAL
Optionally only a subset of columns of the matrix d could be filled/copied by supplying .TRUE.-values in a logical-vector (of length ≥ ncols). This parameter has nothing to do with the famous toothpaste, I think

colpack(:)
LOGICAL, INTENT(IN), OPTIONAL
Selective (internal) column packing can be requested at the end of the function. Purpose of this option is to allow memory saving by packing associated column data internally just before returning from ODB_get. However, its use probably leads into serious memory fragmentation and performance degradation, if used too frequently. If used, the length of this logical-vector has to be ≥ ncols
**index(:)**

INTEGER(4), INTENT(OUT), OPTIONAL
When SQL-request asks data to be delivered in ordered manner (i.e. using ORDERBY (or SORTBY)-statements), the physical sort of the data matrix \( d \) can be prevented by supplying index-vector of length \( \geq n_{rows} \), where the ordering information will be stored. This means that ordered data access needs to be performed via \( d(\text{index}(j),icol) \), and not directly through \( d(j,icol) \).

**sorted**

LOGICAL, INTENT(IN), OPTIONAL
Possible use of ORDERBY (or SORTBY)-statements can be ignored by using this optional parameter and setting it to .FALSE.

**Examples:**

```! A prime example of ODB-usage :
! Execute data query, allocate matrix and load it with data
!
! Please note that you need column#0 for internal info
```

``` INTEGER(4) :: rc, handle
INTEGER(4) :: nrows, ncols, nra, poolno
REAL(8), allocatable :: x(:,:)
```

``` poolno = -1
rc = ODB_select(handle,'query',nrows,ncols,nra=nra,poolno=poolno)
allocate(x(nra,0:ncols))
rc = ODB_get(handle,'query',x,nrows,ncols=nra,poolno=poolno)
do jrows=1,nrows
   do jcols=1,ncols
      print *,x(jrows,jcols)
   enddo
endo
```

### 4.2.7 ODB_getnames

Usage of the function `ODB_getnames` is as follows:

``` rc = ODB_getnames ( handle, dname, mode &
&          , [outnames] ) ```
4.2 Description of the ODB Fortran90 functions

This is a multipurpose function, which returns table names, view (data query) names, or data column names or their types, depending on mode associated with the database behind its handle. This function is a close relative to ODB_varindex, which returns data column indices per their names.

This function is typically called twice in succession. The first call is invoked without the outnames-parameter to get the dimension of the information (i.e. size for outnames), stored in the return code rc. The second call will be called with outnames correctly dimensioned as from previous return code.

The parameters denote:

- **rc** INTEGER(4) Return code, which upon successful completion will be ≥ 0
- **handle** INTEGER(4), INTENT(IN) Database handle as returned by ODB_open
- **dtname** CHARACTER(LEN=*) , INTENT(IN) Query name (in lower case letter) as specified in the CREATE VIEW-statement. Usually points also to the SQL-query file dtname.sql; sometimes even to database specific file $dbname_dtname.sql
- **mode** CHARACTER(LEN=*) , INTENT(IN) A character string, which determines the type of information the function should return in character array outnames. The mode can be as follows:

<table>
<thead>
<tr>
<th>mode</th>
<th>Returned info</th>
</tr>
</thead>
<tbody>
<tr>
<td>'datatype', 'type', 'ctype'</td>
<td>Internal ODB datatype</td>
</tr>
<tr>
<td>'ftndatatype', 'ftntype'</td>
<td>Fortran90 datatype</td>
</tr>
<tr>
<td>'varname', 'name'</td>
<td>Fully qualified column names (like obstype@hdr)</td>
</tr>
<tr>
<td>'tablename', 'table'</td>
<td>Returns all the table names (when dtname is '*')</td>
</tr>
<tr>
<td>'viewname', 'view'</td>
<td>Returns all those SQLs that have been registered directly or indirectly via ODB_addview</td>
</tr>
</tbody>
</table>

- **outnames(:)** CHARACTER(LEN=*) , INTENT(OUT), OPTIONAL Character string vector containing the information requested by the mode

**Examples:**
INTEGER(4) :: handle, ninfo
CHARACTER(LEN=128), allocatable :: clinfo(:)

! Get column names (in a form 'colname@tablename')
ninfo = ODB_getnames(handle,'queryname','name')
allocate(clinfo(ninfo))
ninfo = ODB_getnames(handle,'queryname','name',&
   & outnames=clinfo)
do j=1,ninfo
   print *, 'column name#', j, '=' ', trim(clinfo(j))
enddo
deallocate(clinfo)

! Get table names
ninfo = ODB_getnames(handle,'*','table')
allocate(clinfo(ninfo))
ninfo = ODB_getnames(handle,'*','table',&
   & outnames=clinfo)

! Get registered view names
ninfo = ODB_getnames(handle,'*','view')
allocate(clinfo(ninfo))
ninfo = ODB_getnames(handle,'*','view',&
   & outnames=clinfo)

4.2.8 ODB_getsize

Usage of the function ODB_getsize is as follows:

rc = ODB_getsize ( handle, dtname, nrows, ncols &
   & [nra], [poolno] )

This function returns the number of rows (nrows) and columns (ncols) behind the data query or table dtname.
The parameters denote:

rc
   INTEGER(4)
   Return code which upon successful completion return value ≥ 0, which is the same as the nrows
4.2  Description of the ODB Fortran90 functions

**handle**

*INTEGER(4), INTENT(IN)*

Database handle as returned by ODB_open

**dtname**

*CHARACTER(LEN=*), INTENT(IN)*

This parameter is normally a tablename (a character string beginning with @-sign) or SQL-request (viewname). In case of SQL-request, this routine just returns the dimensions of the latest query, if still active i.e. ODB_cancel had not been called since the previous ODB_select on dtname

**nrows**

*INTEGER(4), INTENT(OUT)*

The number of rows behind the dtname

**ncols**

*INTEGER(4), INTENT(OUT)*

The number of columns behind the dtname

**nra**

*INTEGER(4), INTENT(OUT), OPTIONAL*

Optimal number of rows that ought to be allocated for subsequent data matrix in order to reduce memory bank conflicts on vector machines or cache-line misses on RISC-architectures. The value is guaranteed to be \( \geq \) nrows, but the exact value is computer platform dependent.

**poolno**

*INTEGER(4), INTENT(IN), OPTIONAL*

The pool number. If omitted, the default = -1 will be assumed i.e. all applicable pools owned by the particular MPI-task will be scanned

*Examples:*  

```fortran
! Get current size of table '@hdr' at pool number 9
INTEGER(4) :: rc, handle
INTEGER(4) :: nrows, ncols
rc = ODB_getsize(handle,'@hdr',nrows,ncols,poolno=9)

! The same as previous, but for all (locally owned) pools
rc = ODB_getsize(handle,'@hdr',nrows,ncols)
! -- or :
rc = ODB_getsize(handle,'@hdr',nrows,ncols,poolno=-1)

! Get dimensions of currently active data 'query' i.e.
! such query on which ODB_select has been executed,
! but not yet ODB_cancel
rc = ODB_getsize(handle,'query',nrows,ncols)

! The same as previous, but get optimal leading dimension
```
INTEGER(4) :: nlda
REAL(8), allocatable :: array(:,:,)
rc = ODB_getsize(handle,'query',nrows,ncols,nra=nlda)
allocate(array(nlda,0:ncols))

4.2.9 ODB_getval

Usage of the function ODB_getval is as follows:

```
value = ODB_getval ( handle, varname & & [viewname] )
```

Returns the current value of $\text{variable varname}$ which can optionally belong to SQL-query $\text{viewname}$ only. Unlike its counterpart function ODB_setval (see 4.2.17), this function always returns type REAL(8) value.

The parameters denote:

- **value** REAL(8)
  The current value of the $\text{variable varname}$.

- **handle** INTEGER(4), INTENT(IN)
  Database handle as returned by ODB_open.

- **varname** CHARACTER(LEN=*), INTENT(IN)
  The $\text{variable as specified via SET-statement either in data definition layout (ddl-file) or SQL-query. In the latter case, the viewname must be present.}$

- **viewname** CHARACTER(LEN=*), INTENT(IN), OPTIONAL
  The optional SQL-query name.

**Examples:**

```
! Get the number of updates as defined in the layout
INTEGER(4) :: handle
INTEGER(4) :: imxupd
imxupd = ODB_getval(handle,'$nmxupd')
```

```
! Given the following data query:
```
4.2 Description of the ODB Fortran90 functions

SET $flp = 1.234;
CREATE VIEW jackpot AS
    SELECT etc. ...

! Get floating point parameter value associated with it
REAL(8) :: value
value = ODB_getval(handle, '$flp', viewname='jackpot')

4.2.10 ODB_init

Usage of the function ODB_init is as follows:

rc = ODB_init ( [myproc], [nproc], [pid], [tid], [ntid] )

Initializes ODB processing for Fortran90 interface and creates the necessary MPI-processes and OpenMP-threads, if necessary. This is normally the very first call to ODB-library. The source code resides in module file $SRC/odb/module/odb.F90. The parameters denote:

rc  INTEGER(4)
    Return code, which upon successful completion returns zero

myproc  INTEGER(4), INTENT(OUT), OPTIONAL
    Process id as returned by the ECMWF MPL-library. The value should always be between range 1..nproc

nproc  INTEGER(4), INTENT(OUT), OPTIONAL
    Number of MPI-processes allocated by the ECMWF MPL-library

pid  INTEGER(4), INTENT(OUT), OPTIONAL
    Unix process id of the caller

tid  INTEGER(4), INTENT(OUT), OPTIONAL
    OpenMP thread id as returned by the ECMWF OML-library. The value should always be in the range 1..ntid

ntid  INTEGER(4), INTENT(OUT), OPTIONAL
    Number of OpenMP-threads allocated

Examples:
! Initialize ODB and (possibly) parallel processing
INTEGER(4) :: rc
rc = ODB_init()

! Get parallel processing parameters, too
INTEGER(4) :: myproc, nproc ! MPI (MPL)
INTEGER(4) :: numthreads, tid ! OpenMP (OML) threads
INTEGER(4) :: pid ! Unix process id of the caller
rc = ODB_init(myproc=myproc, nproc=nproc, &
   & numthreads, tid=tid, pid=pid)

4.2.11 ODB_open

Usage of the function ODB_open is as follows:

handle = ODB_open ( dbname, status, npools &
   & [old_npools], [maxpoolno] )

Opens connection to a database dbname with opening mode status. If the database is
new, then npools will be allocated. If the database was previously present, the function
optionally returns how many pools there were just before ODB_open. Another optional argu-
ment maxpoolno indicates the actual maximum pool number to be used for data access.
This can be less than npools. If function ODB_init had not been called yet, ODB_open
will take care of doing so.

The parameters denote:

handle INTEGER(4)
Database handle. Upon successful return this value must be $\geq 1$. This parameter acts like a virtual unit number for opened database and must
be passed as the first argument to most of the ODB Fortran90 layer’s functions

dbname CHARACTER(LEN=*), INTENT(IN)
Database name to be opened. This character string has to be a valid
database name i.e. start with capital letters [A-Z] and contain there-
after only capital letters or numbers i.e. [A-Z0-9]. No underscores
are accepted

status CHARACTER(LEN=*) , INTENT(IN)
Opening mode. Three different modes are accepted: ’NEW’ for opening
a new database, ’OLD’ for opening an existing database for update,
and 'READONLY' for opening an existing database which cannot be updated (even accidentally)

\[ \text{npoools} \] INTEGER(4), INTENT(INOUT)  
Number of pools in database. For new databases this defines how many pools (i.e. partitions) need to be created. The number can be increased later on by calling ODB_addpools-function. For existing database it just returns number of pools present. These may have been adjusted by an internal call to ODB_addpools, since npools is always rounded up to the nearest number of MPI-processors in use.

\[ \text{old_npoools} \] INTEGER(4), INTENT(OUT), OPTIONAL  
Existing number of pools before ODB_open was issued. For a new database this would be zero. For an existing database the npools may have already been adjusted by an internal call to ODB_addpools.

\[ \text{maxpoolno} \] INTEGER(4), INTENT(OUT), OPTIONAL  
Indicates how many pools will be used to perform data queries. The original npools may have been smaller than the number of MPI-processors in use, therefore ODB_addpools-function is called, which makes npools larger than the true number of partitions. The excess of these "dummy" partitions will not be created nor accessed when the database was opened with status='READONLY'. The maxpoolno indicates therefore the true number of pools that contain any data.

**Examples:**

```
! Create new database with 4 pools  
! Any existing data will be removed
INTEGER(4) :: handle, npools
npools = 4
handle = ODB_open(‘MYDB’,’NEW’,npools)

! Open and existing or create a new database,  
! if it doesn’t exist; thus "npoools" may need to be set
npools = 4
handle = ODB_open(‘MYDB’,’UNKNOWN’,npools)
print *,’Number of pools is : ’,npools

! Open an existing database
! If database doesn’t exist, the program will abort
npools = 0
handle = ODB_open(‘DATABAAZ’,’OLD’,npools)
print *,’Number of pools is : ’,npools
```
! Open an existing database for read-only access
! Database must be an existing one
npools = 0
handle = ODB_open('DATABA AZ','READONLY',npools)
print *, 'Number of pools is : ', npools

4.2.12 ODB_print

Usage of the function ODB_print is as follows:

```fortran
rc = ODB_print ( handle, dtname &
    & [file], [poolno], [maxpoolno], [setvars], [values] &
    & [pevar], [replicate_pe], [show_db_index] &
    & [inform_progress], [open_file], [close_file] &
    & [append_mode] )
```

Function ODB_print enables SQL-query to be executed, get data and place it into a result file. Thus this function is a very quick way to create an output file from a data query without need to call ODB_addview, ODB_select, allocate array, ODB_get and ODB_cancel.

The parameters denote:

- **rc**: INTEGER(4)
  Return code, which upon successful completion will be $\geq 0$

- **handle**: INTEGER(4), INTENT(IN)
  Database handle as returned by ODB_open

- **dtname**: CHARACTER(LEN=*), INTENT(IN)
  Query name (in lower case letter) as specified in the CREATE VIEW-statement. Usually points also to the SQL-query file `dtname.sql`; sometimes even to database specific file `$dbname_dtname.sql`.

- **file**: CHARACTER(LEN=*), INTENT(IN), OPTIONAL
  The file name where to write the outcome of the query. If not supplied, then defaults to "`dtname.rpt`".

- **poolno**: INTEGER(4), INTENT(IN), OPTIONAL
  The pool number. If omitted, the default = -1 will be assumed i.e. all applicable pools owned by the particular MPI-task will be scanned.
4.2 Description of the ODB Fortran90 functions

maxpoolno
INTEGER(4), INTENT(IN), OPTIONAL

setvars(:)
CHARACTER(LEN=*), INTENT(IN), OPTIONAL
The same as in ODB_select

values(:)
REAL(8), INTENT(IN), OPTIONAL
The same as in ODB_select

pevar
CHARACTER(LEN=*), INTENT(IN), OPTIONAL
The same as in ODB_select

replicate_pe
INTEGER(4), INTENT(IN), OPTIONAL
The same as in ODB_select

show_db_index
LOGICAL, INTENT(IN), OPTIONAL
Flag to indicate whether to show internal database index (the “famous” 0th column). By default it is not shown

inform_progress
LOGICAL, INTENT(IN), OPTIONAL
If .TRUE., then the first MPI-task will display a rudimentary text-based progress-bar to assure that ODB-software is really doing something

open_file
LOGICAL, INTENT(IN), OPTIONAL
Flag to indicate whether to open the file. Useful only when looping over many pools. See examples. By default .TRUE.

close_file
LOGICAL, INTENT(IN), OPTIONAL
Flag to indicate whether to close the file. Useful only when looping over many pools. See examples. By default .TRUE.

append_mode
LOGICAL, INTENT(IN), OPTIONAL
Flag to indicate whether to open the file in append-mode (concatenation). Useful only when you want to merge results from multiple invocations of ODB_print – or even multiple runs of an application program using ODB_print By default .FALSE.

Examples:

!-- Execute SQL and put the result into file 'query.rpt'
!  All pools involved
INTEGER(4) :: rc, handle
rc = ODB_print(handle, 'query')

!-- Loop over the pools
! Open the file upon first pool, close upon last
! Assume just participating MPI-task
INTEGER(4) :: jp, npools
do jp=1,npools
  rc = ODB_print(handle, 'query', poolno=jp, &
    file='result.rpt', &
    open_file = (jp == 1), &
    close_file = (jp == npools))
enddo ! do jp=1,npools

!-- Append results from two ODB_print's into one file
rc = ODB_print(handle, 'query1',
  file='total.rpt') ! Create
rc = ODB_print(handle, 'query2',
  file='total.rpt', &
  append_mode=.TRUE.) ! Append

!-- Show progress bar for curiosity
rc = ODB_print(handle, 'query', &
  inform_progress=.TRUE.)

4.2.13 ODB_put

Usage of the function ODB_put is as follows:

```
rc = ODB_put ( handle, dtname, d, nrows &
  & ncols, [poolno], [colput], [colpack], [sorted] )
```

This function copies data from user supplied working array/matrix d back into database internal cache. Currently the DATATYPE of the array has to be either INTEGER(4) or REAL(8), but as indicated in the counterpart function’s ODB_get description, this may change in the future. See more from function ODB_get (4.2.6)

Note that this function does not perform any I/O-operations, thus database files are not yet updated. You need to issue a function ODB_swapout, ODB_store, ODB_release or ODB_close to physically update the database on disk. Thus, one can use ODB_put also for READONLY-databases, and it will then update only the in-memory data structures. This could be sometimes useful. One may want to extract data via SQL and perform temporary (in-memory) update for some items in the query. Then another query will be launched using
the modified – now in-memory values. And yet nothing ends up on disk, since database was protected from permanent updates.

The parameters denote:

\begin{itemize}
  \item **rc**\(\text{INTEGER}(4)\)
    Return code, which upon successful completion returns the number of rows actually put from array \(d\) back into database internal cache
  \item **handle**\(\text{INTEGER}(4), \text{INTENT(IN)}\)
    Database handle as returned by \texttt{ODB\_open}
  \item **dtname**\(\text{CHARACTER(LEN=*)}, \text{INTENT(IN)}\)
    Query name (in lower case letter) as specified in the \texttt{CREATE VIEW} - statement. Usually points also to the SQL-query file \texttt{dtname\_sql}; sometimes even to database specific file \texttt{$dbname\_dtname\_sql}\)
  \item **d(:,0:)**\(\text{DATATYPE}, \text{INTENT(INOUT)}\)
    The data array to be copied back into database internal cache
  \item **nrows**\(\text{INTEGER}(4), \text{INTENT(IN)}\)
    The number of rows to be copied
  \item **ncols**\(\text{INTEGER}(4), \text{INTENT(INOUT)}, \text{OPTIONAL}\)
    Upon input the number of columns to be copied. Upon output indicates how many columns actually copied.
  \item **poolno**\(\text{INTEGER}(4), \text{INTENT(IN)}, \text{OPTIONAL}\)
    The pool number. If omitted, the default \(= -1\) will be assumed i.e. all applicable pools owned by the particular MPI-task will be scanned
  \item **colput(:)**\(\text{LOGICAL}, \text{INTENT(IN)}, \text{OPTIONAL}\)
    Similar to \texttt{colget}\(-\)parameter in function \texttt{ODB\_get}. That is to say, optionally only subset of columns can be passed back to database. A logical-vector of length \(\geq \text{ncols}\)
  \item **colpack(:)**\(\text{LOGICAL}, \text{INTENT(IN)}, \text{OPTIONAL}\)
    The same as \texttt{colpack}\(-\)parameter in function \texttt{ODB\_get}.
  \item **sorted**\(\text{LOGICAL}, \text{INTENT(IN)}, \text{OPTIONAL}\)
    When set to \texttt{.TRUE.}, enforces physical sorting of data array \(d\) with respect to column\#0 before passing data columns into database. Normally not supplied since the system automatically detects the need for sorting (for example, in case of putting back local data which stemmed from multiple pools)
\end{itemize}
Examples:

```fortran
! Update database (in-memory), deallocate data matrix and
! finally de-activate the data query

INTEGER(4) :: rc, handle
INTEGER(4) :: nrows, ncols, nra, poolno
REAL(8), allocatable :: x(:,:)

poolno = -1
rc = ODB_put(handle,’query’,x,nrows,ncols=ncols,poolno=poolno)
deallocate(x)
rc = ODB_cancel(handle,’query’,poolno=poolno)
```

### 4.2.14 ODB_release

Usage of the function `ODB_release` is as follows:

```fortran
rc = ODB_release ( handle, [poolno] )
```

Removes all data associated with `poolno` from memory. Unlike its sibling routine `ODB_swapout`, this routine only has an effect when the database has been opened in `ODB_open` with status ‘READONLY’. This routine also works for I/O-method#4 (see `ODB_IO_METHOD` (see p.191)), but then there is no way to request data back into memory, unlike if function `ODB_swapout` with I/O-method#4, unless `ODB_close` followed by `ODB_open` are invoked again.

If no explicit pool number is given and if database has been opened in ‘READONLY’-mode, then all pools will be removed from memory.

The parameters denote:

- `rc` : INTEGER(4)
  
  Return code, which upon successful completion will be ≥ 0

- `handle` : INTEGER(4), INTENT(IN)
  
  Database handle as returned by `ODB_open`

- `poolno` : INTEGER(4), INTENT(IN), OPTIONAL
  
  The pool number. If omitted, the default = −1 will be assumed i.e. all applicable pools owned by the particular MPI-task will be scanned
4.2 Description of the ODB Fortran90 functions

Examples:

```fortran
! Remove every pool of data from memory (without saves to disk)
INTEGER(4) :: rc, handle
rc = ODB_release(handle)

! Remove the first pool of every MPI-task
! i.e. pool number ’myproc’ from memory
INTEGER(4) :: myproc
rc = ODB_init(myproc=myproc)
...
rc = ODB_release(handle, poolno=myproc)
```

### 4.2.15 ODB_remove

Usage of the function `ODB_remove` is as follows:

```fortran
rc = ODB_remove ( handle, dtname &
&                         , [poolno] )
```

This function removes data behind the table `dtname` and marks it empty for subsequent database operations. If the pool number `poolno` is missing, then all the similar tables given in `dtname` will be removed.

**Warning**: use this function only if you know exactly what you are doing or you may severely destroy the integrity of your database.

The parameters denote:

- `rc` INTEGER(4)
  Upon successful return this value indicates the number of bytes removed
- `handle` INTEGER(4), INTENT(IN)
  Database handle as returned by `ODB_open`
- `dtname` CHARACTER(LEN=*) , INTENT(IN)
  Table name (begins with (@-sign) to which the removal operation will be applied. Unlike in many other functions, you presently cannot use the '*'-syntax to indicate all table, nor refer to SQL-queries. That makes use of this dangerous function less attractive i.e. you really need to convince yourself whether it is a good idea to call this function at all!

poolno INTEGER(4), INTENT(IN), OPTIONAL
The pool number. If omitted, the default $=-1$ will be assumed i.e. all applicable pools owned by the particular MPI-task will be scanned.

Examples:

```fortran
! Remove permanently table '@zztop' from database
! All pools
INTEGER(4) :: rc, handle
rc = ODB_remove(handle,'@zztop')

! The same as previous, but only pool number 13 will be removed
rc = ODB_remove(handle,'@zztop', poolno=13)
```

### 4.2.16 ODB_select

Usage of the function ODB_select is as follows:

```fortran
rc = ODB_select ( handle, dtname, nrows, ncols &
& , [nra], [poolno], [setvars], [values], [pevar], [replicate_pe] &
& , [sync] )
```

This function executes SQL (data query) $dtname$ which was previously registered by ODB_addview function. It returns number of rows and columns to be allocated before ODB_get can be issued. Return code $> 0$ denotes local number of rows for parallel queries. For non-parallel queries the number equals to $nrows$.

The parameters denote:

- **rc** INTEGER(4)
  - Return code which upon successful completion return value $> 0$. For non-parallel queries value indicates how many rows satisfied the WHERE-condition and the return code should be equal to $nrows$. For parallel queries the value indicates how many rows from local pools satisfied the WHERE-condition; thus the sum of return codes on every MPI-processor would give value found in $nrows$.

- **handle** INTEGER(4), INTENT(IN)
  - Database handle as returned by ODB_open
4.2 Description of the ODB Fortran90 functions

- **dtname**: CHARACTER(LEN=*) , INTENT(IN)
  - SQL-query name (lower case) or TABLE name (begins with @-sign).
  - In the latter case reverts the call to function ODB_getsize

- **nrows**: INTEGER(4) , INTENT(OUT)
  - Number of rows that satisfy this query

- **ncols**: INTEGER(4) , INTENT(OUT)
  - Number of columns that satisfy this query

- **nra**: INTEGER(4) , INTENT(OUT) , OPTIONAL
  - Optimal number of rows that ought to be allocated for subsequent data matrix in order to reduce memory bank conflicts or cache-line misses.
  - The value is guaranteed to be ≥ nrows, but the exact value is computer platform depend.

- **poolno**: INTEGER(4) , INTENT(IN) , OPTIONAL
  - The pool number. If omitted, the default = -1 will be assumed i.e. all applicable pools owned by the particular MPI-task will be scanned

- **pevar**: CHARACTER(LEN=*) , INTENT(IN) , OPTIONAL
  - Applicable for MPI-parallel data shuffle. More later...

- **replicate_pe**: INTEGER(4) , INTENT(IN) , OPTIONAL
  - Applicable for replicated MPI-parallel queries only. If set to -1, then data obtained from local processors/pools will be replicated on every processor upon ODB_get. If set to a value in the range of 1..nproc, then only the processor in the range will receive the replicated data.

- **setvars(:)**: CHARACTER(LEN=*) , INTENT(IN) , OPTIONAL
  - For parameterized queries and used in connection with the values-parameter. Attempts to override the value of $-variable present in the data query dtname.

- **values(:)**: REAL(8) , INTENT(IN) , OPTIONAL
  - Supplies new values for a set of dollar-variables present in the data query dtname. Variable-list is supplied via setvars. The lifetime of the new value is duration of the SQL-query only. You need to use ODB_setval to change the value of a dollar-variable permanently.

- **sync**: LOGICAL , INTENT(IN) , OPTIONAL
  - If TRUE, then synchronizes queries from every MPI-processor

**Examples:**
! Execute data query on pool number 17 only
! and get dimensions for data matrix allocation
INTEGER(4) :: rc, handle
INTEGER(4) :: nrows, ncols, nra
rc = ODB_select(handle,'query',nrows,ncols,nra=nra,poolno=17)

! Invoke parameterized query, where '$tslot' is
! referred in WHERE-statement, by looping over 4D-Var timeslots
CHARACTER(LEN=32) :: clsetvar(1)
REAL(8) :: vals(size(clsetvar))
INTEGER(4) :: tslot
clsetvar(1) = '$tslot'
do tslot=1,25
  vals(1) = tslot
  rc = ODB_select(handle,'query',nrows,ncols, &
                  & setvars=clsetvar(1:1), values=vals(1:1))
  ... allocate data matrix
  ... fill data matrix with data and work with it
  ... dellocate data matrix
  rc = ODB_cancel(handle,'query')
enddo ! do tslot=1,25

! Invoke parallel data query to collect global view
! on every MPI-task (all tasks and pools involved)
rc = ODB_select(handle,'query',nrows,ncols,nra=nra,&
                & replicate_PE=-1)

! Invoke parallel shuffle against loop-variable '$loop'
rc = ODB_select(handle,'query',nrows,ncols,nra=nra,&
                & pevar='$loop')

4.2.17 ODB_setval

Usage of the function ODB_setval is as follows:

oldvalue = ODB_setval ( handle, varname, newvalue &
                        & , [viewname] )

Re-defines the value $-variable varname by newvalue and apply it optionally to SQL-query viewname only.
The **DATATYPE** can be either **REAL(8)** or **INTEGER(4)**, or types which will be automatically casted into these two basic types.

See also function **ODB_getval** at 4.2.9

The parameters denote:

- **oldvalue**  
  **DATATYPE**  
  Returns the previous value, or 0 if variable was not defined

- **handle**  
  **INTEGER(4)**, **INTENT(IN)**  
  Database handle as returned by **ODB_open**

- **varname**  
  **CHARACTER(LEN=*)**, **INTENT(IN)**  
  The $-variable as specified via **SET**-statement either in data definition layout (ddl-file) or SQL-query. In the latter case, the **viewname** must be present

- **newvalue**  
  **DATATYPE**, **INTENT(IN)**  
  The new value to be assigned

- **viewname**  
  **CHARACTER(LEN=*)**, **INTENT(IN)**, **OPTIONAL**  
  The optional SQL-query name

**Examples:**

```fortran
! Change desired timeslot number to 2 (globally)
INTEGER(4) :: handle, idummy
INTEGER(4) :: ioldvalue
ioldvalue = ODB_setval(handle,'$tslot',2)

! And now restore the old value
! The "idummy" becomes 2
idummy = ODB_setval(handle,'$tslot',ioldvalue)

! Set floating point value to 3.14 in view 'jeffpot'
REAL(8), parameter :: pi_approx = 3.14d0
REAL(8) :: oldvalue
oldvalue = ODB_setval(handle,'$pi',pi_approx, &
       & viewname='jeffpot')
```

### 4.2.18 **ODB_swapout**

Usage of the function **ODB_swapout** is as follows:
This function removes ODB data from memory by optionally saving it to disk, if save-option was set to .TRUE..

The dtname denotes the table name, SQL-query name or '*', where to remove the associated data from memory. When SQL-query is specified, then all tables accessed via this query will be removed from memory. In particular with '*', all tables brought into memory will be removed.

This used to be an effective way of interleaving data in memory between subsequent data pools so that data in two or more pools never needed to coexist. This is useful especially when performing post-processing on systems with limited amount of memory. While it still works fine for I/O-method#1, this function is now being superceded by ODB_release, which also works for the new I/O-method#4 (see more on ODB_IO_METHOD from A) The parameters denote:

\[
\begin{align*}
rc & \quad \text{INTEGER(4)} \\
& \quad \text{Return code, which upon successful completion will be } \geq 0 \\
handle & \quad \text{INTEGER(4), INTENT(IN)} \\
& \quad \text{Database handle as returned by ODB_open} \\
dtname & \quad \text{CHARACTER(LEN=*)}, \text{INTENT(IN)} \\
& \quad \text{Query name (in lower case letter) as specified in the CREATE VIEW-statement. Usually points also to the SQL-query file dtname.sql; sometimes even to database specific file $dbname_dtname.sql} \\
poolno & \quad \text{INTEGER(4), INTENT(IN), OPTIONAL} \\
& \quad \text{The pool number. If omitted, the default } = -1 \text{ will be assumed i.e. all applicable pools owned by the particular MPI-task will be scanned} \\
save & \quad \text{LOGICAL, INTENT(IN), OPTIONAL} \\
& \quad \text{If this parameter is present and was set to .TRUE., and if database was not opened in 'READONLY'-mode, then the routine will update tables associated with the dtname}
\end{align*}
\]

Examples:

\[
\begin{align*}
\text{! Remove table '@errstat' (all pools) from memory without saves} \\
\text{INTEGER(4) :: rc, handle} \\
\text{rc = ODB_swapout(handle, '@errstat')}
\end{align*}
\]
4.2 Description of the ODB Fortran90 functions

4.2.19 ODB_varexist

Usage of the function ODB_varexist is as follows:

```fortran
exist = ODB_varexist ( handle, varname &
                      & [viewname] )
```

You can test the presence of a particular $-variable `varname`, which can optionally belong to SQL-query `viewname`.

See also for related functions ODB_getval (4.2.9) and ODB_setval (4.2.17).

The parameters denote:

- `exist` : LOGICAL
  If the $-variable `varname` is present, returns .TRUE., otherwise .FALSE.
- `handle` : INTEGER(4), INTENT(IN)
  Database handle as returned by ODB_open
- `varname` : CHARACTER(LEN=*), INTENT(IN)
  The $-variable as specified via SET-statement either in data definition layout (ddl-file) or SQL-query. In the latter case, the `viewname` must be present
- `viewname` : CHARACTER(LEN=*), INTENT(IN), OPTIONAL
  The optional SQL-query name

Examples:

```fortran
! Check whether the parameter '$mdi' has been defined at all
INTEGER(4) :: handle
REAL(8) :: value
if (ODB_varexist(handle, '$mdi')) then
  value = ODB_getval(handle, '$mdi')
```
print *, 'Parameter $mdi is present. The current value = ', value
else
  print *, 'Parameter $mdi is NOT present'
endif

4.2.20 ODB_varindex

Usage of the function ODB_varindex is as follows:

```fortran
rc = ODB_varindex ( handle, dtname, names, idx )
```

This function helps you to avoid using hardcoded column index numbers when accessing data in matrices returned by ODB_get. It helps you to find out the column indices (idx) of the given (fully qualified) column names (names). If some column name is not part of data query (or table) dtname, the corresponding element in idx has value zero (0).

The parameters denote:

- **handle**
  INTEGER(4), INTENT(IN)
  Database handle as returned by ODB_open

- **dtname**
  CHARACTER(LEN=*) , INTENT(IN)
  Query name (in lower case letter) as specified in the CREATE VIEW-statement. Usually points also to the SQL-query file dtname.sql; sometimes even to database specific file $dbname_dtname.sql

- **names(:)**
  CHARACTER(LEN=*) , INTENT(IN)
  Fully qualified array of column names, like 'obstype@hdr'. Please note that @LINK-information must be given as follows:
  - For table.offset enter LINKOFFSET(table), and
  - for table.len enter LINKLEN(table).
  In addition both are case sensitive, i.e. LINKOFFSET and LINKLEN must be entered in capital letters

- **idx(:)**
  INTEGER(4), INTENT(OUT)
  Column index numbers to be used to address the second dimension (i.e. column) of data matrices returned from ODB_get. Each individual value must be $\geq 1$, or otherwise the particular column name specified in its corresponding idx-entry has no match in dtname. The dimension of idx must be at least the same of length of the names-vector

**Examples:**
Given the following data query:
CREATE VIEW query AS
    SELECT a, b, c
    FROM table;

Find out column indices for \{c, b, a\} in this order
Should print out: 3, 2, 1

```fortran
INTEGER(4) :: rc, handle
CHARACTER(LEN=128) :: names(3)
INTEGER(4) :: idx(size(names))
names(1) = 'c@table'
names(2) = 'b@table'
names(3) = 'a@table'
rc = ODB_varindex(handle,'query',names,idx)
print *, 'idx(:) = ', idx(:)
```
5 Some examples

In the following few sections we explain how to

- Initialize ODB environment on Unix-level
- Create new, empty database
- Populate database with data
- Make SQL-queries to retrieve data records
- Add more records if certain records didn’t already exist there
- Update existing records
- Archive and remove database from disk

The aim of this exercise is to give a firm understanding how ODB works in practice and what steps need to be taken to store, query and update the database.

The example is an oversimplified simulator to perform synop pressure bias correction at station level, where initially no station or bias information was present. After running 4D-Var for a number of days, such a station level bias correction knowledge gradually builds up.

Data is based upon this knowledge and is put into ODB database in order to be able to retrieve and control its data properly. No more ad-hoc data files are necessary and online/offline queries can be made to monitor validity of such data at any given moment.

5.1 Initialization of ODB environment

In order to run examples in this document, you need to initialize the ODB environment properly. The initialization scripts shown are applicable for ECMWF Linux desktop environment.

Under the Korn-shell the user needs to give the following commands:

```
$ export ODB_VERSION=latest
$ . /vol/odb/ODBDIR/use_odb.sh
```

If you are C-shell (or tcsh) user, you need to give the following commands:

```
% setenv ODB_VERSION latest
% source /vol/odb/ODBDIR/use_odb
```
Since the latest-version is the default, and since ECMWF has use-command available\(^8\), the easiest way of getting going (in either of the shell-environments) is in fact to issue simply a command:

```
use odb
```

5.2 Creating a new database

In this subsection you will be introduced to the following ODB Fortran90 interface functions:

- **ODB_init**: Initialize ODB (and parallel, if applicable) environment
- **ODB_open**: Open and create new database
- **ODB_close**: Save (meta)data upon closing otherwise empty database
- **ODB_end**: Finish with ODB (and terminate parallel processing)

Before creating a new database, you need to specify its data layout. This needs to be placed in file `$dbname.ddl`, where `$dbname` is your choice for database name. The name has to start with a capital letter ([A-Z]), optionally followed by capital letters or numbers (i.e. [A-Z0-9]) No underscores are accepted at present. Also the length of the database name must be no more than 64 characters.

You need to choose whether you want to use one of the recognized database names (i.e. permanent database names) or invent your own database name. This is an important factor, since ODB (mainly) supports static linkage (see [ODB_STATIC_LINKING](see p.194)), where certain most commonly used database names are automatically bound to the ODB compilation system.

There is however a relatively new workaround method explained in 9.2, which allows you to define any database name (and even override the existing permanent ones) on-the-fly and still perform a static linkage.

In this section we assume a completely new database name and call it `DB`. Its layout is shown here:

```fortran
// Database DB: file DB.ddl

SET $mdi = 2147483647;

CREATE TABLE hdr (
```

\(^8\)The `use-command` is just a small script to make life easier
5.2 Creating a new database

It consists of two tables `hdr` and `body`. Table `hdr` can be considered as parent-table since it has a `@LINK` to its child-table `body`. Both tables have a few columns containing mixed 4-byte integer, 64-bit floating point and fixed 8-byte character data. The column names `seqno` and `entryno` can be considered as a kind of key entries, which we try to keep unique. This uniqueness fact is not mentioned anywhere in the data layout, but when we start to populate them with data (see 5.3), it will become clearer.

If you already are familiar with genuine relational databases, the database schema (i.e. data layout) could look like as follows ([Fch02]):

```sql
CREATE TABLE body (  
  entryno  pk1int,  
  varno    pk1int,  
  press    pk9real,  
  obsvalue pk9real,  
  depar    pk9real,  
  obs_error pk9real,  
  bias     pk9real,
);  
```
The compilation of the ODB database layout `DB.ddl` can be accomplished as follows:

```bash
$ newodb DB
```

If database `DB` was already in the list of recognized database names, the compilation would have been:

```bash
$ odbcomp DB.ddl
```

Outcome of this compilation process is explained more in 9.2 and 9.3. The bottom line is that you have got now a library `libDB.a`, which becomes a backbone for your database access of this particular (nothing else) database `DB`.

The next stage is to create a program `createdb.F90` which will handle creation of any (i.e. generic) ODB database. Since we have chosen to oversimplify the matters, this program does nothing else but creates a new and empty database. In a normal situation we would be doing both: a database creation and its population with data. The program is as follows:

```fortran
PROGRAM createdb
!*** createdb.F90 ***
USE odb_module
implicit none
INTEGER(4) :: myproc,nproc,npools,h
INTEGER(4) :: nrows, ncols, nra, rc
CHARACTER(len=64) dbname
rc = ODB_init(myproc=myproc,nproc,npools,h
CALL getarg(1,dbname) ! Database name
npools = 1 ! Change this to a desired number
h = ODB_open(dbname,'NEW',npools)
rc = ODB_close(h,save=.TRUE.)
rc = ODB_end()
END PROGRAM createdb
```
5.2 Creating a new database

To run this program, you need to execute the following commands (but make sure you have initialized your ODB environment first, see 5.1):

```
# Go to directory where DB.ddl sits
# This will become (by default) your database directory
cd /my/DB

# (1) Compile database DB (once in the lifetime):
newodb DB

# (2) Run setup to get the IOASSIGN right:
./DB.setup > DB.IOASSIGN
export IOASSIGN=`pwd`/DB.IOASSIGN

# (3) Make sure database directories get right:
export ODB_DATAPATH_DB=`pwd`
export ODB_SRCPATH_DB=`pwd`

# (4) Compile and link Fortran90 application:
odbf90 createdb.F90 -lDB -o createdb.x

# (5) Run the application
./createdb.x DB
```

If you change the data layout (DB.ddl), you need to redo all steps from 1...5. However, if you just change the Fortran90 application, you just redo the steps 4 and 5. If you decide to relocate the database destination directory, you have to make sure your IOASSIGN-file reflects this change, and environment variables ODB_DATAPATH (see p. 190) and ODB_SRCPATH (see p. 194) get adjusted, too.

The output listing for this example is shown next. You can see that database has been created and it has a certain number of pools. Line numbers are not part of the real output. Nothing much exciting:

```
1: MPL_INIT : MPL_METHOD=JP_BLOCKING_BUFFERED
2: MPL_INIT : MAILBOX SIZE= 1000000
3: MPL_BUFFER_METHOD: 2 540680269
4: >>> Assuming static linking mode
5: ***Warning: Poolmasking ignored altogether
6: OPEN_DB: handle= 1
7: DataBase-name="DB"
8: inuse ? T
9: newdb ? T
10: readonly ? F
11: no. of global pools = 1
```
5.3 Populating with data

We intend to populate the database tables with some initial data. You will be introduced to the following ODB Fortran90 interface functions in this subsection:

- ODB_open: Open an existing database
- ODB_getval: Find out the value of a parameterized variable
- ODB_getsize: Inquire existing table dimensions
- ODB_put: Append full data rows directly to a table
- ODB_close: Save data and close database

If we were using standard relational database engines, we would simply type in the following stream of SQL-commands:

```sql
INSERT INTO hdr
(seqno, date, time, statid, lat, lon)
VALUES
( 1, 20010101, 100000, " 07027", 49.18, -0.45),
( 2, 20010101, 130000, " 07061", 49.82, 3.20),
```
5.3 Populating with data

Unfortunately these are not available in ODB/SQL-language as yet, so we need to put some more effort. We have two choices to supply data to ODB database:

1. Let a Fortran90 program to feed data to ODB

2. Pass text datafile(s) to ODB via simulobs2odb-utility

We will cover here only the more labourous way i.e. using ODB Fortran90 interface. An alternative, a completely novel approach in connection with simulated observations will be covered in section 10.4. In this approach the nearly free formatted text datafiles containing desired values are passed into a script (simulobs2odb), which further down pre-processes the data so that the ODB-application odbsimulobs can take over and populate the database directly from these text files.

An ODB Fortran90 application populate.F90, which is able to feed our database, could look something like:

```fortran
PROGRAM populate
!*** populate.F90 ***
USE odb_module
implicit none
REAL(8), ALLOCATABLE :: x(:, :)
REAL(8) mdi ! Missing data indicator ("NULL")
INTEGER(4) :: myproc, nproc, npools, jp, h
INTEGER(4) :: nrows, ncols, rc
character(len=64) dbname, tblname
rc = ODB_init(myproc=myproc,nproc=nproc)
CALL getarg(1,dbname) ! Database name
npools = 0 ! Gets true value from ODB_open()
h = ODB_open(dbname,'OLD',npools)
mdi = abs(ODB_getval(h, '$mdi'))
jp = 1 ! Here: Populate poolno#1 only for now
if (myproc == mod(jp-1,nproc)+1) then
tblname = '@hdr' ! Note '@'-sign in front of
```
rc = ODB_getsize(h, tblname, nrows, ncols, poolno=jp)
nrows = 3 ! Three new rows for TABLE hdr
ALLOCATE(x(nrows,0:ncols))
! segno date time statid ...
! ... lat lon body.offset body.len
x(1,1:ncols) = (/1d0, 20010101d0, 100000d0, 8H 07027, &
& 49.18d0, -0.45d0, 0d0, 2d0/)
x(2,1:ncols) = (/2d0, 20010101d0, 130000d0, 8H 07061, &
& 49.82d0, 3.20d0, 2d0, 2d0/)
x(3,1:ncols) = (/3d0, 20010101d0, 130000d0, 8H 07075, &
& 49.78d0, 4.63d0, 4d0, 1d0/)
! Note: the following APPENDs after the existing data
rc = ODB_put(h, tblname, x, nrows, ncols, poolno=jp)
DEALLOCATE(x)
tblname = '@body' ! Note '@'-sign in front of
crc = ODB_getsize(h, tblname, nrows, ncols, poolno=jp)
nrows = 5 ! Five new rows for TABLE body
ALLOCATE(x(nrows,0:ncols))
! entryno, varno, press, obsvalue, depar, obs_error, bias
x(1,1:ncols) = &
(&/1d0, 110d0, 657.05d0, 98530.00d0, -85.86d0, 69.03d0, -mdi/)
x(2,1:ncols) = &
(&/2d0, 39d0, 98530.00d0, 282.70d0, 1.80d0, 1.95d0, -mdi/)
x(3,1:ncols) = &
(&/1d0, 110d0, 990.47d0, 98480.00d0, -221.59d0, 69.09d0, -mdi/)
x(4,1:ncols) = &
(&/2d0, 39d0, 98480.00d0, 279.80d0, 1.06d0, 1.95d0, -mdi/)
x(5,1:ncols) = &
(&/1d0, 110d0, 1451.38d0, 98250.00d0, -123.32d0, 69.36d0, -mdi/)
! Note: the following APPENDs after the existing data
rc = ODB_put(h, tblname, x, nrows, ncols, poolno=jp)
DEALLOCATE(x)
endif
rc = ODB_close(h, save=.TRUE.)
rc = ODB_end()
END PROGRAM populate

To run this program, you need to execute the following commands (we assume that you have already successfully performed the steps 1..3 as shown in 5.2):

```bash
# (4) Compile and link Fortran90 application:
odbf90 populate.F90 -lDB -o populate.x

# (5) Run the application
./populate.x DB
5.4 Simple data querying

The output listing for this example is shown next. It is pretty much similar compared to the database creation output earlier (5.2), but shows that database has been opened as an existing one (line#9). Line numbers are not part of the real output:

```
1: MPL_INIT : MPL_METHOD=JP_BLOCKING_BUFFERED
2: MPL_INIT : MAILBOX_SIZE= 1000000
3: MPL_BUFFER_METHOD: 2 540680269
4: >>> Assuming static linking mode
5: ***Warning: Poolmasking ignored altogether
6: OPEN_DB: handle= 1
7: DataBase-name="DB"
8: inuse ? T
9: newdb ? F
10: readonly ? F
11: no. of global pools = 1
12: global max poolno = 1
13: no. of local pools = 1
14: sequence no. scaling factor = 1073741824
15: max. no. of rows per pool = 1073741823
16: I/O-method = 1
17: iounit = 0
18: ntables = 2
19: *** newio.c:init_IOs() ***
20: ODB_WRITE_EMPTY_FILES=0
21: ODB_CONSIDER_TABLES=*  
22: ODB_IO_KEEP_INCORE=1
23: ODB_IO_FILESIZE=32 MB
24: ODB_IO_BUFSIZE=1048576 bytes
25: ODB_IO_GRPSIZE=1 (or max no. of pools)
26: ODB_IO_PROFILE=0
27: ODB_IO_VERBOSE=0
28: ODB_IO_METHOD=1
29: ODB_IO_LOCK=0
30: ***********************
31: END of OPEN_DB
```

This example was for illustration purposes only and in practice you would like to read data from an input file and pass it to ODB. However, this has now been made automatic by introduction of simulobs2odb-utility.

5.4 Simple data querying

This subsection introduces the following ODB Fortran90 interface functions:

- `ODB_open`: Open an existing database for 'READONLY'-access
• ODB_addview: Register the query
• ODB_getnames: Find out column names in the query
• ODB_select: Execute and find out allocation dimensions of the request
• ODB_get: Fetch data satisfying the request into the user space
• ODB_cancel: Deactivate the query and free internal indices
• ODB_release: Remove a data pool from memory (without saves)
• ODB_swapout: Remove particular table(s) and pool from memory
• ODB_close: Close database without saving it

In this example we continue working with the database created and populated in the previous sections. A logical continuation is to check that the data fed into database is what we would expect. The next thing is to show some more complicated queries and that all these queries can be accomplished by using the generic data query processing program `querydata.F90` as follows:

```fortran
PROGRAM querydata
!*** querydata.F90 ***
USE odb_module
implicit none
REAL(8), ALLOCATABLE :: x(:,:)
INTEGER(4) :: myproc,nproc,npools,jp,h
INTEGER(4) :: nrows, ncols, nra, rc
INTEGER(4) :: numargs, iargc, jque, jr, jc, nc
character(len=64) dbname, queryname
character(len=64), allocatable :: colnames(:)
rc = ODB_init(myproc=myproc,nproc=nproc)
numargs = iargc()
if (numargs < 1) then
   CALL ODB_abort('querydata',&
   'Usage: querydata DBname [query1 [query2] ...]',numargs)
endif
CALL getarg(1, dbname) ! Database name
do jque=2,numargs
   CALL getarg(jque, queryname) ! Query name
   npools = 0 ! Gets true value from ODB_open()
   h = ODB_open(dbname,'READONLY',npools)
   write(*,*)'Exec query#',jque-1,'="'//trim(queryname)//"'
   rc = ODB_addview(h, queryname) ! Not necessary anymore
   nc = ODB_getnames(h, queryname, 'name')
   ALLOCATE(colnames(nc))
   do jp=myproc,npools,nproc ! Round-robin, "my" pools only
      rc = ODB_select(h, queryname, nrows, ncols, nra=nc,poolno=jp)
   enddo
```

5.4 Simple data querying

Compilation and linking of this generic query processor program can be performed similarly to the data populator-program 5.3. The *queryname* denotes the SQL-request in the corresponding CREATE VIEW statement(s) shown later:

```fortran
if (nrows > 0) then
   ALLOCATE(x(nra,0:ncols))
   rc = ODB_get(h,queryname,x,nrows,ncols,poolno=jp)
   write(*,'((3a20))') (trim(colnames(jc)),jc=1,nc)
   do jr=1,nrows
      write(*,'(1p,(5x,3(1x,g20.10)))') x(jr,1:ncols)
   enddo
   DEALLOCATE(x)
endif
rc = ODB_cancel(h,queryname,poolno=jp)
! The following is the same as the less preferred
! rc = ODB_swapout(h, '*', poolno=jp)
rc = ODB_release(h,poolno=jp)
edo ! do jp=myproc,npools,nproc
   DEALLOCATE(colnames)
   rc = ODB_close(h)
edo ! do jque=2,numargs
rc = ODB_end()
END PROGRAM querydata
```

When you apply new query, you need to repeat the steps 2…4. To check that the data fed into the database is what we expect it to be, we start with simple queries like `querydata1.sql`:

```sql
CREATE VIEW querydata1 AS SELECT * FROM hdr;
```

or `querydata2.sql`:

```sql
```
CREATE VIEW querydata2 AS SELECT * FROM body;

They ask for all columns (and thus ‘*’) from single tables. In terms of genuine database engine, these queries would have been exactly identical, if the engine supported the CREATE VIEW-statements. Otherwise they should have been just queries forwarded directly to their client-server, like:

```
SELECT * FROM hdr;
SELECT * FROM body;
```

From the output of these queries we can judge that the database has been loaded correctly. You can also see what commands were used to produce desired result:

```
$ odbf90 -c querydata.F90
$ odbcomp -lDB querydata1.sql
$ odbf90 querydata.o -lDB -o querydata.x
$ ./querydata.x DB querydata1
...
END of OPEN_DB
Exec query# 1 ="querydata1"
seqno@hdr date@hdr time@hdr
statid@hdr lat@hdr lon@hdr
LINKOFFSET(body)@hdr LINKLEN(body)@hdr
1.000000000 20010101.00 100000.0000
6.0135615738-154 49.18000000 -0.4500000000
0.000000000 2.000000000
2.000000000 20010101.00 130000.0000
6.0135615738-154 49.82000000 3.200000000
2.000000000 2.000000000
3.000000000 20010101.00 130000.0000
6.0135615738-154 49.78000000 4.630000000
4.000000000 1.000000000
$ odbcomp -lDB querydata2.sql
$ odbf90 querydata.o -lDB -o querydata.x
$ ./querydata.x DB querydata2
...
END of OPEN_DB
Exec query# 2 ="querydata2"
entryno@body varno@body press@body
obsvalue@body depar@body obs_error@body
bias@body
1.000000000 110.0000000 657.0500000
98530.00000 -85.86000000 69.03000000
```
5.4 Simple data querying

Note that you can also execute both queries in a single program invocation. Refer to the next output to see how that can be done. This is useful if you have a number of queries that you want to execute in a batch mode:

```
$ odbf90 -c querydata.F90
$ odbcomp -lDB querydata1.sql querydata2.sql
$ odbf90 querydata.o -lDB -c querydata.x
$ ./querydata.x DB querydata1 querydata2

Exec query# 1 ="querydata1"
seqno@hdr  date@hdr  time@hdr
statid@hdr  lat@hdr  lon@hdr
LINKOFFSET(body)@hdr  LINKLEN(body)@hdr
1.000000000  20010101.00  100000.0000
6.0135615738-154  49.18000000  -0.4500000000
0.000000000  2.000000000
2.000000000  20010101.00  130000.0000
6.0135615738-154  49.82000000  3.2000000000
2.000000000  2.000000000
3.000000000  20010101.00  130000.0000
6.0135615738-154  49.78000000  4.6300000000
4.000000000  1.000000000

Exec query# 2 ="querydata2"
entryno@body  varno@body  press@body
obsvalue@body  depar@body  obs_error@body
bias@body
1.000000000  110.00000000  657.05000000
98530.00000  -85.86000000  69.03000000
0.000000000
2.000000000  39.00000000  98530.00000
282.7000000  1.80000000  1.9500000000
```
Note that the station id (\texttt{statid}-column) has been printed improperly. This is because it is stored as 64-bit floating point number, but is in fact an 8-byte character string. By having more targeted format-statement we could have fixed this problem. We could have used function \texttt{ODB\_getnames} to resolve the type of any column and made format-statements more type-sensitive. However, it is maybe easier to use \texttt{ODB\_print} for now to resolve formatting problems.

### 5.5 More complicated requests

We continue our description of data queries, but move on into somewhat more complex queries. We use the same program \texttt{querydata.F90} from the previous subsection. You should always remember to ask only such column entries that you really need, since the ODB/SQL-query language is presently not very remarkable and is not for example yet capable of processing subqueries or aggregate functions. These features present in standard SQL often reduce the number of output rows dramatically. For example, asking for all columns as in the previous example is normally an overkill and waste of computer resources. And when you use more tables (the \texttt{FROM}-clause), the table joins will can produce too much information to be analyzed properly.

Here are a few examples of queries, which produce more targeted output:

```sql
CREATE VIEW cx1 AS
  SELECT lat, lon
  FROM hdr;

CREATE VIEW cx2 AS
  SELECT date, time
  FROM hdr
  WHERE seqno = 2;
```

The first view (\texttt{cx1}) pulls out all latitudes and longitudes from table \texttt{hdr}. The second view (\texttt{cx2}) gives date and time information, but under the condition that the sequence number (\texttt{seqno}) is equal to two.
5.5 More complicated requests

CREATE VIEW cx3 AS
SELECT press, obsvalue
FROM hdr, body // Natural join
WHERE seqno = 2;

The third example (cx3) shows our first table join. In the ODB/SQL, we normally do not have to worry about getting a huge number of rows as output due to potential cartesian product: the number of rows in the parent table hdr times the number of rows in the child table body. This is because of the special @LINK-column structure, which automatically recognizes the parent-child relation (hierarchy). In a genuine relational databases this would be called natural join and would require (at least) that seqno would be present in both tables as a controlling key. Note also that we ask columns (in SELECT) solely belonging to table body, whereas the WHERE-condition refers to other table hdr. This shows once again that we do not have to specify the columns found in WHERE-conditions in the SELECT-clause, too.

CREATE VIEW cx4 AS
SELECT seqno, "@body"
FROM hdr, body // Natural join
WHERE statid = '07027'
AND varno = 110;

The fourth query (cx4) asks for all columns found in body-table under a character a string condition. The column statid is a right justified character string. In ODB/SQL the WHERE-condition automatically right-adjusts any constant character expression by padding as many blanks in front of that the string would finally occupy the maximum 8 characters. This is quite different to what the genuine relational database engines do: there character strings are by default assumed to be left-justified. Furthermore, this query also shows how to use the AND-condition to effectively eliminate the number of rows in output.

CREATE VIEW cx5 AS
SELECT seqno, date, time, varno, obsvalue@body
FROM hdr, body // Natural join
WHERE (statid RLIKE '070.*' AND varno = 39)
OR seqno = 3;
The final case (cx5) shows how to use the OR-condition and regular expression (LIKE) in finding out a character string match for a station identifier. The regular expression is a Unix-like regular expression. Without using the OR-condition we would have got only such entries, which contained temperatures (varno equal to 39), since the last hdr-row (seqno equal to 3) body-table entries didn’t have any temperatures. But use of the OR-condition gave us also the data associated with the last hdr-row.

When we run these queries, the program produces the following output:

```bash
$ odbf90 -c querydata.F90
$ odbcsql -lDB complex.sql
$ odbf90 querydata.o -lDB -o querydata.x
$ ./querydata.x DB cx1 cx2 cx3 cx4 cx5
...
Exec query# 1 ="cx1"
    lat@hdr   lon@hdr
    49.180000000  -0.4500000000
    49.820000000  3.2000000000
    49.780000000  4.6300000000
...
Exec query# 2 ="cx2"
    date@hdr   time@hdr
    20010101.00  130000.0000
...
Exec query# 3 ="cx3"
    press@body   obsvalue@body
    990.47000000 98480.000000
    98480.000000 279.8000000
...
Exec query# 4 ="cx4"
    seqno@hdr  entryno@body  varno@body
    press@body  obsvalue@body  depar@body  bias@body
    1.000000000 1.000000000 110.0000000
    657.05000000 98530.00000  -85.8600000
    69.03000000 0.00000000
...
Exec query# 5 ="cx5"
    seqno@hdr  date@hdr  time@hdr
    varno@body  obsvalue@body
    1.000000000 20010101.00 100000.0000
    39.00000000 282.7000000
    2.000000000 20010101.00 130000.0000
    39.00000000 279.8000000
    3.000000000 20010101.00 130000.0000
    110.00000000 98250.00000
```
Using traditional database engines, the queries involving joins and more complex WHERE-condition (cx3 .. cx5) should have been written as follows:

```
-- cx3
SELECT press, obsvalue
FROM hdr NATURAL JOIN body
WHERE hdr.seqno = 2;
-- cx4
SELECT seqno, body.*
FROM hdr NATURAL JOIN body
WHERE statid = ' 07027'
    AND varno = 110;
-- cx5
SELECT seqno, date, time,
    varno, body.obsvalue
FROM hdr NATURAL JOIN body
WHERE (statid LIKE ' 070%' AND varno = 39)
    OR hdr.seqno = 3;
```

### 5.6 Using parameterized queries

In this subsection we perform a series of parameterized queries. They are ordinary ODB/SQL-queries except that the constant values in the WHERE-condition are parameterized using $-variables, and whose current value can be altered from within ODB Fortran90 layer. This is very useful, since we can effectively execute different requests using the same ODB/SQL-query definition. You will learn about how to

- **ODB_getval**: Get the existing value of a parameterized variable
- **ODB_setval**: Alter the value of a parameterized variable
- **ODB_select**: Temporarily change the values of parameterized variables

For parameterized queries we need to modify our previous program a little in order to be able to change the parameter values on-the-fly. For simplicity we also removed the loop over input arguments. The first program `paramquery1.F90` looks as follows:

```
PROGRAM paramquery1
  !*** paramquery1.F90 ***
  USE odb_module
```
We could also refrain from calling function `ODB_setval` and let the function `ODB_select` to alter the parameter value for the duration when the SQL is executed. Such a program
(paramquery2.F90) is as follows:

```fortran
PROGRAM paramquery2
! *** paramquery2.F90 ***
USE odb_module
implicit none
REAL(8), ALLOCATABLE :: x(:, :)
REAL(8) newvalue, oldvalue
INTEGER(4) :: myproc, nproc, npools, jp, h
INTEGER(4) :: nrows, ncols, nra, rc
INTEGER(4) :: jr, jc, nc
character(len=64) dbname, queryname, varname, cval
character(len=64), allocatable :: colnames(:)
rc = ODB_init(myproc=myproc, nproc=nproc)
CALL getarg(1, dbname) ! Database name
CALL getarg(2, queryname) ! Query name
CALL getarg(3, varname) ! $-variable name without '$'
CALL getarg(4, cval) ! REAL value of $-variable
read(cval, '*') newvalue
npools = 0 ! Gets true value from ODB_open()
h = ODB_open(dbname, 'READONLY', npools)
rc = ODB_addview(h, queryname) ! Not necessary anymore
oldvalue = ODB_getval(h, '$'//varname, viewname=queryname)
write('(*,*) myproc, ': oldvalue at begin >', oldvalue
nc = ODB_getnames(h, queryname, 'name')
ALLOCATE(colnames(nc))
nc = ODB_getnames(h, queryname, 'name', colnames)
write('(*,*) myproc, ': Exec query='//trim(queryname)//'*/'
do jp=myproc, npools, nproc ! Round-robin, "my" pools only
  rc = ODB_select(h, queryname, nrows, ncols, nra=nra, poolno=jp, &
     & setvars=(''$/varname'', values=(*newvalue/*))
  if (nrows > 0) then
   ALLOCATE(x(nra, 0:ncols))
    rc = ODB_get(h, queryname, x, nrows, ncols, poolno=jp)
    write('(*,'(3a20))' (trim(colnames(jc)), jc=1, nc)
do jr=1, nrows
      write('(*,'(1p,(5x,3(1x,g20.10)))') x(jr, 1:ncols)
endo
DEALLOCATE(x)
endif
rc = ODB_cancel(h, queryname, poolno=jp)
! The following is the same as the less preferred
! rc = ODB_swapout(h, '***', poolno=jp)
rc = ODB_release(h, poolno=jp)
endo
DEALLOCATE(colnames)
oldvalue = ODB_getval(h, '$'//varname, viewname=queryname)
write('(*,*) myproc, ': oldvalue at end >', oldvalue
```

5.6 Using parameterized queries
We use the following, a slightly artificial parameterized query and during the execution we vary the value of sequence number ($seqno) between 1 and 3:

```sql
// A parameterized query
SET $seqno = 0;
CREATE VIEW paramquery AS
    SELECT seqno@hdr, entryno, depar
    FROM hdr, body
    WHERE seqno = $seqno
;
```

After running the parameterized query with different parameter values, the first program paramquery1.F90 produces the following output:

```
$ odbf90 -c paramquery1.F90
$ odbcomp -lDB paramquery.sql
$ odbf90 paramquery1.o -lDB -o paramquery1.x
$ ./paramquery1.x DB paramquery seqno 1
... END of OPEN_DB
1 : oldvalue,newvalue > 0.000000000000000000E+00 1.00000000000000000
1 : Exec query="paramquery"
    seqno@hdr entryno@body depar@body
    1.000000000 1.000000000 -85.86000000
    1.000000000 2.000000000 1.800000000
1 : oldvalue at end > 1.00000000000000000

$ ./paramquery1.x DB paramquery seqno 2
... END of OPEN_DB
1 : oldvalue,newvalue > 0.000000000000000000E+00 2.00000000000000000
1 : Exec query="paramquery"
    seqno@hdr entryno@body depar@body
    2.000000000 1.000000000 -221.5900000
    2.000000000 2.000000000 1.060000000
1 : oldvalue at end > 2.00000000000000000

$ ./paramquery1.x DB paramquery seqno 3
... END of OPEN_DB
```
5.6 Using parameterized queries

The second program (paramquery2.F90) produces similar output. Note, that the oldvalue-variable (i.e. the parameterized variable) retains its value throughout the program unlike in the first example. This is because its get changed only temporarily by the ODB_select:

```
$ odbf90 -c paramquery2.F90
$ odbcomp -lDB paramquery.sql
$ odbf90 paramquery2.o -lDB -o paramquery2.x
$ ./paramquery2.x DB paramquery seqno 1
...
END of OPEN_DB
1 : oldvalue at begin > 0.000000000000000000E+00
1 : Exec query="paramquery"
    seqno@hdr  entryno@body  depar@body
     1.000000000  1.000000000  -85.8600000
     1.000000000  2.000000000  1.800000000
1 : oldvalue at end > 0.000000000000000000E+00
$ ./paramquery2.x DB paramquery seqno 2
...
END of OPEN_DB
1 : oldvalue at begin > 0.000000000000000000E+00
1 : Exec query="paramquery"
    seqno@hdr  entryno@body  depar@body
     2.000000000  1.000000000  -221.5900000
     2.000000000  2.000000000  1.060000000
1 : oldvalue at end > 0.000000000000000000E+00
$ ./paramquery2.x DB paramquery seqno 3
...
END of OPEN_DB
1 : oldvalue at begin > 0.000000000000000000E+00
1 : Exec query="paramquery"
    seqno@hdr  entryno@body  depar@body
     3.000000000  1.000000000  -123.3200000
1 : oldvalue at end > 0.000000000000000000E+00
```

In the following is shown how queries would look like in a genuine relational databases (like MySQL), you would need to use @-variable instead of $-variable and the parameterized SQL-file would look like:
-- Set parameter
SET @seqno = 0;

-- Increase @seqno by 1
SET @seqno = @seqno + 1;

-- paramquery
SELECT hdr.seqno, entryno, depar
FROM hdr NATURAL JOIN body
WHERE hdr.seqno = @seqno;

-- Increase @seqno by 1 etc.

5.7 Updating database

Very often it is necessary to modify some values in the database. Modification can be unconditional or conditional. That former means we change column entry values regardless of its and any other columns’ existing value. On the other hand the latter means we want to fulfil in essence some WHERE-condition, before updates takes place.

Using genuine relational database engines we could make single value updates using the following commands (the last SQL is used for verification purposes only):

-- 1st update
UPDATE body SET bias=0;

-- 2nd update
UPDATE body SET bias=30
WHERE varno = 110;

-- 3rd update
UPDATE hdr NATURAL JOIN body
SET bias=abs(0.25*depar)
WHERE time = 100000
AND varno != 39;

-- To verify changes
SELECT statid, varno, depar, bias
FROM hdr NATURAL JOIN body;

The ODB/SQL doesn’t currently support the UPDATE-statement and all updates need to be performed by using the CREATE VIEW-mechanism. This allows to execute the SQL, bring the associated data to user space (Fortran90 program), apply modifications and pass modified
5.7 Updating database

columns back to database using function ODB_put. In fact this mechanism, albeit requiring a Fortran90 program, is much richer than the UPDATE-statement shown before, because we can make multi-value updates. That is to say, a single column (say bias at table body) is not restricted in getting constant values, or just values that stem from some other variable, but can have any values supplied by the application program possibly through a complex algorithm.

In this example you will learn how to

- ODB_open: Open a database in 'UNKNOWN'-mode
- ODB_select: Execute and find out allocation dimensions of the request
- ODB_get: Fetch data satisfying the request into the user space
- ODB_put: Propagate modifications back to the database using the active request (view)
- ODB_cancel: Deactivate the query and free internal indices
- ODB_close: Save updated data and close the database

Note that this example demonstrates how to use ODB_select without ODB_addview, since in the latest version of the ODB software the ODB_select (also ODB_getnames and ODB_varindex) calls ODB_addview automatically if the SQL-query has not yet been registered. In the dynamic linking platforms, the use of ODB_addview is still needed, if one wants to re-introduce the same queries over again while running the application.

We intend to simulate the UPDATE-statements shown before to prove that the action available for database engines can take place by using proper collaboration with ODB/SQL-statements and the ODB Fortran90 application.

At first sight it would sound easier to launch UPDATE-statements similar to the ones available in the genuine database engines’ client-server environment than using an embedded approach by means of ODB Fortran90 application. However, when an application, like IFS 4D-Var is already present, and is MPI-parallelized and performs very complicated calculations before updating database values, then the approach available through the ODB software becomes viable.

Now pay consider the 0\textsuperscript{th} column of the array allocated for ODB_get. This is so called control-column and contains internal information for ODB to perform updates through function ODB_put. The column zero consists of composite numbers denoting the pool (partition) and row index number, where the data originated from. That is to say, you can safely sort the array with respect to any data column and the function ODB_put finds its way to propagate data back to the database.

To drive the updates in this example, we need a set of CREATE VIEW-statements They are shown in the following:
CREATE VIEW update_1 AS
SELECT bias
FROM body;

CREATE VIEW update_2 AS
SELECT bias
FROM body
WHERE varno = 110;

CREATE VIEW update_3 AS
SELECT bias,
    depar READONLY
FROM hdr, body
WHERE time = 100000
    AND varno != 39;

CREATE VIEW verify AS
SELECT time,varno,depar,bias
FROM hdr,body;

Note in the third update the use of READONLY-keyword. This is ODB/SQL-language specific and prevents a Fortran90 application from performing updates to this column (depar). We could have achieved the same by using the colput(:) logical vector in the call to function ODB_put. The ODB Fortran90 application is fine-tuned for these queries and could look as follows:

PROGRAM updatedb
!*** updatedb.F90 ***
USE odb_module
implicit none
REAL(8), ALLOCATABLE :: x(:,:)
INTEGER(4) :: myproc,nproc,npools,jp,h
INTEGER(4) :: nrows, ncols, nra, rc
INTEGER(4) :: jupd, jr, jc, nc
character(len=64) :: dbname, verify, upd(3) = &
    & (/’update_1’,’update_2’,’update_3’/)
character(len=64), allocatable :: colnames(:)
rc = ODB_init(myproc=myproc,nproc=nproc)
CALL getarg(1,dbname)  ! Database name
npools = 0 ! Gets true value from ODB_open()
h = ODB_open(dbname,’UNKNOWN’,npoools)
verify = ’verify’
nc = ODB_getnames(h,verify,'name')
ALLOCATE(colnames(nc))
nc = ODB_getnames(h,verify,'name',colnames)
jp = -1 ! Here: We apply for all pools in one go
do jupd=1,size(upd)
  !-- Update#<jupd>
  write(*,*)'Exec update#',jupd
  rc = ODB_select(h,upd(jupd),nrows,ncols,nra=nra,poolno=jp)
  ALLOCATE(x(nra,0:ncols))
  rc = ODB_get(h,upd(jupd),x,nrows,ncols,poolno=jp)
  !-- Perform the update
  CALL update(jupd,x,nrows,ncols,nra)
  ! The following would update data in memory, not on disk
  rc = ODB_put(h,upd(jupd),x,nrows,ncols,poolno=jp)
  DEALLOCATE(x)
  rc = ODB_cancel(h,upd(jupd),poolno=jp)
  !-- Verify changes (... which are not on disk yet, though)
  write(*,*)'Exec verify'
  rc = ODB_select(h,verify,nrows,ncols,nra=nra,poolno=jp)
  ALLOCATE(x(nra,0:ncols))
  rc = ODB_get(h,verify,x,nrows,ncols,poolno=jp)
  do jr=1,nrows
    write(*,'((4a15))') (trim(colnames(jc)),jc=1,nc)
    !enddo
  DEALLOCATE(x)
  rc = ODB_cancel(h,verify,poolno=jp)
enddo
DEALLOCATE(colnames)
rc = ODB_close(h,save=.TRUE.) ! Goes to disk only now
rc = ODB_end()
CONTAINS
SUBROUTINE update(kupd,x,nrows,ncols,nra)
implicit none
INTEGER(4), intent(in) :: kupd,nrows,ncols,nra
REAL(8), intent(inout) :: x(nra,0:ncols)
if (kupd == 1) then
  x(1:nrows,1) = 0 ! UPDATE body SET bias=0;
else if (kupd == 2) then
  x(1:nrows,1) = 30 ! UPDATE body SET bias=30 WHERE ...
else if (kupd == 3) then ! ... SET bias=abs(0.25*depar)
  x(1:nrows,1) = abs(0.25*x(1:nrows,2))
endif
END SUBROUTINE update
END PROGRAM updatedb
When we run the program (again: after appropriate compilations), the output will be as follows:

```bash
$ odbcomp -lDB update.sql
$ odbf90 -c updatedb.F90
$ odbf90 updatedb.o -lDB -o updatedb.x
$ ./updatedb.x DB
...
OPEN_DB: handle= 1
DataBase-name="DB"
...
newdb ? F
readonly ? F
no. of global pools = 1
...
END of OPEN_DB

Exec update# 1

<table>
<thead>
<tr>
<th>time@hdr</th>
<th>varno@body</th>
<th>depar@body</th>
<th>bias@body</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000.00</td>
<td>110.00000</td>
<td>-85.860000</td>
<td>0.0000000</td>
</tr>
<tr>
<td>100000.00</td>
<td>39.000000</td>
<td>1.8000000</td>
<td>0.0000000</td>
</tr>
<tr>
<td>130000.00</td>
<td>110.00000</td>
<td>-221.59000</td>
<td>0.0000000</td>
</tr>
<tr>
<td>130000.00</td>
<td>39.000000</td>
<td>1.0600000</td>
<td>0.0000000</td>
</tr>
<tr>
<td>130000.00</td>
<td>110.00000</td>
<td>-123.32000</td>
<td>0.0000000</td>
</tr>
</tbody>
</table>

Exec update# 2

<table>
<thead>
<tr>
<th>time@hdr</th>
<th>varno@body</th>
<th>depar@body</th>
<th>bias@body</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000.00</td>
<td>110.00000</td>
<td>-85.860000</td>
<td>30.000000</td>
</tr>
<tr>
<td>100000.00</td>
<td>39.000000</td>
<td>1.8000000</td>
<td>0.0000000</td>
</tr>
<tr>
<td>130000.00</td>
<td>110.00000</td>
<td>-221.59000</td>
<td>30.000000</td>
</tr>
<tr>
<td>130000.00</td>
<td>39.000000</td>
<td>1.0600000</td>
<td>0.0000000</td>
</tr>
<tr>
<td>130000.00</td>
<td>110.00000</td>
<td>-123.32000</td>
<td>30.000000</td>
</tr>
</tbody>
</table>

Exec update# 3

<table>
<thead>
<tr>
<th>time@hdr</th>
<th>varno@body</th>
<th>depar@body</th>
<th>bias@body</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000.00</td>
<td>110.00000</td>
<td>-85.860000</td>
<td>21.465000</td>
</tr>
<tr>
<td>100000.00</td>
<td>39.000000</td>
<td>1.8000000</td>
<td>0.0000000</td>
</tr>
<tr>
<td>130000.00</td>
<td>110.00000</td>
<td>-221.59000</td>
<td>30.000000</td>
</tr>
<tr>
<td>130000.00</td>
<td>39.000000</td>
<td>1.0600000</td>
<td>0.0000000</td>
</tr>
<tr>
<td>130000.00</td>
<td>110.00000</td>
<td>-123.32000</td>
<td>30.000000</td>
</tr>
</tbody>
</table>
```

Note that we verify the changes after every call to function ODB_put. Furthermore, it is important to note that changes from calling function ODB_put go to disk only after ODB_close, ODB_swapout or ODB_release, all with save-option set to TRUE.
5.8 Adding more data

The last ODB Fortran90 example shows how to create a program and a set of queries which would add new data rows to the database, if they didn’t already exist there. As a practical case, consider that you want to a new synoptic station, but you are not quite sure whether you already have this station in the database. And you want to avoid duplicate definitions. You will learn a few new function in this subsection and just how to find out via ODB_varindex the column index belonging to a particular set of columns. The use of this function improves portability and makes your program more generic.

We start with a query, which will be used to test whether a synop station 07088 already exists in database. If it doesn’t, the associated will be appended to database. However, if it does exist, we will modify depar and bias column values of the corresponding surface pressure. The SQL-request looks as follows:

```sql
// To check for existence of a station ident
SET $all = 0;

READONLY;
CREATE VIEW checkupd AS
SELECT "@hdr", varno, depar, bias UPDATED
FROM hdr,body
WHERE (statid = '07088' AND varno = 110)
OR $all;
```

The subroutine feedrow.F90 used by the main program is as follows. We had to use ODB_remove-technique, because there is a bug in the current ODB software, when this kind of append is requested:

```fortran
SUBROUTINE feedrow(h,tblname,poolno,nd,d)
USE odb_module
implicit none
INTEGER(4), intent(in) :: h,poolno,nd
REAL(8),intent(in) :: d(nd)
character(len=*):: tblname
INTEGER(4) :: nrows, ncols, jp, rc
REAL(8), ALLOCATABLE :: x(:,:),
jp = poolno
rc = ODB_getsize(h,tblname,nrows,ncols,poolno=jp)
if (ncols/=nd) CALL ODB_abort('feedrow'//trim(tblname),'/ncols/=nd',ncols-nd)
```
Please note that you currently cannot parametrize character strings in your WHERE-condition.

The program `appenddb.F90` looks as follows:

```fortran
!*** appenddb.F90 ***
USE odb_module
implicit none
REAL(8), ALLOCATABLE :: x(:,:)
REAL(8) bodyoffset, bodylen, bias, all, seqno
INTEGER(4) :: myproc,nproc,npools,jp,h
INTEGER(4) :: nrows, ncols, nra, rc
INTEGER(4) :: nrows_body, ncols_body
INTEGER(4) :: jupd, jr, jc, nc, idx(1)
character(len=64) :: dbname, queryname, tblname, chbias(1)
character(len=64), allocatable :: colnames(:)
rc = ODB_init(myproc=myproc,nproc=nproc)
CALL getarg(1,dbname) ! Database name
npools = 0 ! Gets true value from ODB_open()
h = ODB_open(dbname,’OLD’,npools)
queryname = ’checkupd’
jp = 1 ! Here: We apply for pool#1 only
write(*,*)’=== Check existence’
rc = ODB_cancel(h,queryname,poolno=jp)
rc = ODB_select(h,queryname,nrows,ncols,nra=nra,poolno=jp)
bias = 10; nc = ncols
if (nrows == 0) then
   rc = ODB_cancel(h,queryname,poolno=jp)
   write(*,*)’==&gt; Station did not exist; adding it now’
   !*** Get the total length of @body so far
   rc = ODB_getsize(h,’@body’,nrows_body,ncols_body,poolno=jp)
   bodyoffset = nrows_body ! The existing max. rows
   bodylen = 1 ! We will only add one new row into ’@body’
   !*** Get the next free sequence number
   rc = ODB_getsize(h,’@hdr’,nrows,ncols,poolno=jp)
   seqno = nrows + 1 ! Next free sequence number
   CALL feedrow(h,’@hdr’,jp,nrows,ncols, &
                  ! seqno date time statid ...
                  ! & (seqno,20010101d0,140000d0,8H 07088, &
                  ! & 59.18d0, -7.45d0, bodyoffset, bodylen/))
```
5.8 Adding more data

CALL feedrow(h,’@body’,jp,ncols_body, &
! entryno, varno, press, obsvalue, depar, obs_error, bias
& (/1d0, 110d0, 857.05d0, 99530.00d0, -88.90d0, 80.03d0, bias/))
else ! Was present ==> we update the bias by increasing by it 1% + bias
write(*,*)’==> Station did indeed exist; just updating the bias’
chbias(1) = ‘bias@body’ ; idx(1) = 0
rc = ODB_varindex(h,queryname,chbias(:),idx=idx(:))
if (idx(1) == 0) CALL ODB_abort(’appenddb’,’Bias column not present’,-1)
ALLOCATE(x(nra,0:ncols))
rc = ODB_get(h,queryname,x,nrows,ncols,poolno=jp)
x(1:nrows,idx(1)) = x(1:nrows,idx(1))*0.01 + bias ! Updated bias
rc = ODB_put(h,queryname,x,nrows,ncols,poolno=jp)
DEALLOCATE(x)
rc = ODB_cancel(h,queryname,poolno=jp)
endif
write(*,*)’=== Verifying contents’
ALLOCATE(colnames(nc))
nc = ODB_getnames(h,queryname,’name’,colnames)
all = 1 ! Ask for *all* rows now
rc = ODB_select(h,queryname,nrows,ncols,nra=nra,poolno=jp, &
    & setvars=(`/all/`), values=(`/all/`))
ALLOCATE(x(nra,0:ncols))
rc = ODB_get(h,queryname,x,nrows,ncols,poolno=jp)
write(*,’((3a20))’) (trim(colnames(jc)),jc=1,nc)
do jr=1,nrows
    write(*,’(1p,(5x,3(1x,g20.10)))’)) x(jr,1:ncols)
endo
DEALLOCATE(x)
rc = ODB_cancel(h,queryname,poolno=jp)
DEALLOCATE(colnames)
rc = ODB_close(h,save=.TRUE.) ! Goes to disk only now
rc = ODB_end()
END PROGRAM appenddb

The output listing, when the synop station was not yet in database is as follows:

$ odbcomp -lDB checkupd.sql
$ odbf90 -c appenddb.F90 feedrow.F90
$ odbf90 appenddb.o feedrow.o -lDB -o appenddb.x
$ ./createdb.x DB # Re-create the database to ensure ...
$ ./populate.x DB # ... a fresh start
$ ./appenddb.x DB # this is the *first* run; station not there
...
END of OPEN_DB
=== Check existence
==> Station did not exist; adding it now
=== Verifying contents
The second output shows how the `bias` column gets updated when the synop station in concern was previously already in database. We repeat only the changed parts of output:

```
$ ./appenddb.x DB  # this is the *second* run; station already there
...
END of OPEN_DB
=== Check existence
==> Station did indeed exist; just updating the bias
=== Verifying contents

seqno@hdr    date@hdr    time@hdr
statid@hdr   lat@hdr     lon@hdr
LINKOFFSET(body)@hdr  LINKLEN(body)@hdr  varno@body
depar@body   bias@body

... (unchanged) ...

4.0000000000  20010101.00  140000.0000
6.0135615738-154 49.82000000  3.200000000
2.0000000000  2.0000000000
-221.5900000  0.000000000
6.0135615738-154 49.78000000  4.630000000
4.0000000000  1.0000000000
-123.3200000  0.000000000
4.0000000000  20010101.00
6.0135615738-154 59.18000000  -7.450000000
5.0000000000  1.0000000000
-88.900000000 10.00000000
```
5.9 Archiving and removing database

The final subsection in this section illustrates how to archive and remove ODB database in the file system level, and how possibly to re-introduce a database on some other place and/or machine.

The default location of the database is the current directory. You can change this by defining either database specific environment variables

- ODB_DATAPATH_$dbname (see p.191) for data files
- ODB_SRCPATH_$dbname (see p.194) for metadata

or use the generic location definition variables

- ODB_DATAPATH (see p.190) for data files
- ODB_SRCPATH (see p.194) for metadata

The former definition takes precedence, if specified. If none of these are defined, the ODB software will decide to use the current directory during ODB_open. You need have access to data directories, and very often also the right to write/update files.

It is recommended to label the root database directory to start with the database name, followed by optional qualifiers, like observational group, time of analysis, and so on. For example, ECMWF’s IFS system creates large AIRS ECMA-database into root directory prefixed with $WDIR/ECMA.airs, where the $WDIR denotes the working directory for particular $BASETIME i.e. analysis period.

Since ODB database is normally located under a single root directory, it is easy to back it up, create a tarfile, remove it or transfer it to another machine.

Suppose one has a database DB again located under the directory /local/disk/DB. In order to create tarfile and remove it, one only needs to do the following (the mycomp$ denotes the command line prompt on "my"-machine):

```
mycomp$ cd /local/disk
mycomp$ tar -cvf DB.tar DB
mycomp$ rm -rf DB
```

You can untar your database on some other machine and start working with it by:
mycomp$ rcp /local/disk/DB.tar username@host:/remote/dir/
mycomp$ rlogin host -l username
host$ cd /remote/dir/
host$ tar -xvf DB.tar
host$ cd DB
host$ export ODB_DATAPATH_DB=/remote/dir/DB
host$ export ODB_SRCPATH_DB=/remote/dir/DB
6 ODB interface to IFS

The IFS analysis uses two different database layouts called ECMA and CCMA. In addition, some pre- and postprocessing utilities may benefit from utilizing the ECMASCR layout. So far we have decided to have exactly the same layout in the ECMASCR than in the ECMA, but this is generally up to the user what to do with the ECMASCR. You can find more about these database layouts from section 7.

The analysis is divided into several steps. The first step is called Screening or the First trajectory (i.e. traj#0). It uses the ECMA layout to screen all observations in order to mask out bad or undesired observations. After Screening the remaining observations are transferred into the CCMA layout. This is a subset of the ECMA database both in terms of volume and the number of tables and columns.

The CCMA database is used all through the Minimization process, except that in the Final trajectory, the ECMA steps in again, if needed. Currently ECMWF uses the ECMA database for the Final trajectory run.

6.1 Architecture

The interface between IFS and ODB is indirect. That is to say, IFS sees ODB via Fortran90 module yomdb, which resides in the file $SRC/ifs/module/yomdb.F90. This module essentially defines data structures, where observational data is copied from each SQL-
query. The module file includes another important file, $\text{SRC/ifs/common/yomdb_-vars.h}$, where the observational data arrays are defined for use by data structure $o_(it)$ (i.e., one element for each OpenMP-thread $it$). Most of the symbols found in that file are pre-processing macro-symbols, which get filtered through the file $\text{SRC/ifs/common/yomdb_-defs.h}$, while the Fortran90 routines include them via $\text{SRC/ifs/common/openmp_-obs.h}$. The latter file also includes the file $\text{SRC/ifs/common/itdef.h}$.
6.2 Observational arrays, MDB-variables and MLNK-pointers

Data is transferred between ODB and IFS using observational arrays described in the following table. The basic idea is that the ODB interface layer to IFS fills these arrays with data that had just been extracted from a database. IFS then accesses and often modifies these arrays before passing them back to the database via the interface layer (see also figure 18).

Table 6: Observational arrays in the ODB interface to IFS

<table>
<thead>
<tr>
<th>Array name</th>
<th>Description</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROBHDR</td>
<td>Data from hdr-related tables</td>
<td>REAL(8)</td>
</tr>
<tr>
<td>MOBHDR</td>
<td>The same as ROBHDR, but contains only INTEGER columns</td>
<td>INTEGER(4)</td>
</tr>
<tr>
<td>ROBODY</td>
<td>Data from body-related tables</td>
<td>REAL(8)</td>
</tr>
<tr>
<td>MOBODY</td>
<td>The same as ROBODY, but contains only INTEGER columns</td>
<td>INTEGER(4)</td>
</tr>
<tr>
<td>ROBSU</td>
<td>Usually similar to ROBHDR, but used only in setup/special routines</td>
<td>REAL(8)</td>
</tr>
<tr>
<td>MOBSU</td>
<td>The same as ROBSU, but contains only INTEGER columns</td>
<td>INTEGER(4)</td>
</tr>
<tr>
<td>ROBDDR</td>
<td>Array for ddr-table data for convenience</td>
<td>REAL(8)</td>
</tr>
<tr>
<td>SATHDR</td>
<td>Satellite specific hdr-information</td>
<td>REAL(8)</td>
</tr>
<tr>
<td>SATPRED</td>
<td>Satellite predictor hdr-info, is often aliased to SATHDR</td>
<td>REAL(8)</td>
</tr>
<tr>
<td>SATBODY</td>
<td>Satellite specific body-information</td>
<td>REAL(8)</td>
</tr>
<tr>
<td>MLNKH2B</td>
<td>Runtime link between hdr and body related information extracted via ODB/SQL</td>
<td>INTEGER(4)</td>
</tr>
</tbody>
</table>

Advance warning: please note that all the INTEGER-arrays (except MLNKH2B) are subject to be removed from the ODB/IFS-interface in the near future release, since they can hold only 31-bits of control word information in the zeroth column. The increase in the number of observations and pools require the use of 63-bit control word information. This can be achieved by removing references to these arrays while removing ODB-functions ODB_i get and ODB_iput – the INTEGER(4) equivalents of ODB_get and ODB_put, respectively.

The MDB-variables and MLNK-pointers (see more in section 7) are used to address the particular column of the observational arrays in IFS. The values for the MDB/MLNK-pointers are registered once in the routine INITMDB and their values get dynamically assigned to a column number in the routine MAPVARDB for each OpenMP-thread. This means that they get fresh values in every call to GETDB and are reset back to a missing (or purposely a bad)
value in PUTDB, so that if they are referred outside the GETDB–PUTDB -block, the program will crash and normally produce a traceback.

All @LINK-entries have use MLNK-pointers, which have two elements: offset and length. For a @LINK column entry child@parent, the MLNK-pointers from master table (parent) to child table (child) would be:

1. MLNK_PARENT2CHILD(1) denotes the offset (child.offset@parent)
2. MLNK_PARENT2CHILD(2) denotes the length (child.len@parent)

Internally the ODB software knows the offset to child-table as LINKOFFSET(child)@parent (or just LINKOFFSET(child) when unique), and the length respectively as LINKLEN(child)@parent (or LINKLEN(child)). These notations need to be used in functions like ODB_varindex, when searching the column number for the given @LINK-object (i.e. offset and/or length).

### 6.3 The main interface functions

The IFS accesses database(s) without calling the ODB core functions directly. This interface layer passes data from ODB internal data structures into observational arrays for use by IFS. There are a few advantages of coding this way:

1. IFS does not depend directly on ODB-modules i.e. can be compiled without referencing to ODB software,
2. IFS often needs simultaneous access to multiple data queries at once (ROBHDR, ROBODY, ...), therefore the interface is coded to allow access retrievals – that is: multiple SQL data queries,
3. Underlying change in ODB core libraries does not (normally) affect IFS code.

The interface layer between ODB and IFS consists of the following subroutines:

- **OPENDB**: Open database (p.117)
- **SETACTIVEDB**: Change currently active database (p.118)
- **INITMDB**: Initialize MDB-pointer addresses (p.119)
- **CTXINITDB**: Initialize mapping of retrievals to data queries (SQL) (p.119)
- **GETDB**: Execute a set of SQLs (i.e. a retrieval) and get data to observational arrays for use by IFS (p.121)
- **PUTDB**: Pass data from observational arrays back to in-memory database arrays (p.123)
- **CLOSEDB**: Close currently active database with or without updating it (p.125)
6.3 The main interface functions

These routines are associated with the module yomdb and another module context, which is resides in the file $SRC/odb/module/context.F90.

In addition there is observational data shuffle routine SHUFFLEDB, which can transform data between two different database, like ECMA and CCMA. Its source code resides in the file $SRC/odb/cma2odb/shuffledb.F90. Other routines to be discussed: $SRC/odb/cma2odb/swapout.F90, $SRC/odb/cma2odb/store.F90, $SRC/odb/cma2odb/load.F90 and $SRC/odb/lib/msgpass_obsdata.F90.

6.3.1 OPENDB

Usage of the subroutine OPENDB is as follows:

```
CALL OPENDB ( CDNAME, CDSTAT, KINFO, KINFOLEN )
```

Subroutine OPENDB opens a connection to one of the known databases (see 7). Up to JP-MAXSIMDB can be opened simultaneously for the ODB interface to IFS. Routine OPENDB is a wrapper routine built on the top of ODB_open. Another purpose is to initialize the IFS interface layer. It also makes sure that data retrievals under OpenMP-locks get done properly i.e. it initializes the derived datatype array o_(::) from the module yomdb.
(see $SRC/ifs/module/yomdb.F90). The source file resides in $SRC/odb/cma2odb/opendb.F90.

The parameters denote:

**CDNAME**

CHARACTER (LEN=*) , INTENT (IN)

Database name. Currently recognized names are ECMA, CCMA and ECMASCR.

**CDSTAT**

CHARACTER (LEN=*) , INTENT (IN)

Opening mode. Allowed values are 'NEW', 'OLD' and 'READONLY'.

**KINFO(KINFOLEN)**

INTEGER (4) , INTENT (INOUT)

Integer information vector. To supply or return useful information, the length (KINFOLEN) should be set to 2. In that case, the first element (KINFO(1)) contains the number of pools and the second element (KINFO(2)) contains (upon return) the value of the maximum number of updates specified in the data layout (the parameter $nmxupd).

**KINFOLEN**

INTEGER (4) , INTENT (IN)

Number of elements in KINFO-vector.

**Examples:**

```fortran
INTEGER(4) :: info(2)
CALL OPENDB('ECMA', 'OLD', info, size(info))
CALL OPENDB('CCMA', 'NEW', info, size(info))
CALL OPENDB('ECMASCR', 'READONLY', info, size(info))
```

### 6.3.2 SETACTIVEDB

Usage of the subroutine SETACTIVEDB is as follows:

```fortran
CALL SETACTIVEDB ( CDNAME )
```

This changes the currently active database, which is the database to which all subsequent GETDB, PUTDB etc. calls will be targeted. The source file resides in $SRC/odb/cma2odb/setactivedb.F90.

The parameters denote:
6.3 The main interface functions

6.3.1 CDNAME

**CDNAME**

*CHARACTER (LEN=*), INTENT (IN)*

The name of database which is to become the currently active database

**Examples:**

-- Set active database to ECMA and work with it
CALL SETACTIVEDB('ECMA')

-- Now work with CCMA database for a change
CALL SETACTIVEDB('CCMA')

6.3.3 INITMDB

Usage of the subroutine INITMDB is as follows:

```fortran
CALL INITMDB(KNXMUPD)
```

The purpose of this routine is to map the MDB column address variables into data layout variables of the recognized databases. This routine can be found from `$SRC/odb/cma2odb/initmdb.F90`. It is used in association with the (generic) data layout file `$SRC/odb/ddl/cma.h` for the ECMA, ECMASC and CCMA databases. The parameters denote :

**KNMXUPD**

*INTEGER (4), INTENT (IN)*

Supplies the actual number of updates. The value must be less or equal than `$NMXUPD` defined in `$SRC/odb/ddl/cma.h`

**Examples:**

```fortran
INTEGER(4) :: imxupd
imxupd = 2 ! Maximum number of updates in this run
CALL INITMDB(imxupd)
```

6.3.4 CTXINITDB

Usage of the subroutine CTXINITDB is as follows:
CALL CTXINITDB ( CDNAME, CDRETR, KVERSION, KNMXUPD )

This routine defines all recognized data retrievals to be used in GETDB and PUTDB and maps them into the given set of SQL-query files, that reside under the directory $SRC/odb/ddl/. Retrievals are identified from GETDB and PUTDB by their name (CDNAME) and version number (KVERSION). The source file resides in $SRC/odb/cma2odb/ctxinitdb.F90. The parameters denote:

- **CDNAME**: CHARACTER (LEN=*), INTENT (IN)
  Database name. Currently supported names are ECMA, CCMA and ECMASCR

- **CDRETR**: CHARACTER (LEN=*), INTENT (IN)
  Retrieval name

- **KVERSION**: INTEGER (4), INTENT (IN)
  Retrieval version number. This number is often the same as the current (minimization or trajectory) update number

- **KNMXUPD**: INTEGER (4), INTENT (IN)
  The actual number of minimizations to be performed (or trajectory updates) in the current 4D-Var-run

**Examples:**

```fortran
! The very first time (in opendb.F90 for example)
integer(4) :: imxupd
imxupd = <max. no. of updates in this run>
CALL CTXINITDB(cdname, '', 0, imxupd)

! or normally (like in getdb.F90)
character(len=*), intent(in) :: cdretr
integer(4), intent(in) :: kversion
integer(4) :: imxupd
imxupd = <max. no. of updates in this run>
CALL CTXINITDB('ECMA', cdretr, kversion, imxupd)
```
6.3.5 GETDB

Usage of the subroutine GETDB is as follows:

```fortran
CALL GETDB ( CDRETR, KVERSION, KRET, KINFO, KINFOLEN &
& , PINFO, KPINFOLEN, KSET, KTSLOT, KPOOLNO &
& , KOBSTYPE, KCODETYPE, KSENSOR )
```

Subroutine GETDB executes one or more SQL-queries for the active database under the cover of the ODB to IFS interface. The routine copies data for each OpenMP-thread to observational arrays declared in module yomdb. For each context i.e. (retrieval,version)-pair for there must be a set of SQL-queries available. Initialization of contexts is done in routine CTXGETDB. The source file resides in $SRC/odb/cma2odb/getdb.F90. The parameters denote:

- **CDRETR**: CHARACTER(LEN=*) , INTENT(IN)
  Retrieval name initialized in subroutine CTXGETDB. This parameter with KVERSION determines which SQL-queries will be invoked.

- **KVERSION**: INTEGER(4) , INTENT(IN)
  Retrieval version number.

- **KRET**: INTEGER(4) , INTENT(INOUT)
  Return code, which (if non-negative) denotes number of rows placed in the first observational array in the sequence of SQL-queries executed. That is to say, it is mostly the number of rows in ROBHDR-array.

- **KINFO(KINFOLEN)**: INTEGER(4) , INTENT(INOUT)
  Integer information vector. Its length (KINFOLEN) should normally be set to zero (0). However, if non-zero, then the elements are interpreted as follows:
  The KINFO(1) denotes parameterized variable $hdr_min, the KINFO(2) variable $hdr_max, and the KINFO(3) variable $all. These variables are only used in a very few SQL-queries.

- **KINFOLEN**: INTEGER(4) , INTENT(IN)
  Number of elements in the integer information vector KINFO. Normally zero.
**PINFO (KPINFOLEN)**
REAL(8), INTENT(INOUT)
Auxiliary information vector. Not used at the moment, but could in the future contain (say) station identification information or some other information, which requires 8-bytes of storage.

**KPINFOLEN**
INTEGER(4), INTENT(IN)
Number of elements in the real information vector PINFO. Normally zero.

**KSET**
INTEGER(4), INTENT(IN)
A set number as calculated in the IFS routine ECSET. Refers to the parameterized variable \$kset, and if present in the SQL's WHERE-condition, the value of KSET will temporarily replace the default value, for the current OpenMP-thread only. If set to a missing data value (MDIDB) or a non-positive number, then the default value of parameterized variable \$kset does not get changed at all, unless the value is $1$, which denotes any set number.

**KTSLOT**
INTEGER(4), INTENT(IN)
The 4D-Var timeslot number. Refers to the parameterized variable \$tslot, and if present in the SQL's WHERE-condition, the value of KTSLOT will temporarily replace the default value, for the current OpenMP-thread only. If set to a missing data value (MDIDB) or a non-positive number, then the default value of parameterized variable \$tslot does not get changed at all, unless the value is $-1$, which denotes any timeslot number.

**KPOOLNO**
INTEGER(4), INTENT(IN)
The pool number to which the SQLs are to be applied. Usually all pools are scanned, which corresponds to the value of $1$.

**KOBSTYPE**
INTEGER(4), INTENT(IN)
The particular (CMA) observation type (see table B.2). Refers to the parameterized variable \$obstype, and if present in the SQL's WHERE-condition, the value of KOBSTYPE will temporarily replace the default value, for the current OpenMP-thread only. If set to a missing data value (MDIDB) or a non-positive number, then the default value of parameterized variable \$obstype does not get changed at all, unless the value is $-1$, which denotes any observation type.
6.3 The main interface functions

KCODETYPE INTEGER(4), INTENT(IN)
The particular (CMA) observation codetype. Refers to the parameterized variable $codetype$, and if present in the SQL’s WHERE-condition, the value of KCODETYPE will temporarily replace the default value, for the current OpenMP-thread only. If set to a missing data value (MDIDB) or a non-positive number, then the default value of parameterized variable $codetype$ does not get changed at all, unless the value is $-1$, which denotes any observation codetype.

KSENSOR INTEGER(4), INTENT(IN)
The particular satellite sensor number (see table B.4.1). Refers to the parameterized variable $sensor$, and if present in the SQL’s WHERE-condition, the value of KSENSOR will temporarily replace the default value, for the current OpenMP-thread only. If set to a missing data value (MDIDB) or a negative number, then the default value of parameterized variable $sensor$ does not get changed at all, unless the value is $-1$, which denotes any sensor.

Examples:

```
INTEGER(4) :: iversion, iret, info(0)
INTEGER(4) :: iset, itimeslot, ipoolno, icma_obstype
INTEGER(4) :: icma_codetype, isensor
REAL(8) :: zinfo(0)

iversion = 0 ! version (or sometimes update) number
iset = MDIDB ! set number N/A
itimeslot = -1 ! All
ipoolno = 2 ! Scan pool number 2 only
icma_obstype = 7 ! SATEM
icma_codetype = 210 ! ATOVS
isensor = 11 ! AIRS-data

CALL GETDB('RETRIEVAL_NAME', iversion, iret, info, 0, zinfo, 0, &
& iset, itimeslot, ipoolno, icma_obstype, icma_codetype, isensor)
```

6.3.6 PUTDB

Usage of the subroutine PUTDB is as follows:
This routine updates the database by copying data in observational arrays back to the desired cells in the database tables (that are held in memory). In practice only those columns are updated, which were not under READONLY-condition of the corresponding SQL-queries of the retrieval. Parameters denote the same as in the counterpart routine GETDB. The source file resides in file $SRC/odb/cma2odb/putdb.F90. The parameters denote:

- **CDRETR**: CHARACTER(LEN=*) , INTENT(IN)
  - Retrieval name initialized in subroutine CTXINITDB. This parameter with KVERSION determines which SQL-queries will be invoked.

- **KVERSION**: INTEGER(4) , INTENT(IN)
  - Retrieval version number.

- **KRET**: INTEGER(4) , INTENT(INOUT)
  - Return code.

- **KINFO(KINFOLEN)**: INTEGER(4) , INTENT(INOUT)
  - Integer information vector.

- **KINFOLEN**: INTEGER(4) , INTENT(IN)
  - Number of elements in the integer information vector KINFO.

- **PINFO(KPINFOLEN)**: REAL(8) , INTENT(INOUT)
  - Real information vector.

- **KPINFOLEN**: INTEGER(4) , INTENT(IN)
  - Number of elements in the real information vector PINFO.

**Examples**:

```fortran
INTEGER(4) :: iversion, iret, info(0)
REAL(8) :: zinfo(0)
iversion = 0
CALL PUTDB('RETRIEVAL_NAME', iversion, iret, info, 0, zinfo, 0)
```
6.3.7 CLOSEDDB

Usage of the subroutine CLOSEDDB is as follows:

\[
\text{CALL CLOSEDDB ( LDSAVE )}
\]

This subroutine closes connection to the currently active ODB database. The source file resides in $SRC/odb/cma2odb/closedb.F90. The parameters denote:

- **LDSAVE**: LOGICAL, INTENT(IN)

Examples:

```fortran
! Close database with possible updates going to disk
CALL CLOSEDDB(.TRUE.)

! Close database without updating changes
CALL CLOSEDDB(.FALSE.)
```

6.4 Additional interface functions

There are a few other interface functions one should be aware of. In some situations it maybe essential to understand how the routine $SRC/odb/cma2odb/ctxgetdb.F90 works, therefore we start from this subroutine.

6.4.1 CTXGETDB

Usage of the subroutine CTXGETDB is as follows:

\[
\text{CALL CTXGETDB ( KHANDLE, KCTXID, LDEXECV, KEXECV \&}
\text{& KSET, KTSLOT, KPOOLNO, KOBSTYPE, KCODETYPE \&}
\text{& KSENSOR, KHDR_MIN, KHDR_MAX, KALL, KRETCODE )}
\]

Routine CTXGETDB is one of the most important "slave"-routines in ODB interface to IFS. It invokes executions of (normally) multiple SQLs related to the current retrieval and allocates all the necessary observational arrays as instructed by the routine CTXINITDB. The routine is called from GETDB and its source code resides in $SRC/odb/cma2odb/ctxgetdb.F90. The parameters denote:
KEEXECV INTEGER(4), INTENT(IN)
The number of SQLs to be executed, at most

LDEXECV (KEEXECV) LOGICAL, INTENT(IN)
Flag vector to indicates that particular SQL needs (or needs not) to be executed

KHANDLE INTEGER(4), INTENT(IN)
Database handle

KCTXID INTEGER(4), INTENT(IN)
Context id as defined in CTXINITDB (and returned by routine CTXID)

KSET INTEGER(4), INTENT(IN)
Observational set number as defined in IFS routine ECSET ($SRC/ifs/var/ecset.F90) or \(-1\) to take all sets in one go, or MDIDB when not applicable

KT SLOT INTEGER(4), INTENT(IN)
4D-Var-timeslot number in concern, or \(-1\) for all timeslots, or MDIDB when not applicable

KPOOLNO INTEGER(4), INTENT(IN)
Apply requests to particular pool number only, or \(-1\) for all pools

KOBSTYPE INTEGER(4), INTENT(IN)
Particular CMA observation type in concern, or either \(-1\) for any observation type, or MDIDB when not applicable

KCODETYPE INTEGER(4), INTENT(IN)
Particular CMA codetype in concern, or either \(-1\) for any code-type, or MDIDB when not applicable

KSENSOR INTEGER(4), INTENT(IN)
Satellite sensor indicator, or either \(-1\) for any sensor number, or MDIDB when not applicable

KHDR_MIN INTEGER(4), INTENT(IN)
Special case: refers to retrieved minimum row number, which is usually related to table @hdr (thus the naming convention). Used with SQL WHERE-condition like $hdr_min <= #hdr <= $hdr_max
6.4 Additional interface functions

KHDR_MAX  INTEGER(4), INTENT(IN)
The same as previous one, but refers to maximum row number

KALL  INTEGER(4), INTENT(IN)
Used in some data shuffle routines to indicate whether to filter out (in SQL) all observations (=1) or just the active, CCMA-related ones (=0). In addition MeteoFrance sometime uses the value 2

KRETCODE  INTEGER(4), INTENT(OUT)
Return code to indicate how many rows retrieved for the principal data query (usually @hdr-related)

6.4.2 CTXPUTDB

Usage of the subroutine CTXPUTDB is as follows:

CALL CTXPUTDB ( KHANDLE, KCTXID, KRETCODE )

Routine CTXPUTDB is the counterpart to CTXGETDB and is called from PUTDB. The routine is responsible for passing data from allocated observational arrays back to in-memory database data structures. After this it deallocates these arrays. The source file resides in $SRC/odb/cma2odb/ctxputdb.F90. The parameters denote:

KHANDLE  INTEGER(4), INTENT(IN)
Database handle

KCTXID  INTEGER(4), INTENT(IN)
Context id

KRETCODE  INTEGER(4), INTENT(OUT)
Return code

6.4.3 GETATDB

Usage of the subroutine GETATDB is as follows:

CALL GETATDB ( CDRETR, KSET, KINFO, KINFOLEN )
The purpose of subroutine GETATDB is to get a full database table directly. It is related to subroutine GETDB, but accesses no data queries. The retrieved full table is placed in one of the observational arrays, like ROBHDR, depending on table name. The source file resides in $SRC/odb/cma2odb/getatdb.F90. The parameters denote:

**CDRETR**
CHARACTER(LEN=*)\,\,INTENT(IN)
Lowercase table name starting with a character "@"

**KSET**
INTEGER(4),\,INTENT(IN)
Used currently as incremental update number to refer to table name @update

**KINFOLEN**
INTEGER(4),\,INTENT(IN)
Number of elements in the integer information vector KINFO. Usually the length needs to be 2

**KINFO(KINFOLEN)**
INTEGER(4),\,INTENT(IN\OUT)
Integer information vector KINFO. The first element returns number of pools in database. The second element returns maximum number of updates applicable to this run.

**Examples:**

```fortran
INTEGER(4) :: info(2)
info(1) = <no. of rows needed for array> \! Here: ROBHDR
info(2) = <pool number in concern> \! Cannot be -1

!-- Allocate ROBHDR (see GETATDB-code for why ROBHDR)
CALL GETATDB('@HDR', -1, info, 2)

!-- Now ROBHDR has been allocated and it has dimensions
! info(1) x ( 1 + no. of columns in '@hdr'-table )

!-- Look at the PUTATDB-example for the rest !
```

### 6.4.4 PUTATDB

Usage of the subroutine PUTATDB is as follows:

```fortran
CALL PUTATDB ( CDRETR, KSET, KINFO, KINFOLEN )
```
6.4 Additional interface functions

Routine PUTATDB either creates a database table by fully populating it with data from appropriate observational array, or appends new data rows to the existing table. This routine is the counterpart of GETATDB and related to PUTDB, except that no data queries (SQLs) are accessed. The source file resides in $SRC/odb/cma2odb/putatdb.F90. The parameters denote:

- **CDRETR**: CHARACTER(LEN=’*’), INTENT(IN)
  Lowercase table name with character "@" prepended into the string

- **KSET**: INTEGER(4), INTENT(IN)
  Used currently as incremental update number to refer to table name @update

- **KINFOLEN**: INTEGER(4), INTENT(IN)
  Number of elements in the integer information vector KINFO.

- **KINFO(KINFOLEN)**: INTEGER(4), INTENT(INOUT)
  Integer information vector KINFO

**Examples:**

```fortran
!-- See GETATDB-example
...

!-- Fill ROBHDR somehow
...

!-- Pass ROBHDR to ODB and deallocate it
CALL PUTATDB('@HDR', -1, info, 2)
```

### 6.4.5 MAPVARDB

Usage of the subroutine MAPVARDB is as follows:

```fortran
CALL MAPVARDB (KVIEWS, CDNAME, KSIZE)
```

Routine MAPVARDB assigns column entry numbers into all (OpenMP-thread specific) MDB-pointers present in data queries CDNAME. It is important to understand that the routine INITMDB (called upon OPENDB) ensures that mapping between every column entry name
of a database and MDB-pointer addresses is established. Thss makes a constant re-assignment of MDB-pointer values possible, whenever the next SQL-query will be launched from within CTXGETDB (called by GETDB). The source file resides in $SRC/odb/cma2odb/mapvardb.F90. The parameters denote:

- **KVIEW**S** INTEGER(4), INTENT(IN)**
  Number of data queries or SQLs (i.e. views) in CDNAME

- **CDNAME(KVIEW)**S** CHARACTER(LEN=*), INTENT(IN)**
  Data query names as per CREATE VIEW for each SQL

- **KSIZE(KVIEW)**S** INTEGER(4), INTENT(IN)**
  Number of columns in each data query CDNAME

### 6.4.6 UNMAPDB

Usage of the subroutine UNMAPDB is as follows:

```fortran
CALL UNMAPDB ( IT )
```

Routine UNMAPDB resets all MDB-pointers for a given OpenMP-thread (IT) back to undefined state. A counterpart routine to MAPVARDB. The source file resides in $SRC/odb/cma2odb/unmapdb.F90. The parameters denote:

- **IT** INTEGER(4), INTENT(IN)
  The OpenMP (OML) thread id. A number between 1 and maximum number of OpenMP-threads

### 6.4.7 SWAPOUTDB

Usage of the subroutine SWAPOUTDB is as follows:

```fortran
CALL SWAPOUTDB ( CDNAME, KPOOLNO, LLSAVE )
```

Routine SWAPOUTDB us used to remove database in-memory pool (or all local pools) from memory with possible update to disk. Please note that due to inadequacies when ODB_–IO_METHOD (see p.191) is 4, then this routine does nothing due to limitations in this I/O-method – currently. The source file resides in $SRC/odb/cma2odb/swapoutdb.F90. The parameters denote:
### 6.4 Additional interface functions

<table>
<thead>
<tr>
<th>CDNAME</th>
<th>CHARACTER(LEN=*), INTENT(IN)</th>
<th>Data query or database table name</th>
</tr>
</thead>
<tbody>
<tr>
<td>KPOOLNO</td>
<td>INTEGER(4), INTENT(IN)</td>
<td>Pool number in concern. Use -1 to include all local pools</td>
</tr>
<tr>
<td>LLSAVE</td>
<td>LOGICAL, INTENT(IN)</td>
<td>Flag for indicating saving the contents of in-memory database to disk before data pools are removed from memory</td>
</tr>
</tbody>
</table>

**Examples:**

```fortran
!-- Swap the whole database ECMA out of memory without saving changes
CALL SWAPOUTDB('ECMA',-1,.FALSE.)

!-- Release database CCMA, pool number 5 from memory
! Save changes to disk
CALL SWAPOUTDB('CCMA', 5,.TRUE.)
```

#### 6.4.8 STOREDB

Usage of the subroutine STOREDB is as follows:

```fortran
CALL STOREDB(KPOOLNO)
```

Routine STOREDB is the counterpart of LOADDB routine. It is used to store current in-memory database into disk. However, the same as in LOADDB limitation applies here: if ODB_IO_METHOD (see p.191) is 4, then this routine does nothing due to limitations in this I/O-method, currently. The source file resides in `$SRC/odb/cma2odb/storedb.F90`. The parameters denote:

| KPOOLNO | INTEGER(4), INTENT(IN) | Pool number in concern. If set to -1, then all (locally owned) pools will be loaded into memory |

**Examples:**

```fortran
!-- Save all pools from ECMA-database into disk
CALL SETACTIVEDB('ECMA')
CALL STOREDB(-1)

!-- Store only pool number 7 from the current database
CALL STOREDB(7)
```
6.4.9 LOADDB

Usage of the subroutine LOADDB is as follows:

\[
\text{CALL LOADDB ( KPOOLNO )}
\]

Routine LOADDB is rarely used as opposed to its counterpart STOREDB. The routine is a high-level implementation of ODB-function ODB_load to load particular (or all) locally owned pools from disk into memory. However, there is an exception: if ODB_IO_METHOD (see p. 191) is 4, then this routine does nothing due to limitations in this I/O-method, currently. The source file resides in $SRC/odb/cma2odb/loaddb.F90. The parameters denote:

- **KPOOLNO** INTEGER(4), INTENT(IN)
  Pool number in concern. If set to -1, then all (locally owned) pools will be loaded into memory

Examples:

```fortran
!-- Load database CCMA, pool number 3 into memory
CALL SETACTIVEDB('CCMA')
CALL LOADDB(3)
```

6.5 Shuffling observations between two databases

The following subroutines are mainly used when database CCMA is created from database ECMA. Routines involved in this context are $SRC/odb/cma2odb/shuffledb.F90 and $SRC/odb/cma2odb/xchangedatadb.F90. Furthermore information accumulated during the 4D-Var-minimization process needs to be passed back from CCMA to ECMA with driver routine $SRC/odb/cma2odb/matchupdb.F90 being called.

6.5.1 SHUFFLEDB

Usage of the subroutine SHUFFLEDB is as follows:

\[
\text{CALL SHUFFLEDB ( CDNAME_IN, CDNAME_OUT, KALL, KANDAT & & , KANTIM, KNMXUPD )}
\]
6.5  Shuffling observations between two databases

Routine **SHUFFLEDB** is an important driver-routine to perform observational data shuffling (e.g. ECMA to CCMA translation) in concert with routine **XCHANGEDATADB**. The source file resides in `$SRC/odb/cma2odb/shuffledb.F90`. The parameters denote:

- **CDNAME_IN**: CHARACTER(LEN=*), INTENT(IN)
  - Input database name

- **CDNAME_OUT**: CHARACTER(LEN=*), INTENT(IN)
  - Output database name

- **KALL**: INTEGER(4), INTENT(IN)
  - Integer flag to indicate whether to include all observations (=1) or just active ones (=0). The latter one is the usual case when creating CCMA from ECMA

- **KANDAT**: INTEGER(4), INTENT(IN)
  - Analysis date

- **KANTIM**: INTEGER(4), INTENT(IN)
  - Analysis time

- **KNMXUPD**: INTEGER(4), INTENT(IN)
  - Maximum number of updates applicable to this run

6.5.2  **XCHANGEDATADB**

Usage of the subroutine **XCHANGEDATADB** is as follows:

```fortran
CALL XCHANGEDATADB ( CDWHAT, CDTABLE, CDVIEW, CDNAME_IN &
  & , CDNAME_OUT, KHANDLE_IN, KHANDLE_OUT, KALL, KTYPE &
  & , KPOOLS, KPOOLIDS )
```

Routine **XCHANGEDATADB** is an important "slave"-routine, when creating ("shuffling") CCMA-database out of ECMA. It is almost generic, that is to say, it collects all still active observations from the ECMA-database and passes them via observational arrays down to the CCMA-database. The routine is called from **SHUFFLEDB** and its source file resides in `$SRC/odb/cma2odb/xchangedatadb.F90`. The parameters denote:

- **CDWHAT**: CHARACTER(LEN=*), INTENT(IN)
  - The case to be handled
CDTABLE  CHARACTER(LEN=*) , INTENT(IN)  
  Database table in concern

CDVIEW  CHARACTER(LEN=*) , INTENT(IN)  
  Data query to be used

CDNAME_IN  CHARACTER(LEN=*) , INTENT(IN)  
  Input database name (usually ECMA)

CDNAME_OUT  CHARACTER(LEN=*) , INTENT(IN)  
  Output database name (usually CCMA)

KHANDLE_IN  INTEGER(4) , INTENT(IN)  
  Input database handle

KHANDLE_OUT  INTEGER(4) , INTENT(IN)  
  Output database handle

KALL  INTEGER(4) , INTENT(IN)  
  Transform all observations. Normally = 0, but can = 1 when used in creating ECMA-database from ECMASCR

KTYPE  INTEGER(4) , INTENT(IN)  
  Special satellite specific case

KPOOLS  INTEGER(4) , INTENT(IN)  
  Number of (local) pools involved

KPOOLIDS(KPOOLS)  INTEGER(4) , INTENT(IN)  
  Local pool numbers

6.5.3 MATCHUPDB

Usage of the subroutine MATCHUPDB is as follows:

CALL MATCHUPDB ( NUM_TIME_SLOTS, LDFINTRAJ_ECMA & 
  & , LD_INTERNAL_ODB2ECMA, KNMXUPD )

Routine MATCHUPDB is used to feed all the essential information gathered during 4D-Var-
minimization process in CCMA-database back to ECMA-database. The parameters denote:

NUM_TIME_SLOTS  INTEGER(4) , INTENT(IN)  
  Number of timeslots to be matched up
6.5 Shuffling observations between two databases

- **KNMXUPD**: INTEGER(4), INTENT(IN)
  Number of updates applicable

- **LDFINTRAJ_ECMA**: LOGICAL, INTENT(IN)
  Flag for final trajectory

- **LD_INTERNAL_ODB2ECMA**: LOGICAL, INTENT(IN)
  Obsolescent. Used to trigger ECMA-file creation out of ECMA/ODB-database internally i.e. when database was still in memory, and thus saving some crucial computer resources
7 Databases related to IFS

Currently there are two databases which are directly connected for use by IFS (databases ECMA and CCMA) and one indirectly connected (database ECMASC). The database ECMA contains all observations to be used in a 4D-Var analysis. These are the observations converted from external sources like BUFR-files or ASCII text files into ODB. The very first step (traj#0) of the 4D-Var suite screens observations found in ECMA to select a suitable set for the subsequent 4D-Var minimization process. These so called active observations are placed into a separate database called CCMA. Both databases are distributed between MPI-tasks so that every task will process roughly the same number of observations in each timeslot of 4D-Var.

The database ECMASC is meant to be a temporary database, which is normally supposed to have exactly the same data layout as the ECMA. But this similarity can of course be altered to suit local needs. The purpose of this "replica" of ECMA is allow observations to be fed into the ECMASC-database as soon as they arrive and then ECMASC is converted ("shuffled") to ECMA-database to get load balancing of observations between MPI-tasks in subsequent 4D-Var-run. However, this processing is nowadays rarely exercised at ECMWF due to its high cost in terms of processing time, memory and disk requirement. Our (near) load balancing of ECMA is a result of quick examination and timeslotting of BUFR-data before it is decoded into the database.

To add your own database either an automatically recognized (permanent) database or on a temporary basis, see section 9.2 for details.

7.1 Table hierarchies

In order to interpret the data layout entries used in IFS related databases, you need to understand the table hierarchies currently employed. Figures 19 and 20 show these hierarchies in database ECMA which is nearly the same (with less tables) for CCMA. Figure 19 is for all observation types and figure 20 for satellite specific tables. In the satellite data hierarchy all satellite specific tables communicate via an intermediate table sat. This table also contains the INTEGER-equivalent of satellite id, that is satid@sat. Its value should always be the same as ident@hdr for convenience.

Note that not all observation types share satellite table information. When this is the case, the corresponding @LINK-length (see 6.2) is set to zero.

7.2 User defined datatypes

Before introducing database tables and column entries currently used in ECMA and CCMA, we need to describe the associated user defined datatypes also known in ODB/SQL-jargon
as *Bitfield* s. The data layout definition for the IFS code benefits from a number of user defined datatypes. They are explained in the following subsections.

There can be up to 32-bits (assigned from right to left) per user defined datatype. Each member of particular datatype can be referred to in the ODB/SQL as *col_entry.member@table*. 
7.2 User defined datatypes

For instance, the observation characteristics column entry’s obschar member codetype can be referred to as obschar.codetype or with the fully qualified definition obschar.codetype@hdr. All type definitions explained here can be found from file $SRC/odb/ddl/cma.h.

7.2.1 Observation characteristics

Observation characteristics contains the following four member fields:

Table 7: Data members for user defined datatype obschar_t

<table>
<thead>
<tr>
<th>Member name</th>
<th>Bit range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>codetype</td>
<td>0–9</td>
<td>Observation code type</td>
</tr>
<tr>
<td>instype</td>
<td>10–19</td>
<td>Instrument type</td>
</tr>
<tr>
<td>retrtype</td>
<td>20–25</td>
<td>Retrieval type</td>
</tr>
<tr>
<td>datagroup</td>
<td>26–31</td>
<td>Data processing group</td>
</tr>
</tbody>
</table>

And its – now obsolete – type definition was as follows:

```
CREATE TYPE obschar_t AS (  
codetype bit10,  
instype bit10,  
retrtype bit6,  
datagroup bit6
);
```

Note that this datatype has been removed, since we were running out of bits available for the datagroup-member and because it is more convenient to refer to codetype as a separate column entry. Therefore, in the releases of IFS from CY28R2 onwards shows separate column entries in the hdr-table called codetype, instype⁹, retrtype and areatype (not datagroup!) with no obschar-word anymore.

7.2.2 Analysis datum flags

---

⁹Two "t”s, i.e. not instype!
Table 8: Data members for user defined datatype `datum_flag_t`

<table>
<thead>
<tr>
<th>Member name</th>
<th>Bit range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>final</td>
<td>0–3</td>
<td>Final flag</td>
</tr>
<tr>
<td>fg</td>
<td>4–7</td>
<td>First guess flag</td>
</tr>
<tr>
<td>depar</td>
<td>8–11</td>
<td>Departure flag</td>
</tr>
<tr>
<td>varQC</td>
<td>12–15</td>
<td>Variational quality control flag</td>
</tr>
<tr>
<td>blacklist</td>
<td>16–19</td>
<td>Blacklist flag</td>
</tr>
<tr>
<td>ups</td>
<td>20</td>
<td>Metéo-France: d’utilisation par analyse ...</td>
</tr>
<tr>
<td>uvt</td>
<td>21</td>
<td>... de pression de surface</td>
</tr>
<tr>
<td>uhu</td>
<td>22</td>
<td>... de vent et temperature</td>
</tr>
<tr>
<td>ut2</td>
<td>23</td>
<td>... d’utilisation par analyse de temperature a 2m</td>
</tr>
<tr>
<td>uh2</td>
<td>24</td>
<td>... d’humidite a 2m</td>
</tr>
<tr>
<td>uv1</td>
<td>25</td>
<td>... de vent a 10m</td>
</tr>
<tr>
<td>uvr</td>
<td>26</td>
<td>... de precipitations</td>
</tr>
<tr>
<td>usn</td>
<td>27</td>
<td>... de neige</td>
</tr>
<tr>
<td>usst</td>
<td>28</td>
<td>... de temperature de surface de la mer</td>
</tr>
<tr>
<td></td>
<td>29–31</td>
<td>Unused bits</td>
</tr>
</tbody>
</table>

7.2.3 Report/datum status flags

Table 9: Data members for user defined datatype `status_t`

<table>
<thead>
<tr>
<th>Member name</th>
<th>Bit range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>active</td>
<td>0</td>
<td>Active flag</td>
</tr>
<tr>
<td>passive</td>
<td>1</td>
<td>Passive flag</td>
</tr>
<tr>
<td>rejected</td>
<td>2</td>
<td>Rejected flag</td>
</tr>
<tr>
<td>blacklisted</td>
<td>3</td>
<td>Blacklisted flag</td>
</tr>
</tbody>
</table>
| monthly       | 4         | Not for database CCMA: Blacklist failcode bits, fails in ...
| constant      | 5         | ... monthly monitoring part of Blacklist                 |
| experimental  | 6         | ... constant part                                       |
| whitelist     | 7         | Whitelisted (i.e. un-Blacklisted)                       |
|               | 4–31      | Unused for database CCMA                                |
|               | 8–31      | Unused for database ECMA/ECMASCR                       |

Its type definition is implemented as follows:
7.2 User defined datatypes

CREATE TYPE status_t AS (
    active bit1,
    passive bit1,
    rejected bit1,
    blacklisted bit1,
#ifdef ECMA
    // BLACKLIST FAILCODE BITS
    monthly bit1,
    constant bit1,
    experimental bit1,
    whitelist bit1,
#endif /* ECMA */
    /* ECMA */
);

7.2.4 Report RDB-flags

These flags are defined for each report (hdr-table row).

Table 10: Data members for user defined datatype report_rdbflag_t

<table>
<thead>
<tr>
<th>Member name</th>
<th>Bit range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lat_humon</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>lat_QCsub</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>lat_override</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>lat_flag</td>
<td>3–4</td>
<td></td>
</tr>
<tr>
<td>lat_HQC_flag</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>lon_humon</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>lon_QCsub</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>lon_override</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>lon_flag</td>
<td>9–10</td>
<td></td>
</tr>
<tr>
<td>lon_HQC_flag</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>date_humon</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>date_QCsub</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>date_override</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>date_flag</td>
<td>15–16</td>
<td></td>
</tr>
<tr>
<td>date_HQC_flag</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>time_humon</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>time_QCsub</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>time_override</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>time_flag</td>
<td>21–22</td>
<td></td>
</tr>
<tr>
<td>time_HQC_flag</td>
<td>23</td>
<td></td>
</tr>
</tbody>
</table>

continued on next page
Table 10: Datatype report_rdbflag_t ...continued

<table>
<thead>
<tr>
<th>Member name</th>
<th>Bit range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stalt_humon</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>stalt_QCsub</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>stalt_override</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>stalt_flag</td>
<td>27–28</td>
<td></td>
</tr>
<tr>
<td>stalt_HQC_flag</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td></td>
<td>30–31</td>
<td>Unused bits</td>
</tr>
</tbody>
</table>

7.2.5 Datum RDB-flags

These flags are defined for each datum (body-table row).

Table 11: Data members for user defined datatype datum_rdbflag_t

<table>
<thead>
<tr>
<th>Member name</th>
<th>Bit range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>press_humon</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>press_QCsub</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>press_override</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>press_flag</td>
<td>3–4</td>
<td></td>
</tr>
<tr>
<td>press_HQC_flag</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>press_judged_prev_—an</td>
<td>6–7</td>
<td></td>
</tr>
<tr>
<td>press_used_prev_an</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>press_unused_6</td>
<td>9–14</td>
<td>Alignment bits (unused)</td>
</tr>
<tr>
<td>varno_humon</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>varno_QCsub</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>varno_override</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>varno_flag</td>
<td>18–19</td>
<td></td>
</tr>
<tr>
<td>varno_HQC_flag</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>varno_judged_prev_—an</td>
<td>21–22</td>
<td></td>
</tr>
<tr>
<td>varno_used_prev_an</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td></td>
<td>24–31</td>
<td>Unused bits</td>
</tr>
</tbody>
</table>

7.2.6 TEMP/PILOT level flags

These flags are applicable for TEMP/PILOT body-entries only. They are not present in CCMA-database.
## 7.2 User defined datatypes

### Table 12: Data members for user defined datatype level_t

<table>
<thead>
<tr>
<th>Member name</th>
<th>Bit range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>0–8</td>
<td>TEMP/PILOT level identification</td>
</tr>
<tr>
<td>maxwind</td>
<td>9</td>
<td>Maximum wind level</td>
</tr>
<tr>
<td>tropopause</td>
<td>10</td>
<td>Tropopause</td>
</tr>
<tr>
<td>D_part</td>
<td>11</td>
<td>D-part</td>
</tr>
<tr>
<td>C_part</td>
<td>12</td>
<td>C-part</td>
</tr>
<tr>
<td>B_part</td>
<td>13</td>
<td>B-part</td>
</tr>
<tr>
<td>A_part</td>
<td>14</td>
<td>A-part</td>
</tr>
<tr>
<td>surface</td>
<td>15</td>
<td>Surface level</td>
</tr>
<tr>
<td>signwind</td>
<td>16</td>
<td>Significant wind level</td>
</tr>
<tr>
<td>signtemp</td>
<td>17</td>
<td>Significant temperature level</td>
</tr>
<tr>
<td></td>
<td>18–31</td>
<td>Unused bits</td>
</tr>
</tbody>
</table>

### 7.2.7 Report event#1 flags

### Table 13: Data members for user defined datatype report_event1_t

<table>
<thead>
<tr>
<th>Member name</th>
<th>Bit range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>no_data</td>
<td>0</td>
<td>No data available</td>
</tr>
<tr>
<td>all_rejected</td>
<td>1</td>
<td>All rejected</td>
</tr>
<tr>
<td>bad_practice</td>
<td>2</td>
<td>Bad reporting practice</td>
</tr>
<tr>
<td>rdb_rejected</td>
<td>3</td>
<td>Rejected in RDB</td>
</tr>
<tr>
<td>rdb_activated</td>
<td>4</td>
<td>Activated in RDB</td>
</tr>
<tr>
<td>whitelist_activated</td>
<td>5</td>
<td>Activated via whitelist</td>
</tr>
<tr>
<td>horipos_outrange</td>
<td>6</td>
<td>Horizontal position out of range</td>
</tr>
<tr>
<td>vertpos_outrange</td>
<td>7</td>
<td>Vertical position out of range</td>
</tr>
<tr>
<td>time_outrange</td>
<td>8</td>
<td>Time out of range</td>
</tr>
<tr>
<td>redundant</td>
<td>9</td>
<td>Redundant report</td>
</tr>
<tr>
<td>land</td>
<td>10</td>
<td>Land observation</td>
</tr>
<tr>
<td>sea</td>
<td>11</td>
<td>Sea observation</td>
</tr>
<tr>
<td>stalt_missing</td>
<td>12</td>
<td>Station altitude</td>
</tr>
<tr>
<td>modsurf_stalt_distance</td>
<td>13</td>
<td>Surface modified due to stalt</td>
</tr>
<tr>
<td>namelist_rejected</td>
<td>14</td>
<td>Rejected via NAMELIST</td>
</tr>
<tr>
<td>QC_failed</td>
<td>15</td>
<td>Quality control failure</td>
</tr>
<tr>
<td></td>
<td>16–31</td>
<td>Unused bits</td>
</tr>
</tbody>
</table>
7.2.8 Report event#2 flags

These flags are not generic, but differ from one observation type to another. Therefore it is easier to alias it to 32-bit (packed) integer \texttt{pk1int} as follows:

\begin{verbatim}
CREATE TYPE report_event2_t = pk1int;
\end{verbatim}

7.2.9 Report blacklist flags

When set, these bits indicate which column entry/entries in table \texttt{hdr} (or related, like \texttt{atovs}) were responsible for causing blacklisting to occur. So the last column in the next table should be read as "Blacklisting caused by" followed by the text in the column.

In practice these column entries are present in the IF-conditions of the Blacklist language definition file that resulted blacklisting of current report.

Table 14: Data members for user defined datatype \texttt{report\_blacklist_t}

<table>
<thead>
<tr>
<th>Member name</th>
<th>Bit range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obstype</td>
<td>0</td>
<td>Observation type</td>
</tr>
<tr>
<td>statid</td>
<td>1</td>
<td>Station identifier</td>
</tr>
<tr>
<td>codetype</td>
<td>2</td>
<td>CMA codetype</td>
</tr>
<tr>
<td>instype</td>
<td>3</td>
<td>Instrument type</td>
</tr>
<tr>
<td>date</td>
<td>4</td>
<td>Measurement date</td>
</tr>
<tr>
<td>time</td>
<td>5</td>
<td>Measurement time</td>
</tr>
<tr>
<td>lat</td>
<td>6</td>
<td>Latitude</td>
</tr>
<tr>
<td>lon</td>
<td>7</td>
<td>Longitude</td>
</tr>
<tr>
<td>stalt</td>
<td>8</td>
<td>Station altitude</td>
</tr>
<tr>
<td>scanpos</td>
<td>9</td>
<td>Scan position</td>
</tr>
<tr>
<td>retrtype</td>
<td>10</td>
<td>Retrieval type</td>
</tr>
<tr>
<td>QI_1</td>
<td>11</td>
<td>Quality indicator 1 (satob)</td>
</tr>
<tr>
<td>QI_2</td>
<td>12</td>
<td>Quality indicator 2 (satob)</td>
</tr>
<tr>
<td>QI_3</td>
<td>13</td>
<td>Quality indicator 3 (satob)</td>
</tr>
<tr>
<td>modoro</td>
<td>14</td>
<td>Model orography</td>
</tr>
<tr>
<td>lsmask</td>
<td>15</td>
<td>Land sea mask (either 0 or 1)</td>
</tr>
<tr>
<td>rlsmask</td>
<td>16</td>
<td>Land sea mask (real; 0.0 .. 1.0)</td>
</tr>
<tr>
<td>modPS</td>
<td>17</td>
<td>Model pressure</td>
</tr>
<tr>
<td>modTS</td>
<td>18</td>
<td>Model surface temperature</td>
</tr>
<tr>
<td>modT2M</td>
<td>19</td>
<td>Model 2 metre temperature</td>
</tr>
<tr>
<td>modtop</td>
<td>20</td>
<td>Model top pressure</td>
</tr>
<tr>
<td>sensor</td>
<td>21</td>
<td>Sensor id</td>
</tr>
<tr>
<td>fov</td>
<td>22</td>
<td>Field of view</td>
</tr>
<tr>
<td>satza</td>
<td>23</td>
<td>Satellite zenith angle</td>
</tr>
</tbody>
</table>

continued on next page …
7.2 User defined datatypes

Table 14: Datatype report_blacklist_t …continued

<table>
<thead>
<tr>
<th>Member name</th>
<th>Bit range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>andate</td>
<td>24</td>
<td>Analysis date</td>
</tr>
<tr>
<td>antime</td>
<td>25</td>
<td>Analysis time</td>
</tr>
<tr>
<td></td>
<td>26–31</td>
<td>Unused bits</td>
</tr>
</tbody>
</table>

7.2.10 Datum blacklist flags

When set, these bits indicate which column entry/entries in table body were responsible for causing blacklisting to occur. So the last column in the next table should be read as "Blacklisting caused by" followed by the text in the column. In practice these column entries are present in the IF-condition(s) of the Blacklist definition file resulting blacklisting of current datum.

Table 15: Data members for user defined datatype datum_blacklist_t

<table>
<thead>
<tr>
<th>Member name</th>
<th>Bit range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>varno</td>
<td>0</td>
<td>Variable number</td>
</tr>
<tr>
<td>vertco_type</td>
<td>1</td>
<td>Vertical coordinate type</td>
</tr>
<tr>
<td>press</td>
<td>2</td>
<td>Pressure level/channel number</td>
</tr>
<tr>
<td>press_rl</td>
<td>3</td>
<td>Reference pressure level</td>
</tr>
<tr>
<td>prcode</td>
<td>4</td>
<td>SYNOP pressure code</td>
</tr>
<tr>
<td>obsvalue</td>
<td>5</td>
<td>Observed value</td>
</tr>
<tr>
<td>fg_depar</td>
<td>6</td>
<td>First guess departure</td>
</tr>
<tr>
<td>obs_error</td>
<td>7</td>
<td>Observation error</td>
</tr>
<tr>
<td>fg_error</td>
<td>8</td>
<td>First guess error</td>
</tr>
<tr>
<td>winchan_dep</td>
<td>9</td>
<td>Window channel departure</td>
</tr>
<tr>
<td>obs_t</td>
<td>10</td>
<td>??</td>
</tr>
<tr>
<td></td>
<td>11–31</td>
<td>Unused bits</td>
</tr>
</tbody>
</table>

7.2.11 Variable presence flags

This flag can be used to indicate (roughly) what kind of variable numbers (varno) are present in the body-table belonging to this observation spot. Note, that this entry is relatively new and may not always be correctly filled. When matured, it should speed up SQL-queries requiring body-entries, since scan of body-related tables can be reduced to bare minimum by requesting these hdr-flags in the SQL-query.
The last column indicates presence of (any) $\text{varno@body}$ belonging to particular category.

### Table 16: Data members for user defined datatype $\text{varno_-presence_t}$

<table>
<thead>
<tr>
<th>Member name</th>
<th>Bit range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>other</td>
<td>0</td>
<td>For non-categorized variable number</td>
</tr>
<tr>
<td>ps</td>
<td>1</td>
<td>Surface pressure</td>
</tr>
<tr>
<td>wind</td>
<td>2</td>
<td>Wind-components</td>
</tr>
<tr>
<td>t</td>
<td>3</td>
<td>Temperature</td>
</tr>
<tr>
<td>z</td>
<td>4</td>
<td>Geopotential</td>
</tr>
<tr>
<td>rh</td>
<td>5</td>
<td>Relative humidity</td>
</tr>
<tr>
<td>q</td>
<td>6</td>
<td>Specific humidity</td>
</tr>
<tr>
<td>obs2m</td>
<td>7</td>
<td>2 meter observations</td>
</tr>
<tr>
<td>snow</td>
<td>8</td>
<td>Snow analysis related observations</td>
</tr>
<tr>
<td>rain</td>
<td>9</td>
<td>Rain</td>
</tr>
<tr>
<td>rawbt</td>
<td>10</td>
<td>Brightness temperature</td>
</tr>
<tr>
<td>rawra</td>
<td>11</td>
<td>Raw radiance</td>
</tr>
<tr>
<td>ozone</td>
<td>12</td>
<td>Ozone</td>
</tr>
<tr>
<td>dz</td>
<td>13</td>
<td>Thickness</td>
</tr>
<tr>
<td>pwc</td>
<td>14</td>
<td>PWC</td>
</tr>
<tr>
<td>time_period</td>
<td>15</td>
<td>Time period (of rain)</td>
</tr>
<tr>
<td></td>
<td>16–31</td>
<td>Unused bits</td>
</tr>
</tbody>
</table>

### 7.2.12 Datum event#1 flags

These are body-table related events and they describe reason why certain body-enty has been rejected – or even activated.

### Table 17: Data members for user defined datatype $\text{datum_event1_t}$

<table>
<thead>
<tr>
<th>Member name</th>
<th>Bit range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vertco_missing</td>
<td>0</td>
<td>Vertical coordinate missing</td>
</tr>
<tr>
<td>obsvalue_missing</td>
<td>1</td>
<td>Observed value missing</td>
</tr>
<tr>
<td>fg_missing</td>
<td>2</td>
<td>First guess value missing</td>
</tr>
<tr>
<td>rdb_rejected</td>
<td>3</td>
<td>Rejected due to $\text{rdbflags}$</td>
</tr>
<tr>
<td>rdb_activated</td>
<td>4</td>
<td>Activated due to $\text{rdbflags}$</td>
</tr>
<tr>
<td>whitelist_activated</td>
<td>5</td>
<td>Activated due to whitelisting</td>
</tr>
<tr>
<td>bad_practice</td>
<td>6</td>
<td>Bad reporting practice</td>
</tr>
<tr>
<td>vertpos_outrange</td>
<td>7</td>
<td>Vertical position out of range</td>
</tr>
<tr>
<td>reflevel_outrange</td>
<td>8</td>
<td>Reference level out of range</td>
</tr>
<tr>
<td>fg2big</td>
<td>9</td>
<td>First guess too big</td>
</tr>
<tr>
<td>depar2big</td>
<td>10</td>
<td>Analysis departure too big</td>
</tr>
</tbody>
</table>

continued on next page…
Table 17: Datatype datum_event1_t ...continued

<table>
<thead>
<tr>
<th>Member name</th>
<th>Bit range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obs_error2big</td>
<td>11</td>
<td>Observation error too big</td>
</tr>
<tr>
<td>datum_redundant</td>
<td>12</td>
<td>Redundant body entry</td>
</tr>
<tr>
<td>level_redundant</td>
<td>13</td>
<td>Redundant level information</td>
</tr>
<tr>
<td>land</td>
<td>14</td>
<td>Land point</td>
</tr>
<tr>
<td>sea</td>
<td>15</td>
<td>Sea point</td>
</tr>
<tr>
<td>not_analysis_varno</td>
<td>16</td>
<td>Not an analysis variable</td>
</tr>
<tr>
<td>duplicate</td>
<td>17</td>
<td>Duplicate body entry</td>
</tr>
<tr>
<td>levels2many</td>
<td>18</td>
<td>Too many levels</td>
</tr>
<tr>
<td>multilevel_check</td>
<td>19</td>
<td>Multilevel check failed</td>
</tr>
<tr>
<td>level_selection_bit1</td>
<td>20</td>
<td>Level selection ??</td>
</tr>
<tr>
<td>vertco_consistency</td>
<td>21</td>
<td>Vertical consistency failed ??</td>
</tr>
<tr>
<td>vertco_type_changed</td>
<td>22</td>
<td>Vertical coordinate type has changed</td>
</tr>
<tr>
<td>namelist_rejected</td>
<td>23</td>
<td>Rejection via NAMELIST</td>
</tr>
<tr>
<td>combined_flagging</td>
<td>24</td>
<td>Combined flagging</td>
</tr>
<tr>
<td>report_rejected</td>
<td>25</td>
<td>Whole report is rejected</td>
</tr>
<tr>
<td>varQC_performed</td>
<td>26</td>
<td>Variational quality control has been performed</td>
</tr>
<tr>
<td>contam_cld_flag</td>
<td>27</td>
<td>Cloud contamination</td>
</tr>
<tr>
<td>contam_rain_flag</td>
<td>28</td>
<td>Rain contamination</td>
</tr>
<tr>
<td></td>
<td>29–31</td>
<td>Unused bits</td>
</tr>
</tbody>
</table>

7.2.13 Datum event#2 flags

These flags are not generic, but differ from observation type to another. Therefore it is easier to alias it to 32-bit (packed) integer pk1int as follows:

```
CREATE TYPE datum_event2_t = pk1int;
```

7.3 Database ECMA

Database ECMA is the backbone for IFS interfacing with ODB. It contains all observations from (possibly pre-thinned) BUFR-file (or external ASCII text file). The following subsections list all column entries for each table with their description and corresponding MDB/MLNK-pointers, which are used to reference these columns in IFS. The definitive order of the column entries is not necessarily the same as in data definition layout file. Please consult the $SRC/odb/ddl.ECMA/ECMA.ddl and all files it includes.
7.3.1 Table desc

This table holds descriptive data for each database. With one data row definition for each pool. By examining its columns, you can find out whether the database has been used operationally (`expver` equals to right justified ‘0001’), what is the analysis timestamp concerned, when the database was created or last modified, who was the creator and modifier. Furthermore, one can find how many trajectory updates are expected from this database (i.e. how many update-tables are containing non-missing data) and whether positional coordinates (`lat, lon`) are already in radians or still in degrees. In addition there are links to a few tables below the table hierarchy.

Table 18: Database table desc

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>expver</td>
<td>string</td>
<td>Experiment version</td>
<td>MDB_EXPVER_AT_DESC</td>
</tr>
<tr>
<td>andate</td>
<td>YYYYMMDD</td>
<td>Analysis date</td>
<td>MDB_ANDDATE_AT_DESC</td>
</tr>
<tr>
<td>antime</td>
<td>HHMMSS</td>
<td>Analysis time</td>
<td>MDB_ANTIME_AT_DESC</td>
</tr>
<tr>
<td>creddate</td>
<td>YYYYMMDD</td>
<td>DB creation date</td>
<td>MDB_CREADATE_AT_DESC</td>
</tr>
<tr>
<td>creatime</td>
<td>HHMMSS</td>
<td>DB creation time</td>
<td>MDB_CREATETIME_AT_DESC</td>
</tr>
<tr>
<td>creaby</td>
<td>string</td>
<td>User who created database</td>
<td>MDB_CREABY_AT_DESC</td>
</tr>
<tr>
<td>moddate</td>
<td>YYYYMMDD</td>
<td>DB modification date</td>
<td>MDB_MODDATE_AT_DESC</td>
</tr>
<tr>
<td>modtime</td>
<td>HHMMSS</td>
<td>DB modification time</td>
<td>MDB_MODTIME_AT_DESC</td>
</tr>
<tr>
<td>modby</td>
<td>string</td>
<td>User who modified database</td>
<td>MDB_MODBY_AT_DESC</td>
</tr>
<tr>
<td>mxup_traj</td>
<td>pk1int</td>
<td>Max. no. of trajectory updates</td>
<td>MDB_MXUP_TRAJ_AT_DESC</td>
</tr>
<tr>
<td>latlon_rad</td>
<td>pk1int</td>
<td>Flag for (lat, lon) being in radians</td>
<td>MDB_LATLON_RAD_AT_DESC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddrno</td>
<td>@LINK</td>
<td>Offsets &amp; lengths to ...</td>
<td>MLNK_DESC2DDRS</td>
</tr>
<tr>
<td>poolmask</td>
<td>@LINK</td>
<td>poolmask</td>
<td>MLNK_DESC2POOLMASK</td>
</tr>
<tr>
<td>timeslot_index</td>
<td>@LINK</td>
<td>timeslot_index</td>
<td>MLNK_DESC2TIMESLOT_INDEX</td>
</tr>
</tbody>
</table>

7.3.2 Table ddrs

This table contains Data Description Records for analysis and is still in a little-bit old fashioned format: data needs to be accessed nearly as a bulk data, rather than referring to column entries with their natural names.

Table 19: Database table ddrs

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddrno</td>
<td>pk1int</td>
<td>CMA DDR-block number</td>
<td>MDB_DDRNO_AT_DDRS</td>
</tr>
</tbody>
</table>

continued on next page …
7.3 Database ECMA

### Table 19: Database table ddr

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordno</td>
<td>pk1int</td>
<td>Word number within DDR-block</td>
<td>MDB_WORDNO_AT_DDRS</td>
</tr>
<tr>
<td>bulkdata</td>
<td>pk9real</td>
<td>DDR-data as in former CMA-files</td>
<td>MDB_BULKDATA_AT_DDRS</td>
</tr>
</tbody>
</table>

#### 7.3.3 Table poolmask

This table contains data distribution information for use by poolmasked data access. This information is useful, since you can also find how particular observational groups are split between data pools, or how much data a particular timeslot has in terms of location or measurement information.

### Table 20: Database table poolmask

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tslot</td>
<td>pk1int</td>
<td>Timeslot number</td>
<td>MDB_TSLOT_AT_POOLMASK</td>
</tr>
<tr>
<td>obstype</td>
<td>pk1int</td>
<td>observation type (CMA)</td>
<td>MDB_OBSTYPE_AT_POOLMASK</td>
</tr>
<tr>
<td>codetype</td>
<td>pk1int</td>
<td>codetype (CMA)</td>
<td>MDB_CODETYPE_AT_POOLMASK</td>
</tr>
<tr>
<td>sensor</td>
<td>pk1int</td>
<td>Satellite sensor</td>
<td>MDB_SENSOR_AT_POOLMASK</td>
</tr>
<tr>
<td>bufrtype</td>
<td>pk1int</td>
<td>BUFR type</td>
<td>MDB_BURFRTYPO_AT_POOLMASK</td>
</tr>
<tr>
<td>subtype</td>
<td>pk1int</td>
<td>BUFR subtype</td>
<td>MDB_SUBTYPO_AT_POOLMASK</td>
</tr>
<tr>
<td>satinst</td>
<td>pk1int</td>
<td>WMO satellite instrument</td>
<td>MDB_SATINST_AT_POOLMASK</td>
</tr>
<tr>
<td>poolno</td>
<td>pk1int</td>
<td>Pool number</td>
<td>MDB_POOLNO_AT_POOLMASK</td>
</tr>
<tr>
<td>hdr_count</td>
<td>pk1int</td>
<td>Count of spots/pixels</td>
<td>MDB_HDR_COUNT_AT_POOLMASK</td>
</tr>
<tr>
<td>body_count</td>
<td>pk1int</td>
<td>Count of body entries</td>
<td>MDB_BODY_COUNT_AT_POOLMASK</td>
</tr>
<tr>
<td>max_bodylen</td>
<td>pk1int</td>
<td>Max. no. of bodies</td>
<td>MDB_MAX_BODYLEN_AT_POOLMASK</td>
</tr>
</tbody>
</table>

The count of spots/pixels refers to the number of different observation locations both in space (lat, lon) and measurement time (date, time). The count of body entries tells you the number of individual measurements in terms of (say) wind component, temperatures, humidities etc.
### 7.3.4 Table timeslot_index

This table enables direct access to timeslot-based data without need to scan the whole index or hdr tables for that purpose. Optimally this table should contain only as many rows per pool as there are observational timeslots specified for the particular 4D-Var-run.

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeslot</td>
<td>pk1int</td>
<td>Timeslot number</td>
<td>MDB_TIMESLOT_AT_TIMESLOT_INDEX</td>
</tr>
<tr>
<td>index</td>
<td>@LINK</td>
<td>Offset&amp;length to index</td>
<td>MLNK_TIMESLOT_INDEX2INDEX</td>
</tr>
</tbody>
</table>

In order to make any use of this table (and speeding up SQLs), its child table (index) must already have been sorted with respect to its column timeslot@index.

### 7.3.5 Table index

This table controls timeslot information and enables shuffle and matchup procedures to work correctly. This is to say, the table is used for complicated parallel data shuffles between MPI-tasks when feeding information back and forth between ECMA and CCMA databases. Another purpose is to keep track of kset’s and pointers to IFS EC- and GOM-arrays.

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>pk1int</td>
<td>Report &quot;owner&quot; pool in ECMA</td>
<td>MDB_TARGET_AT_INDEX</td>
</tr>
<tr>
<td>procid</td>
<td>pk1int</td>
<td>Destination pool for CCMA</td>
<td>MDB_PROCID_AT_INDEX</td>
</tr>
<tr>
<td>tslot</td>
<td>pk1int</td>
<td>Timeslot number</td>
<td>MDB_TSlot_AT_INDEX</td>
</tr>
<tr>
<td>kset</td>
<td>pk1int</td>
<td>kset as from ECSET</td>
<td>MDB_KSET_AT_INDEX</td>
</tr>
<tr>
<td>abnob</td>
<td>pk1int</td>
<td>ECxxx-array pointer</td>
<td>MDB_ABNOB_AT_INDEX</td>
</tr>
<tr>
<td>mapomm</td>
<td>pk1int</td>
<td>GOMxxx-array pointer</td>
<td>MDB_MAPOMM_AT_INDEX</td>
</tr>
<tr>
<td>hdr</td>
<td>@LINK</td>
<td>Offset&amp;length to hdr</td>
<td>MLNK_INDEX2HDR</td>
</tr>
</tbody>
</table>

As explained earlier in 7.3.4, this table needs to be sorted with respect to its column timeslot@index to be efficient.
This table contains location and time information for every observation spot/pixel. It is also known as report data and has links down the line to its body-related tables and the satellite specific intermediate-table sat. Note also that if the observation type is not satellite data, then the @LINK-length to sat-table is zero.

Another important point is to note that positional coordinates (lat, lon) are normally expressed in radians, unless you examine database just after creating it from BUFR-file, in which case they are in degrees. The value of column entry latlon_rad@desc shows the conversion status.

Table 23: Database table hdr

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>seqno</td>
<td>pk1int</td>
<td>Observation sequence number</td>
<td>MDBONM</td>
</tr>
<tr>
<td>obstype</td>
<td>pk1int</td>
<td>(CMA) Observation type</td>
<td>MDBOTP</td>
</tr>
<tr>
<td>codetype</td>
<td>pk1int</td>
<td>Observation (CMA) code type</td>
<td>MDB_CODETYPE_AT_HDR</td>
</tr>
<tr>
<td>insttype</td>
<td>pk1int</td>
<td>Observation instrument type</td>
<td>MDB_INSTTYPE_AT_HDR</td>
</tr>
<tr>
<td>retrtype</td>
<td>pk1int</td>
<td>Observation retrieval type</td>
<td>MDB_RETRTYPE_AT_HDR</td>
</tr>
<tr>
<td>areatype</td>
<td>pk1int</td>
<td>Observation area type</td>
<td>MDB_AREATYPE_AT_HDR</td>
</tr>
<tr>
<td>zone</td>
<td>pk9int</td>
<td>??? check MF ??</td>
<td>MDB_ZONE_AT_HDR</td>
</tr>
<tr>
<td>date</td>
<td>YYYYMMDD</td>
<td>Observation date</td>
<td>MDBDAT</td>
</tr>
<tr>
<td>time</td>
<td>HHMMSS</td>
<td>Exact observation time</td>
<td>MDBETM</td>
</tr>
<tr>
<td>status</td>
<td>status_t</td>
<td>Report’s status</td>
<td>MDBRST</td>
</tr>
<tr>
<td>event1</td>
<td>report_event1_t</td>
<td>Report’s events (part 1), see ??</td>
<td>MDBREV1</td>
</tr>
<tr>
<td>sortbox</td>
<td>pk1int</td>
<td>Sorting box</td>
<td>MDBBOX</td>
</tr>
<tr>
<td>site dep</td>
<td>pk1int</td>
<td>Site dependent integer</td>
<td>MDBSTD</td>
</tr>
<tr>
<td>source</td>
<td>string</td>
<td>Source id of obs. (re-analysis)</td>
<td>MDB_SOURCE_AT_HDR</td>
</tr>
<tr>
<td>statid</td>
<td>string</td>
<td>Station id in character form</td>
<td>MDBSID</td>
</tr>
<tr>
<td>ident</td>
<td>pk1int</td>
<td>Numeric statid or zero</td>
<td>MDB_IDENT_AT_HDR</td>
</tr>
<tr>
<td>lat</td>
<td>pk9real</td>
<td>Latitude (normally in radians)</td>
<td>MDBLAT</td>
</tr>
<tr>
<td>lon</td>
<td>pk9real</td>
<td>Longitude (normally in radians)</td>
<td>MDBLON</td>
</tr>
<tr>
<td>stalt</td>
<td>pk9real</td>
<td>Station altitude</td>
<td>MDBALT</td>
</tr>
<tr>
<td>modoro</td>
<td>pk9real</td>
<td>Model’s orography</td>
<td>MDBMOR</td>
</tr>
<tr>
<td>trlat</td>
<td>pk9real</td>
<td>Transformed lat</td>
<td>MDBTLA</td>
</tr>
<tr>
<td>trlon</td>
<td>pk9real</td>
<td>Transformed lon</td>
<td>MDBTLO</td>
</tr>
<tr>
<td>insspec</td>
<td>instrm_t</td>
<td>Inst. spec. word positions</td>
<td>MDBINS</td>
</tr>
<tr>
<td>anemoht</td>
<td>pk9real</td>
<td>Height of anemometer</td>
<td>MDBHOAN{S</td>
</tr>
<tr>
<td>baroht</td>
<td>pk9real</td>
<td>Height of barometer</td>
<td>MDBHOBA{S</td>
</tr>
<tr>
<td>sensor</td>
<td>pk1int</td>
<td>Satellite sensor indicator</td>
<td>MDBSSIA</td>
</tr>
<tr>
<td>numlev</td>
<td>pk1int</td>
<td>No. of distinct pressure levels</td>
<td>MDB_NUMLEV_AT_HDR</td>
</tr>
<tr>
<td>numactiveb</td>
<td>pk1int</td>
<td>No. of active body entries</td>
<td>MDB_NUMACTIVEB_AT_HDR</td>
</tr>
</tbody>
</table>

continued on next page...
Table 23: Database table hdr continued

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>checksum</td>
<td>pk9real</td>
<td>Sum of obsvalue's in body-table that are not missing data</td>
<td>MDB_CHECKSUM_AT_HDR</td>
</tr>
<tr>
<td>varno_presence</td>
<td>varno_presence_t</td>
<td>Variable presence flag</td>
<td>MDB_VARNO_PRESENCE_AT_HDR</td>
</tr>
<tr>
<td>rdbflag</td>
<td>report_rdbflag_t</td>
<td>Report’s flags</td>
<td>MDBRFL</td>
</tr>
<tr>
<td>subtype</td>
<td>pk1int</td>
<td>BUFR-subtype</td>
<td>MDB_SUBTYPE_AT_HDR</td>
</tr>
<tr>
<td>bufrtype</td>
<td>pk1int</td>
<td>BUFR-type</td>
<td>MDB_BUFRTYPE_AT_HDR</td>
</tr>
<tr>
<td>satinst</td>
<td>pk1int</td>
<td>Satellite instrument in BUFR/WMO</td>
<td>MDB_SATINST_AT_HDR</td>
</tr>
<tr>
<td>satname[2]</td>
<td>string</td>
<td>Extended satellite name (i.e. two word, up to 16 bytes)</td>
<td>MDB_SATNAME_AT_HDR(2)</td>
</tr>
<tr>
<td>sat</td>
<td>@LINK</td>
<td>Link to sat</td>
<td></td>
</tr>
<tr>
<td>body</td>
<td>@LINK</td>
<td>Link to body</td>
<td></td>
</tr>
<tr>
<td>errstat</td>
<td>@LINK</td>
<td>Link to errstat</td>
<td></td>
</tr>
<tr>
<td>update[:]</td>
<td>@LINK</td>
<td>Link to $NMXUPD update-tables</td>
<td></td>
</tr>
<tr>
<td>thinningkey[$NUMTHBOX]</td>
<td>pk9real</td>
<td>Thinning key</td>
<td>MDB_THINNINGKEY_AT_HDR(;)</td>
</tr>
<tr>
<td>thinningtimekey</td>
<td>report_black-list_t</td>
<td>Thinning time key</td>
<td>MDB_THINNINGTIMEKEY_AT_HDR</td>
</tr>
<tr>
<td>blacklist</td>
<td>@LINK</td>
<td>Report’s blacklist events</td>
<td>MDBRBLE</td>
</tr>
<tr>
<td>event2</td>
<td>@LINK</td>
<td>Report events (part 2)</td>
<td>MDBREV2</td>
</tr>
<tr>
<td>bufr</td>
<td>@LINK</td>
<td>Link to BUFR-data</td>
<td>MDB_SUBSETNO_AT_HDR</td>
</tr>
<tr>
<td>subsetno</td>
<td>pk9int</td>
<td>BUFR-multisubset no. (=0 for single)</td>
<td>MDBRBLB</td>
</tr>
<tr>
<td>station_type</td>
<td>pk1int</td>
<td>SYNOP/SHIPs (whether DRIBU)</td>
<td>MDB_STATION_TYPE_AT_HDR</td>
</tr>
<tr>
<td>sonde_type</td>
<td>pk1int</td>
<td>For TEMP bias correction</td>
<td>MDB_SONDE_TYPE_AT_HDR</td>
</tr>
<tr>
<td>satem_instdata</td>
<td>pk1int</td>
<td>Instrument data used in processing old-SATEM</td>
<td>MDB_SATEM_INSTDATA_AT_HDR</td>
</tr>
<tr>
<td>satem_dataproc</td>
<td>pk1int</td>
<td>Data processing technique used for old-SATEM</td>
<td>MDB_SATEM_DATAPROC_AT_HDR</td>
</tr>
</tbody>
</table>

Note, that its data rows can appear in any order, since its parent table (index) (and grandparent (timeslot_index) have arranged @LINK-offsets in such a way that timeslot access is will be direct.

7.3.7 Table body

This table contains the actual measurement information accompanied with first guess and analysis departure values (once calculated) and analysis status flag information.
### Table 24: Database table body

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>varno</td>
<td>pk1int</td>
<td>Variable number</td>
<td>MDBVNM</td>
</tr>
<tr>
<td>vertco_type</td>
<td>pk1int</td>
<td>Vertical coordinate type</td>
<td>MDBVCO</td>
</tr>
<tr>
<td>anflag</td>
<td>datum_flag_t</td>
<td>Observation flags, see 7.2.2</td>
<td>MDBFLG</td>
</tr>
<tr>
<td>status</td>
<td>status_t</td>
<td>Observation status, see 7.2.3</td>
<td>MDBDSTA</td>
</tr>
<tr>
<td>event1</td>
<td>datum_event1_t</td>
<td>Observation events (part 1), see ??</td>
<td>MDBDEV1</td>
</tr>
<tr>
<td>entryno</td>
<td>pk1int</td>
<td>Entry sequence number</td>
<td>MDBESQN</td>
</tr>
<tr>
<td>press</td>
<td>pk9real</td>
<td>Vert. coord. (pressure, channel no.)</td>
<td>MDBPPP</td>
</tr>
<tr>
<td>press_r1</td>
<td>pk9real</td>
<td>Vert. coord. reference level</td>
<td>MDBPRL</td>
</tr>
<tr>
<td>obsvalue</td>
<td>pk9real</td>
<td>Observed value of variable</td>
<td>MDBVAR</td>
</tr>
<tr>
<td>aux[$NUMAUX]</td>
<td>pk9real</td>
<td>Auxillary observed values. For u-wind component aux[1] contains wind speed (ff) and aux[2] contains wind direction (dd). For v-wind component values are just reversed.</td>
<td>MDB_AUX_AT_BODY(:)</td>
</tr>
<tr>
<td>biascorr</td>
<td>pk9real</td>
<td>Bias correction</td>
<td>MDB[TORB</td>
</tr>
<tr>
<td>rdbflag</td>
<td>datum_flag_t</td>
<td>Observation RDB-flags, see ??</td>
<td>MDBRDFL</td>
</tr>
<tr>
<td>an_depar</td>
<td>pk9real</td>
<td>Observed minus Analysed value</td>
<td>MDBOMN</td>
</tr>
<tr>
<td>fg_depar</td>
<td>pk9real</td>
<td>Observed minus First guess value</td>
<td>MDBOMF</td>
</tr>
<tr>
<td>fg_check_1</td>
<td>pk9real</td>
<td>First guess check no. 1</td>
<td>MDBFGC1</td>
</tr>
<tr>
<td>fg_check_2</td>
<td>pk9real</td>
<td>First guess check no. 2</td>
<td>MDBFGC2</td>
</tr>
<tr>
<td>cld_fg_depar</td>
<td>pk9real</td>
<td>FG departure for cloudy radiances</td>
<td>MDB_CLD_FG_DEPAR_AT_BODY</td>
</tr>
<tr>
<td>surfemiss</td>
<td>pk9real</td>
<td>Surface emissivity</td>
<td>MDB_SURFEMISS_AT_BODY</td>
</tr>
<tr>
<td>csr_pclear</td>
<td>pk9real</td>
<td>% of clear pixels in mean CSR</td>
<td>MDB_CSR_PCLEAR_AT_BODY</td>
</tr>
<tr>
<td>csr_pcloudy</td>
<td>pk9real</td>
<td>% of cloudy pixels in mean CSR</td>
<td>MDB_CSR_PCLOUDY_AT_BODY</td>
</tr>
<tr>
<td>rank_cld</td>
<td>pk9real</td>
<td>Channel ranking for cloud detection</td>
<td>MDB_RANK_CLD_AT_BODY</td>
</tr>
<tr>
<td>rank_an</td>
<td>pk9real</td>
<td>Channel ranking for analysis</td>
<td>MDB_RANK_AN_AT_BODY</td>
</tr>
<tr>
<td>mf_vertco_type</td>
<td>pk1int</td>
<td>Meteo-France/CANARI-only vertical coordinate type</td>
<td>MDRBRBC</td>
</tr>
<tr>
<td>mf_log_p</td>
<td>pk9real</td>
<td>Pressure used in CANARI (log(p))</td>
<td>MDRBRBPO</td>
</tr>
<tr>
<td>mf_stddev</td>
<td>pk9real</td>
<td>Obs. std dev at bottom layer</td>
<td>MDRBROE</td>
</tr>
<tr>
<td>blacklist</td>
<td>datum_blacklist_t</td>
<td>Not in database CCMA</td>
<td>MDBDBLE</td>
</tr>
<tr>
<td>event2</td>
<td>datum_event2_t</td>
<td>Datum events (part 2)</td>
<td>MDBDEV2</td>
</tr>
</tbody>
</table>

*continued on next page...*
Table 24: Database table body ... continued

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ppcode</td>
<td>pk1int</td>
<td>Synop pressure code</td>
<td>MDBSYPC</td>
</tr>
<tr>
<td>level</td>
<td>level_t</td>
<td>Pilot level id</td>
<td>MDB{TELID</td>
</tr>
</tbody>
</table>

7.3.8 Table errstat

This table contains observation error statistics.

Table 25: Database table errstat

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_obs_error</td>
<td>pk9real</td>
<td>Final observation error</td>
<td>MDBFOE</td>
</tr>
<tr>
<td>obs_error</td>
<td>pk9real</td>
<td>Observation error</td>
<td>MDBOER</td>
</tr>
<tr>
<td>repres_error</td>
<td>pk9real</td>
<td>Representativeness error</td>
<td>MDBRER</td>
</tr>
<tr>
<td>pers_error</td>
<td>pk9real</td>
<td>Persistence error</td>
<td>MDBPER</td>
</tr>
<tr>
<td>lg_error</td>
<td>pk9real</td>
<td>First guess error</td>
<td>MDBFGE</td>
</tr>
</tbody>
</table>

7.3.9 The update-tables

There are a total of $NMXUPD$ update-tables, one table for each 4D-Var-update. They all have identical column entries. These intermediate departures for each update $IUPD \leq MXUP_\text{-TRAJ}$ are:

Table 26: Database table update

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial</td>
<td>pk9real</td>
<td>obs. minus initial value</td>
<td>MDBIOM0(IUPD)</td>
</tr>
<tr>
<td>hires</td>
<td>pk9real</td>
<td>obs. minus high res. value</td>
<td>MDBIFC1(IUPD)</td>
</tr>
<tr>
<td>lores</td>
<td>pk9real</td>
<td>obs. minus low res. value</td>
<td>MDBIFC2(IUPD)</td>
</tr>
<tr>
<td>final</td>
<td>pk9real</td>
<td>obs. minus final value</td>
<td>MDBIOMN(IUPD)</td>
</tr>
</tbody>
</table>

---

10Expanded into tablenames update_1, update_2 and so on
7.3 Database ECMA

7.3.10 Table sat

This table acts as a "middle-man" between table hdr and satellite specific (header-)information tables. See figure 20. Note that not all satellite specific table information is present in both ECMA and CCMA.

Table 27: Database table sat

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>satid</td>
<td>pk1int</td>
<td>Integer equivalent of statid@hdr or zero. Should also equal to ident@hdr</td>
<td>MDB_SATID_AT_SAT</td>
</tr>
<tr>
<td>reo3</td>
<td>@LINK</td>
<td>Offsets &amp; lengths to ...</td>
<td>MLNK_SAT2REO3</td>
</tr>
<tr>
<td>satob</td>
<td>@LINK</td>
<td></td>
<td>MLNK_SAT2SATOB</td>
</tr>
<tr>
<td>satem</td>
<td>@LINK</td>
<td></td>
<td>MLNK_SAT2SATEM</td>
</tr>
<tr>
<td>scatt</td>
<td>@LINK</td>
<td></td>
<td>MLNK_SAT2SCATT</td>
</tr>
<tr>
<td>ssmi</td>
<td>@LINK</td>
<td></td>
<td>MLNK_SAT2SSMI</td>
</tr>
<tr>
<td>atovs</td>
<td>@LINK</td>
<td></td>
<td>MLNK_SAT2ATOVS</td>
</tr>
</tbody>
</table>

7.3.11 Table reo3

This table contains ozone specific header-information. Table is one-to-one (one-looper) aligned with respect to the sat-table.

Table 28: Database table reo3

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>instrument_type</td>
<td>pk1int</td>
<td>Satellite instrument(^{11})</td>
<td>MDBRO3IT</td>
</tr>
<tr>
<td>product_type</td>
<td>pk1int</td>
<td>Product type for retr. atm. gases</td>
<td>MDBRO3PT</td>
</tr>
<tr>
<td>lat_fov-corner[1:4]</td>
<td>pk9real</td>
<td>Latitude field of view corner 1-4</td>
<td>MDBRO3LA1..4</td>
</tr>
<tr>
<td>lon_fov-corner[1:4]</td>
<td>pk9real</td>
<td>Longitude field of view corner 1-4</td>
<td>MDBRO3LO1..4</td>
</tr>
<tr>
<td>solar_elevation</td>
<td>pk9real</td>
<td>Solar elevation</td>
<td>MDBRO3SOE</td>
</tr>
<tr>
<td>fov</td>
<td>pk1int</td>
<td>Field of view number</td>
<td>MDBRO3FOV</td>
</tr>
<tr>
<td>cloud_cover</td>
<td>pk9real</td>
<td>Cloud cover</td>
<td>MDBRO3CLC</td>
</tr>
<tr>
<td>cloud_top_press</td>
<td>pk9real</td>
<td>Pressure at top of cloud</td>
<td>MDBRO3CP</td>
</tr>
</tbody>
</table>

\(^{11}\)Satellite instrument word instrument_type@hdr should be replicated into sensor@hdr in order to have to have more generic SQL’s.
Table 28: Database table reo3  ...continued

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>quality_retrieval</td>
<td>pk1int</td>
<td>Quality of retrieval</td>
<td>MDBRO3QR</td>
</tr>
<tr>
<td>number_layers</td>
<td>pk1int</td>
<td>Number of retrieved layers</td>
<td>MDBRO3NL</td>
</tr>
</tbody>
</table>

7.3.12 Table satob

This table contains SATOB-specific header-information. Variable $\text{mx\_satob\_weight}$ is currently set to 43 (used to be 50), but should in fact reflect the value used in prevailing RTTOV-implementation. This variable must have the same (or larger) value as Fortran90 variable $\text{JPMX\_SATOB\_WEIGHT}$ in file $\text{SRC/ifs/module/parcma.F90}$.

Note that the dimension of column entry $\text{weight[\(\cdot\)]}$ is actually defined via SET-variable $\text{mx\_satob\_weight}$. Dimension of $\text{MDB\_WEIGHT\_AT\_SATOB}(:)$ is $\text{JPMX\_SATOB\_WEIGHT}$.

Table 29: Database table satob

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>comp_method</td>
<td>pk1int</td>
<td>Cloud motion comp. method(^{12})</td>
<td>MDBSBCMCM</td>
</tr>
<tr>
<td>instdata</td>
<td>pk1int</td>
<td>Ins. data used in proc.</td>
<td>MDBSBIUP</td>
</tr>
<tr>
<td>dataproc</td>
<td>pk1int</td>
<td>Data proc. technique used</td>
<td>MDBSBDPT</td>
</tr>
<tr>
<td>QI[3]</td>
<td>pk1int</td>
<td>Three quality indicators</td>
<td>MDBSBQI1..3</td>
</tr>
<tr>
<td>segment_size_x</td>
<td>pk9real</td>
<td>Resolution, x-direction</td>
<td>MDB_SEGMENT_SIZE_X_AT_SATOB</td>
</tr>
<tr>
<td>segment_size_y</td>
<td>pk9real</td>
<td>Resolution, y-direction</td>
<td>MDB_SEGMENT_SIZE_Y_AT_SATOB</td>
</tr>
<tr>
<td>chan_freq</td>
<td>pk9real</td>
<td>Sat. chann. centre freq. [Hz]</td>
<td>MDB_CHAN_FREQ_AT_SATOB</td>
</tr>
<tr>
<td>tb</td>
<td>pk9real</td>
<td>Coldest cluster temperature</td>
<td>MDB_TB_AT_SATOB</td>
</tr>
<tr>
<td>t</td>
<td>pk9real</td>
<td>Temperature at SATOB p</td>
<td>MDB_T_AT_SATOB</td>
</tr>
<tr>
<td>zenith</td>
<td>pk9real</td>
<td>Satellite zenith angle</td>
<td>MDB_ZENITH_AT_SATOB</td>
</tr>
<tr>
<td>shear</td>
<td>pk9real</td>
<td>Diff. in speed 50hPa above/below</td>
<td>MDB_SHEAR_AT_SATOB</td>
</tr>
<tr>
<td>t200</td>
<td>pk9real</td>
<td>200hPa temperature</td>
<td>MDB_T200_AT_SATOB</td>
</tr>
<tr>
<td>t500</td>
<td>pk9real</td>
<td>500hPa temperature</td>
<td>MDB_T500_AT_SATOB</td>
</tr>
<tr>
<td>top_mean_t</td>
<td>pk9real</td>
<td>Mean temp. between 80hPa &amp; p</td>
<td>MDB_TOP_MEAN_T_AT_SATOB</td>
</tr>
<tr>
<td>top_wv</td>
<td>pk9real</td>
<td>Integrated WV above p</td>
<td>MDB_TOP_WV_AT_SATOB</td>
</tr>
<tr>
<td>dr_by_dp</td>
<td>pk9real</td>
<td>Diff in temp. 50hPa above/below</td>
<td>MDB_DT_BY_DP_AT_SATOB</td>
</tr>
<tr>
<td>p_best</td>
<td>pk9real</td>
<td>&quot;Best fit&quot; pressure p</td>
<td>MDB_P_BEST_AT_SATOB</td>
</tr>
</tbody>
</table>

\(^{12}\)Note: Cloud motion computational method word should be replicated into sensor@hdr to be able to have more generic SQL's
### 7.3 Database ECMA

#### 7.3.13 Table satob

This table contains scatterometer specific header-information.

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>u_best</td>
<td>pk9real</td>
<td>U at &quot;best fit&quot; p</td>
<td>MDB_U_BEST_AT_SATOB</td>
</tr>
<tr>
<td>v_best</td>
<td>pk9real</td>
<td>V at &quot;best fit&quot; p</td>
<td>MDB_V_BEST_AT_SATOB</td>
</tr>
<tr>
<td>p_old</td>
<td>pk9real</td>
<td>Originally assigned pressure</td>
<td>MDB_P_OLD_AT_SATOB</td>
</tr>
<tr>
<td>u_old</td>
<td>pk9real</td>
<td>U at old pressure</td>
<td>MDB_U_OLD_AT_SATOB</td>
</tr>
<tr>
<td>v_old</td>
<td>pk9real</td>
<td>V at old pressure</td>
<td>MDB_V_OLD_AT_SATOB</td>
</tr>
<tr>
<td>weight[: ]</td>
<td>pk9real</td>
<td>Weights for obs oper.</td>
<td>MDB_WEIGHT_AT_SATOB(:)</td>
</tr>
</tbody>
</table>

#### 7.3.14 Table scatt

This table contains scatterometer specific body-information and is aligned with respect to generic (scatterometer specific) information in body-table.

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>track</td>
<td>pk1int</td>
<td>Satellite track</td>
<td>MDBSCSAT</td>
</tr>
<tr>
<td>cellno</td>
<td>pk1int</td>
<td>Cell number</td>
<td>MDBSCCNO</td>
</tr>
<tr>
<td>prodflag</td>
<td>pk1int</td>
<td>Product flag</td>
<td>MDBSCPFL</td>
</tr>
<tr>
<td>uwi_esa[2]</td>
<td>pk1int</td>
<td>UWI ESA FLAG (2 × 12-bits)</td>
<td>MDB_UWI_ESA_AT_SCATT(2)</td>
</tr>
<tr>
<td>scatt_body</td>
<td>@LINK</td>
<td>Offset&amp;length to scatt_body</td>
<td>MLNK_SCATT2SCATT_BODY</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>azimuth</td>
<td>pk9real</td>
<td>Beam azimuth angle</td>
<td>MDBSCBAA</td>
</tr>
<tr>
<td>incidence</td>
<td>pk9real</td>
<td>Beam incidence angle</td>
<td>MDBSCBIA</td>
</tr>
<tr>
<td>Kp</td>
<td>pk9real</td>
<td>Instrument/Antenna Noise</td>
<td>MDBSCKP</td>
</tr>
<tr>
<td>invresid_1</td>
<td>pk9real</td>
<td>Inv. resid. of 1\textsuperscript{st} solution</td>
<td>MDBSCIRF</td>
</tr>
<tr>
<td>invresid_2</td>
<td>pk9real</td>
<td>Inv. resid. of 2\textsuperscript{nd} solution</td>
<td>MDBSCRIS</td>
</tr>
<tr>
<td>dirskill</td>
<td>pk9real</td>
<td>Directional skill</td>
<td>MDBSCDIS</td>
</tr>
<tr>
<td>mpc</td>
<td>pk1int</td>
<td>Missing packet counter</td>
<td>MDB_MPC_AT_SCATT_BODY</td>
</tr>
</tbody>
</table>

**continued on next page...**
Table 31: Database table `scatt_body`  ...continued

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>wvc_qf</td>
<td>pk1int</td>
<td>Wind vector cell quality flag</td>
<td>MDB_WVC_QF_AT_SCATT_BODY</td>
</tr>
<tr>
<td>nretr_amb</td>
<td>pk1int</td>
<td>No. of retrieved ambiguities</td>
<td>MDB_NRETR_AMB_AT_SCATT_BODY</td>
</tr>
</tbody>
</table>

7.3.15 Table `ssmi`

This table is currently undergoing a major revision. Look out for changes in future releases.

Table 32: Database table `ssmi`

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>orbit</td>
<td>pk1int</td>
<td>Orbit number</td>
<td>MDBORNO</td>
</tr>
<tr>
<td>scanline</td>
<td>pk1int</td>
<td>Scan line number</td>
<td>MDBSLNO</td>
</tr>
<tr>
<td>scanpos</td>
<td>pk1int</td>
<td>Position number along scan</td>
<td>MDBPNAS</td>
</tr>
<tr>
<td>typesurf</td>
<td>pk1int</td>
<td>Type of surface</td>
<td>MDBTOS</td>
</tr>
<tr>
<td>vertsign</td>
<td>pk1int</td>
<td>Vertical significance</td>
<td>MDBVSG</td>
</tr>
<tr>
<td>zenith</td>
<td>pk1int</td>
<td>Zenith angle</td>
<td>MDBZAN</td>
</tr>
<tr>
<td>iterno_conv_</td>
<td>pk1int</td>
<td>No. of iterations for convergence</td>
<td>MDB1DNIT</td>
</tr>
<tr>
<td>ldvar</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>failure_1dvar</td>
<td>pk1int</td>
<td>Failure indicator</td>
<td>MDB1DFIN</td>
</tr>
<tr>
<td>surfpress[1]</td>
<td>pk9real</td>
<td>Surface pressure</td>
<td>MDB_SURFPRESS_AT_SSMI(1)</td>
</tr>
<tr>
<td>skintemp[1]</td>
<td>pk9real</td>
<td>Skin temperature</td>
<td>MDB_SKINTEMP_AT_SSMI(1)</td>
</tr>
<tr>
<td>u10m[1]</td>
<td>pk9real</td>
<td>u-component of 10 metre wind</td>
<td>MDB_U10M_AT_SSMI(1)</td>
</tr>
<tr>
<td>v10m[1]</td>
<td>pk9real</td>
<td>v-component of 10 metre wind</td>
<td>MDB_V10M_AT_SSMI(1)</td>
</tr>
<tr>
<td>prec_st[2]</td>
<td>pk9real</td>
<td>Surface strat. precip. ???</td>
<td>MDB_PREC_ST_AT_SSMI(1:2)</td>
</tr>
<tr>
<td>prec_cv[2]</td>
<td>pk9real</td>
<td>Surface conv. precip. ???</td>
<td>MDB_PREC_CV_AT_SSMI(1:2)</td>
</tr>
<tr>
<td>ssmi_body</td>
<td>@LINK</td>
<td>Offset &amp; length to table ...</td>
<td>MLNK_SSMI2SSMI_BODY</td>
</tr>
</tbody>
</table>

7.3.16 Table `ssmi_body`

Table 33: Database table `ssmi_body`

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>radcost</td>
<td>pk9real</td>
<td>1dvar radiance cost</td>
<td>MDB_RADCOST_AT_SSMI_BODY</td>
</tr>
</tbody>
</table>

continued on next page...
### 7.3 Database ECMA

#### 7.3.17 Table atovs

Table 34: Database table atovs

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>scanline</td>
<td>pk1int</td>
<td>Scan line number</td>
<td>MDBSCLA</td>
</tr>
<tr>
<td>fov</td>
<td>pk1int</td>
<td>Field of view number</td>
<td>MDBFOVA</td>
</tr>
<tr>
<td>station_height</td>
<td>pk9real</td>
<td>Height of station</td>
<td>MDBSTHA</td>
</tr>
<tr>
<td>zenith</td>
<td>pk9real</td>
<td>Satellite zenith angle</td>
<td>MDBSAZA</td>
</tr>
<tr>
<td>bearing_azimuth</td>
<td>pk9real</td>
<td>Bearing or azimuth</td>
<td>MDSABA</td>
</tr>
<tr>
<td>solar zenith</td>
<td>pk9real</td>
<td>Solar zenith angle</td>
<td>MDBSOZA</td>
</tr>
<tr>
<td>solar azimuth</td>
<td>pk9real</td>
<td>Solar azimuth</td>
<td>MDSOBA</td>
</tr>
<tr>
<td>cldtop[1:3]</td>
<td>pk9real</td>
<td>Cloud top press. from HIRS</td>
<td>MDBC_DPTOPL_3_AT_ATOVS</td>
</tr>
<tr>
<td>cldne[1:3]</td>
<td>pk9real</td>
<td>Cloud emissivity from HIRS</td>
<td>MDBC_CLDE_3_AT_ATOVS</td>
</tr>
<tr>
<td>vertsign[3:4]</td>
<td>pk1int</td>
<td>Vertical signification</td>
<td>MVBVSA.4A</td>
</tr>
<tr>
<td>landsea</td>
<td>pk1int</td>
<td>Land/Sea qualifier</td>
<td>MDBLSQA</td>
</tr>
<tr>
<td>landsurf_height</td>
<td>pk9real</td>
<td>Height of land surface</td>
<td>MDBHLSA</td>
</tr>
<tr>
<td>skintemp</td>
<td>pk9real</td>
<td>Skin temperature</td>
<td>MDSKTA</td>
</tr>
<tr>
<td>press_2</td>
<td>pk9real</td>
<td>Pressure 2</td>
<td>MDBPP2A</td>
</tr>
</tbody>
</table>

continued on next page...
Table 34: Database table **atovs** ...*continued*

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>iterno_1dvar</td>
<td>pk1int</td>
<td>1DVAR iteration number</td>
<td>MDB1DITA</td>
</tr>
<tr>
<td>error_1dvar</td>
<td>pk1int</td>
<td>1DVAR error(s)</td>
<td>MDB1DERA</td>
</tr>
<tr>
<td>channel_-1dvar_0</td>
<td>pk1int</td>
<td>1DVAR sat. channel(s) used</td>
<td>MDB1DCUA</td>
</tr>
<tr>
<td>channel_-1dvar_1</td>
<td>pk1int</td>
<td>1DVAR sat. channel(s) used (cont.)</td>
<td>MDB1DCUA1A</td>
</tr>
<tr>
<td>presat_flags</td>
<td>pk1int</td>
<td>PRESAT summary flags</td>
<td>MDB1DPFA</td>
</tr>
<tr>
<td>iterno_conv_-1dvar</td>
<td>pk1int</td>
<td>1DVAR no. of iter. for convergence</td>
<td>MDB1DINIA</td>
</tr>
<tr>
<td>failure_1dvar</td>
<td>pk1int</td>
<td>1DVAR failure indicator</td>
<td>MDB1DFIA</td>
</tr>
<tr>
<td>landsea_obs</td>
<td>pk1int</td>
<td>Land/sea surface from obs. report</td>
<td>MDB_LANDSEA_OBS_AT_ATOVs</td>
</tr>
<tr>
<td>atovs_pred</td>
<td>@LINK</td>
<td>Offset&amp;length to <strong>atovs_-_pred</strong></td>
<td>MLNK_ATOVS2ATOVS_PRED</td>
</tr>
</tbody>
</table>

7.3.18 **Table** **atovs_pred**

In the following table the dimension (i.e. number of columns) for the item **skintemp** is the number of updates ($NMXUPD$).

Table 35: Database table **atovs_pred**

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>predictor[16]</td>
<td>pk9real</td>
<td>Sixteen predictors</td>
<td>MDB1BCP1..16A</td>
</tr>
<tr>
<td>skintemp[:]</td>
<td>pk9real</td>
<td>Skin temperature per update</td>
<td>MDB1BCPTS1()</td>
</tr>
</tbody>
</table>

7.3.19 **Table** **bufr**

This table contains raw BUFR-data in bulk 4-byte chunks.

Table 36: Database table **bufr**

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>bufr</td>
<td>BUFR-data, 4-bytes per element</td>
<td>MDB_MSG_AT_BUFR</td>
</tr>
</tbody>
</table>
7.4 Database ECMASCR

Database ECMASCR has the same hierarchy as ECMA (see figure ??). The ECMASCR should have the same layout as ECMA. Its purpose, if used at all, is to allow creation of "raw" ECMA-database, where observation distribution between pools or MPI-tasks is not perfect. The program odbhshuffle can create a more balanced database ECMA from ECMASCR.

7.5 Database CCMA

The CCMA is a stripped-down version of database ECMA. It contains fewer tables and column entries than ECMA. In essence it contains only active observations after IFS Screening process for use by 4D-Var minimization. Only the following tables are available for database CCMA:

<table>
<thead>
<tr>
<th>Table name</th>
<th>Differs from ECMA ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>desc</td>
<td>NO</td>
</tr>
<tr>
<td>ddrs</td>
<td>NO</td>
</tr>
<tr>
<td>poolmask</td>
<td>NO</td>
</tr>
<tr>
<td>timeslot_index</td>
<td>NO</td>
</tr>
<tr>
<td>index</td>
<td>NO</td>
</tr>
<tr>
<td>hdr</td>
<td>YES</td>
</tr>
<tr>
<td>sat</td>
<td>YES</td>
</tr>
<tr>
<td>reo3</td>
<td>NO</td>
</tr>
<tr>
<td>satob</td>
<td>NO</td>
</tr>
<tr>
<td>atovs</td>
<td>YES</td>
</tr>
<tr>
<td>atovs_pred</td>
<td>NO</td>
</tr>
<tr>
<td>body</td>
<td>YES</td>
</tr>
<tr>
<td>errstat</td>
<td>NO</td>
</tr>
<tr>
<td>update-tables</td>
<td>NO</td>
</tr>
</tbody>
</table>

Only the tables which differ from ECMA are explained in the next subsections. For tables that do not differ, please refer directly to ECMA-tables in 7.3.

7.5.1 Table hdr

The column entries are the same as described in 7.3.6, apart from those available for ECMA only. Also some flags have less members for CCMA database, see for example 7.2.3.

7.5.2 Table sat

This table acts as a "middle-man" between table hdr and satellite specific (header-)information tables. Only the following entries are available for CCMA:
Table 37: Database table sat

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>satid</td>
<td>pk1int</td>
<td>Integer equivalent of statid@hdr or zero Should also equal to ident@hdr</td>
<td>MDB_SATID_AT_SAT</td>
</tr>
<tr>
<td>reo3</td>
<td>@LINK</td>
<td>Offsets &amp; lengths to ... reo3</td>
<td>MLNK_SAT2REO3</td>
</tr>
<tr>
<td>satob</td>
<td>@LINK</td>
<td>satob</td>
<td>MLNK_SAT2SATOB</td>
</tr>
<tr>
<td>atovs</td>
<td>@LINK</td>
<td>atovs</td>
<td>MLNK_SAT2ATOVS</td>
</tr>
</tbody>
</table>

7.5.3 Table atovs

Only the following entries are available for CCMA:

Table 38: Database table atovs

<table>
<thead>
<tr>
<th>Column entry</th>
<th>Type</th>
<th>Description</th>
<th>Column pointer(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>scanline</td>
<td>pk1int</td>
<td>Scan line number</td>
<td>MDBSCLA</td>
</tr>
<tr>
<td>fov</td>
<td>pk1int</td>
<td>Field of view number</td>
<td>MDBFOVA</td>
</tr>
<tr>
<td>atovs_pred</td>
<td>@LINK</td>
<td>Offset &amp; length to atovs_-&gt;pred</td>
<td>MLNK_ATOVS2ATOVS_PRED</td>
</tr>
</tbody>
</table>

7.5.4 Table body

The column entries are the same as described in 7.3.7, apart from those available for ECMA only. Also some flags have less members for CCMA database, see for example 7.2.3.
8 Guidelines for modifications

This section gives guidance on how to modify items in the ODB interface to IFS. Discussion in this section relates to database ECMA and CCMA only. The section covers how to

- add a column entry (page 165),
- modify a column entry (page 166),
- remove a column entry (page 167),
- add an array of column entries (page 167),
- modify an existing data query (page 168),
- add a data query (page 169),
- add a new retrieval (page 170),
- modify an existing retrieval (page 173),
- add a new table (page 173)

8.1 A checklist

Before going into too much detail it maybe useful to summarize through a checklist what needs to be done when a new column entry has to be added, so that the IFS-code will be able to access this new data from ODB. This the most common situation that user needs to address. The checklist presented here paves the way to successfully accomplish this task.

Our example shows how to add a column entry xyz@body i.e. an entry called xyz to table body and how to access this data column from IFS-code with the MDB-pointer MDB_\_XYZ\_AT\_BODY, the data which has been requested to IFS through SQL data queries. The database used in this checklist is the ECMA, but it could as well been the CCMA, in which case you need to substitute the word ECMA with CCMA.

Table 39: A checklist how to add a new column entry

<table>
<thead>
<tr>
<th>Step#</th>
<th>Need to do</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Add the column entry xyz@body</td>
<td>Edit file odb/ddl/cma.h</td>
</tr>
<tr>
<td>2</td>
<td>Introduce the MDB-pointer MDB_XYZ_AT_BODY</td>
<td>Edit files ifs/common/yomdb_defs.h and ifs/common/yomdb_vars.h</td>
</tr>
<tr>
<td>3</td>
<td>Create a connection between the column entry and the MDB-pointer</td>
<td>Edit file odb/cma2odb/initmdb.F90</td>
</tr>
</tbody>
</table>

continued on next page . . .
Table 39: A checklist how to add a new column entry …continued

<table>
<thead>
<tr>
<th>Step#</th>
<th>Need to do</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Add (normally) two data queries</td>
<td>Create files odb/ddl/testhdr.sql and odb/ddl/testbody.sql</td>
</tr>
<tr>
<td>5</td>
<td>Create symbolic links</td>
<td>Make symbolic links from odb/ddl.ECMA/testhdr.sql and odb/ddl.ECMA/testbody.sql to the previous data query files under odb/ddl</td>
</tr>
<tr>
<td>6</td>
<td>Introduce a new retrieval by setting the context name to ’TESTHDR’ and its viewnames to ’testhdr’ and ’testbody’, respectively to match the SQL-queries</td>
<td>Edit file odb/cma2odb/ctxinitdb.F90</td>
</tr>
<tr>
<td>7</td>
<td>Add statements USE yomdb and #include &quot;openmp_obs.h&quot; to enable access to the database</td>
<td>Edit the IFS-routine which needs access to observations</td>
</tr>
<tr>
<td>8</td>
<td>Extract data from ODB to observational arrays by adding calls to GETDB(’TESTHDR’,…) after which you will be able to access the new data column via ROBODY(…,MDB_XYZ_AT_BODY)</td>
<td>Continue editing the IFS-routine</td>
</tr>
<tr>
<td>9</td>
<td>Call PUTDB(’TESTHDR’,…) to put data back to the database</td>
<td>Further continue editing the IFS-routine</td>
</tr>
</tbody>
</table>

8.2 Important files and directories

The basic files and directories you need to become familiar with are as follows

- data queries (SQL) or data definition layout for a particular database, you need to look at directories $SRC/odb/ddl and $SRC/odb/ddl.dbname, where dbname can for example be database ECMA,
- ODB to IFS interface go to directory $SRC/odb/cma2odb (and sometimes $SRC/odb/tools) and also be aware of $SRC/ifs/common and possibly file $SRC/ifs/module/yomdb.F90,
8.3 Adding a column entry

- BUFR2ODB-software (or ODB2BUFR) please refer to files under $SRC/odb/bufr2odb and (possibly) directory $SRC/odb/tools.

It is important to note that ODB to IFS interface revolves mainly around the following files and directories:

- files under directories $SRC/odb/ddl and $SRC/odb/ddl.dbname, of which most notable is $SRC/odb/ddl/cma.h,
- files $SRC/ifs/common/yomdb_defs.h and $SRC/ifs/common/yomdb_vars.h,
- main programs in directory $SRC/odb/tools,
- and sometimes files in directories $SRC/ifs/module, $SRC/ifs/obs_preproc, $SRC/ifs/pp_obs and $SRC/ifs/var in order to modify subtle things in IFS observation handling.

These are usually the only files and directories one needs to worry about. If the core ODB software needs amending, then one needs to look at directories $SRC/odb/module, $SRC/odb/lib and $SRC/odb/aux, or $SRC/odb/compiler for ODB/SQL-compiler source code.

8.3 Adding a column entry

Suppose you need to add a new column entry xyz to table body. You need to do the following to get modifications to the ODB interface to IFS:

1. Choose the datatype for the new column entry. Normally it is
   - pk9real for (64-bit) floating point numbers
   - pk1int for (32-bit) integers
   - string for character strings (element size is 8 bytes)
   - YYYYMMDD for dates in format YYYY (year), MM (month) and DD (day)
   - HHMMSS for times in format HH (hours), MM (minutes) and SS (seconds)

\footnote{Do not use 64-bit integers yet!}
2. Insert line (say datatype is *pk9real*) with description
   *xyz pk9real, // description*
   in file $SRC/odb/ddl/cma.h after the line
   `CREATE TABLE body`

3. Choose the name of the corresponding MDB-pointer.
   The recommended name would be (assuming it fits into 31 characters):
   `MDB_XYZ_AT_BODY`

4. Insert the following line at the end of file $SRC/ifs/common/yomdb_defs.h:
   `#define MDB_XYZ_AT_BODY o_(it)%mdb_xyz_at_body`

5. Insert the following line at the end of file $SRC/ifs/common/yomdb_vars.h:
   `INTEGER_M :: mdb_xyz_at_body ! 'xyz@body'`

6. Insert the following line in alphabetically conforming location
   in file $SRC/odb/cma2odb/initmdb.F90:
   `CALL cmdb_REG('xyz@body', 'MDB_xyz_AT_body', MDB_XYZ_AT_BODY, it, rc)`

Note that the case of the letters in steps 4–6 must be exactly like shown, otherwise C-
preprocessor will not pick up correct translation for `MDB_XYZ_AT_BODY`.

Since this new entry is now available, you may want to modify or even add a new SQL, too.
This is explained in the next sections. All these changes will trigger recompilation of libraries
libifs.a, libodbport.a, libECMA.a, libECMASCR.a and libCCMA.a.

### 8.4 Modifying a column entry

This is probably simplest thing to do, but probably happens also very seldom. A typical
situation might be to rename an existing column name to something more descriptive and
then standardize on this new definition.

Modifications start by editing and changing data definition layout include-file $SRC/odb/ddl/cma.h
– in case of databases CCMA, ECMA and ECMASCR. Otherwise check the file with `ddl`-suffix
in database directory, like the file `PREODB.ddl` in
$SRC/odb/ddl.PREODB for database
`PREODB`.

Then proceed with changes to the MDB-pointer mapping from file $SRC/odb/cma2odb/initmdb.F90:
search for the old column name and rename it. After this also adjust $SRC/ifs/common/yomdb_-
vars.h and $SRC/ifs/common/yomdb_defs.h accordingly. Finally modify any
SQL-files under $SRC/odb/ddl to accomodate the modified column entry over the old one.

An example...ugh ?
8.5 Removing a column entry

To remove a column entry just edit the file $SRC/odb/ddl/cma.h, search for the column name to be eliminated and delete it. After this it is necessary to remove column’s MDB-pointer mapping from file $SRC/odb/cma2odb/initmdb.F90. Furthermore it is desirable to clean the column up from files $SRC/ifs/common/yomdb_vars.h and $SRC/ifs/common/yomdb_defs.h, too.

The final check is ensure that no SQL-file refers to this obsolescent entry anymore; so files with sql-suffix under directory $SRC/odb/ddl need to be checked. And that is all.

An example...ugh?

8.6 Adding an array of column entries

Suppose one needs to add three new column entries, this time into table hdr. And all these column entries share the same characteristics, so they could be viewed as array of column entries, say key[3]. Since this is equivalent to adding manually three single column entries key_1, key_2 and key_3, one should understand instructions explained already in 8.3. But there is a more convenient way to do the modification, when multiple (array) columns are concerned:

1. Choose the datatype for the column entries (as in 8.3)
2. Insert line (say datatype is pk9real)
   
   key[3] pk9real, // description

   into file $SRC/odb/ddl/cma.h in the appropriate place after line

   CREATE TABLE hdr

   Note that this was just a shorthand notation to the following:

   key_1 pk9real, // description
   key_2 pk9real, // description
   key_3 pk9real, // description

3. Choose the name of the corresponding MDB-pointer, which should now also be an array, as follows:

   MDB_KEY_AT_HDR(3)

4. Insert the following line at the end of file $SRC/ifs/common/yomdb_defs.h:

   #define MDB_KEY_AT_HDR o_(it)% mdb_key_at_hdr

5. Insert the following line at the end of file $SRC/ifs/common/yomdb_vars.h:

   INTEGER_M :: mdb_key_at_hdr(3) ! 'key[3]@hdr'

6. Insert the following line in alphabetically correct location
   in file $SRC/odb/cma2odb/initmdb.F90:
CALL cmdb_VECREG('key@hdr', 'MDB_key_AT_hdr', MDB_KEY_AT_HDR, 1, 3, it, rc)
This will initialize all 3 entries in one sweep and is identical with the following:
CALL cmdb_REG('key_1@hdr', 'MDB_key_AT_hdr(1)', MDB_KEY_AT_HDR(1), it, rc)
CALL cmdb_REG('key_2@hdr', 'MDB_key_AT_hdr(2)', MDB_KEY_AT_HDR(2), it, rc)
CALL cmdb_REG('key_3@hdr', 'MDB_key_AT_hdr(3)', MDB_KEY_AT_HDR(3), it, rc)

8.7 Modifying an existing data query

To see the effect of your new column entry as described in 8.3, you may just want to modify existing SQLs. All SQLs are situated under directory $SRC/odb/ddl, independently whether they belong to databases ECMA or CCMA or so. Database specific SQLs are found under $SRC/odb/ddl.ECMA (for ECMA in this case), but are all symbolic links to the physical SQL-files under $SRC/odb/ddl.

Before adding reference to the new entry xyz@hdr in the SQL, you should answer the following questions:

- In which SQL-file(s) this entry should be put ?
- Is the new entry a floating point/string or an integer/bitmap column ?

In this particular case it seems that you want to pass values of the new entry xyz@hdr into the ROBHDR, or solely to the MOBHDR if it is an integer/bitmap value. In some rare cases SATHDR or SATPRED may also be relevant.

Suppose it is both ROBHDR and MOBHDR that are involved. In this case we nearly always fill ROBHDR by calling ODB_get, and then we copy N first columns of ROBHDR into MOBHDR. (Pending CTXINITDB definitions explained in 8.9) (Please see CTXGETDB and INTCOLDB). These first columns of SQL have to be genuinely integer/bitmap-valued or integer overflow/underflow or floating point exception will occur. If any integer valued columns are found after i\textsuperscript{th} column, the check performed by routine INTCOLDB, will ensure program will abort.

Bearing in mind these limitations, you indeed need to know what your column datatype is that you are about to add into the existing SQL. Therefore there are two possibilities:

1. If you have the entry \texttt{seqno} present (recommended), it must be always the first entry in the header and body data queries (in order to get the variable MLNKH2B right)
2. If the datatype is integer-kind, add column at the beginning, but only after entry \texttt{seqno}, if that is present
3. If the datatype is floating point-kind, then add the column at the end of SELECT-statement’s list of column entries (after the integer-kinds)
8.8 Adding a new data query

In this particular example of \texttt{xyz@hdr}, the change would be as follows, when datatype is \textit{pk9real}:

<table>
<thead>
<tr>
<th>Original SQL</th>
<th>Modified SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{CREATE VIEW testhdr AS} \texttt{SELECT seqno, obstype, lat, lon} \texttt{FROM index, hdr} \texttt{WHERE tslot = 1}}</td>
<td>\texttt{CREATE VIEW testhdr AS} \texttt{SELECT seqno, obstype, lat, lon, \texttt{xyz} \texttt{FROM index, hdr} \texttt{WHERE tslot = 1}}</td>
</tr>
</tbody>
</table>

and as follows, when datatype is \textit{pk1int}:

<table>
<thead>
<tr>
<th>Original SQL</th>
<th>Modified SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{CREATE VIEW testhdr AS} \texttt{SELECT seqno, obstype, lat, lon} \texttt{FROM index, hdr} \texttt{WHERE tslot = 1}}</td>
<td>\texttt{CREATE VIEW testhdr AS} \texttt{SELECT seqno, \texttt{xyz}, obstype, lat, lon} \texttt{FROM index, hdr} \texttt{WHERE tslot = 1}}</td>
</tr>
</tbody>
</table>

8.8 Adding a new data query

A new set of SQLs typically needs to be added for a new retrieval associated with a new pair of \texttt{GETDB} and \texttt{PUTDB} calls inserted into IFS. Since SQLs could be generic i.e. used by both \texttt{ECMA} and \texttt{CCMA}, it is strongly advised to create them under \texttt{$SRC/odb/ddl} directory.

Under the ECMWF’s ClearCase version handling system adding new SQLs to ODB IFS -interface obeys the following guidelines:

1. Go to \texttt{/ccvobs/odb/ddl}:
   \begin{verbatim}
   % cd /ccvobs/odb/ddl
   \end{verbatim}

2. Add and edit SQL file:
   \begin{verbatim}
   % addfile -f newfile.sql
   \end{verbatim}

3. Go to database ECMA specific directory (becomes current directory):
   \begin{verbatim}
   % cd /ccvobs/odb/ddl/ECMA
   \end{verbatim}
4. Check-out current directory (cleartool-command alias is normally ct):

   % ct co -nc .

5. Create relative ClearCase symbolic link to SQL-file:

   % ct ln -s ../ddl/newfile.sql .

6. Check-in current directory directory:

   % ct ci -nc .

If SQLs are also related to CCMA, you have to repeat steps 3–6 under directory the /ccvobs/odb/ddl.CCMA too.

If this is not in ClearCase, then replace in the previous /ccvobs with $SRC, ignore steps 4 and 6, replace addfile with your preferred editor command, and finally replace ct ln with regular symbolic link command ln.

8.9 Adding a new retrieval

To do this you need to modify file $SRC/odb/cma2odb/ctxinitdb.F90. A typical case is where four SQLs get assigned to one new retrieval. The first SQL is normally a driving force to fill ROBHDR and the third SQL drives ROBODY filling. The second and the fourth one are normally for MOBHDR and MOBODY, respectively. Typically they do not get their values by executing SQL (ODB-select followed by ODB_get) to fill the arrays, but copy the N first columns out of corresponding ROBHDR and ROBODY – as described in 8.7.

Here is the outline of what you should add to routine $SRC/odb/cma2odb/ctxinitdb.F90:

1. Decide if this retrieval is meant for database ECMA or CCMA or both, or has some other condition to get initialized. If it is for only, say, ECMA, make use of predefined flag LLECMA

2. Suppose you need to add one new retrieval called HIPHOP, version zero (0). It will be bound to SQLs explained in 8.7, 4 views are involved, but 2 of them share data with their master (e.g. MOBHDR → ROBHDR). In most of the cases you just need to add the following piece of code (at the bottom of the source file, just before end of internal subroutine delayed_init):
if (LLECMA) then
  CALL next_context('HIPHOP', version=0, nviews=4)
  do it=1,inumt
    ctx(idctx,it)%view(1)%name = 'testhdr' ! SQL-name
    ctx(idctx,it)%view(1)%ncase = JPCASE_ROBHDR
    ctx(idctx,it)%view(2)%name = 'testhdr'
    ctx(idctx,it)%view(2)%ncase = JPCASE_MOBHDR
    ctx(idctx,it)%view(2)%execute = .FALSE.
    ctx(idctx,it)%view(2)%shared_index = 1 ! Ref. to view(1)
    ctx(idctx,it)%view(2)%mlnh2b = 4   ! Ref. to view(4)
  enddo
endif

After this one can add the following new GETDB and PUTDB calls to IFS to get some action:

```
SUBROUTINE hiphop(...)
USE YOMDB
...
  implicit none
  INTEGER(kind=JPRM) :: info(0)
  REAL(kind=JPRB)   :: zinfo(0)
...
  INTERFACE
    #include "getdb.h"
    #include "putdb.h"
  END INTERFACE
...
  #include "openmp_obs.h"
  !-- begin : the very first executable statement
  !-- Execute SQLs
  iret=0
  iversion=0
  CALL GETDB('HIPHOP',iversion,iret, &
              & info, size(info), zinfo, size(zinfo), &
              & -1, -1, -1, -1, -1, -1)```
if (iret > 0) then
   !-- Do something with ROBHDR, MOBHDR, ROBODY & MOBODY
endif

...  
!-- Update back to (in-memory) database
! (usually the last thing in the file)
CALL PUTDB(’HIPHOP’, iversion, iret, &
   & info, size(info), zinfo, size(zinfo))
END SUBROUTINE hiphop

Subroutine HIPHOP executes SQLs to retrieve data into arrays ROBHDR etc. There are two important points. Firstly, this is OpenMP-thread-safe i.e. the observational arrays are filled per OpenMP-thread and this subroutine can be called safely from within an OpenMP-loop. Secondly, the way we define the retrieval in subroutine CTXINITDB did not impose any MPI-parallelism i.e. observational arrays are filled using only local observational data\(^{14}\). That is to say, from data pools belonging to the owner MPI-task.

The first point means that you can have many retrievals open simultaneously as long as there is only one per OpenMP-thread. To top up the second point, it is also possible to retrieve observational data globally. For example, you may want to view a global flight path and not care about how the aircraft observations were distributed across MPI-tasks or data pools. To accomplish this you just need to add one line into CTXINITDB after beginning of your do it=1, inumt-loop:

\[
\begin{align*}
do & \text{ it}=1, \text{inumt} \\
   & \text{ctx(idctx,it)}\%\text{replicate}_PE = -1 \leftarrow \text{add this line} \\
   & \text{ctx(idctx,it)}\%\text{view}(1)
\end{align*}
\]

This is very convenient but also currently very expensive. You essentially ask for data to be replicated across MPI-tasks. Not only is the volume of data much larger than the locally owned data, but also extra operations need to be performed by the core ODB library to communicate data between MPI-tasks. So only use this option if you really need it.

The characteristics of the function \textit{ODB\_put} (called indirectly by the routine \textit{PUTDB}) are such that in case of replicated views, the update is applied only to local data. That is to say, you can modify as much as you like remote data in your observational arrays, but \textit{ODB\_put} will not put these modifications back to their remote MPI-tasks. This stems from the fact that \textit{ODB\_put} does not perform any message passing between MPI-tasks (when replicate\_PE was set to -1), whereas data gathering phase (\textit{ODB\_get}) involves message passing for replicated views.

\(^{14}\)The variable replicate\_PE was not set to -1
It is important to note that replicated views can only work correctly if your program runs in single threaded OpenMP-mode, that is to say: do not use replicated retrievals, if the calling subroutine is in an OpenMP-loop.

8.10 Modifying an existing retrieval

Usually this involves modifications to routine $SRC/odb/cma2odb/ctxinitdb.F90 only, and rarely to $SRC/odb/cma2odb/getdb.F90.

An example...ugh?

8.11 Adding a new table

To add a new table, you often need to modify many files, since it may affect several functions in the ODB interface to IFS. Luckily it is very seldom that new tables need to be introduced, since it is preferable to use the existing ones.

Anyway, here is one recipe for doing the modification. Suppose the table name is called abcd. It all starts by modifying our usual data layout file $SRC/odb/ddl/cma.h. Decide how the table to be added needs to be related with respect to other tables and add @LINK to this new table from parent-to-be table as follows:

```plaintext
abcd @LINK,
```

Then proceed by adding the table definition. Usually it suffices to add this at the end of data layout file. Do something like this:

```plaintext
CREATE TABLE abcd (  
    col_1 pk1int , // description for column#1  
    col_2 pk9real , // description for column#2  
    col_3 string ,// description for column#3  
);
```

Decide whether this table needs to be visible to the CCMA-database at all. If not, then encapsulate the previous table definition with the conditional block:

```plaintext
#ifdef ECMA  
// table definition here (visible to ECMA & ECMASCR only)  
#endif
```
After this modify the file `$SRC/odb/cma2odb/ctxinitdb.F90` in order to add the new table in the list of "recognized" tables (variable tables) and possibly add some retrievals as explained before. Also new data queries are probably needed, or at least existing one may need to refer to the new table (FROM-clause).

The rest is quite mechanical: all the columns in the new table are completely unknown to the ODB interface to IFS and the system needs to be told about these. So follow instructions in section 8.3.

Finally, this new table may be loaded outside the ODB interface to IFS. Otherwise routine `$SRC/odb/cma2odb/putatdb.F90` needs to be understood, but is out of the scope of this document ;-)

\[15\] Some external ODB-program which directly fills up the table – and makes a correct parental
9 Using other database layouts

This section describes other recognized (i.e. permanent) databases, how to create your own databases which may or may not have anything to do with observation handling, and how to create permanent database layouts which will be automatically recognized by the ODB software without “work-arounds”.

Currently the recognized databases are CCMA, ECMA and ECMASCRI described in the previous chapter, and also database PREODB – mostly for historical reasons. The latter one is described below.

9.1 Database PREODB

This was the very first ODB-database – already well before the ODB software went operational at ECMWF.

At ECMWF this database was used to feed ERA40-reanalysis with conventional observations. The main application was to perform duplicate data checking on PREODB prior to analysis, since observations in this database were derived from many sources (meteorological institutes and organizations) around the world and there were lots of potential duplicate observations which, if entered to the analysis, would have completely destroyed its quality.

There is over 200GB of online data in PREODB that covers ERA-40 assimilation from September 1957 till December 2001. Data are split into 6 hourly time-windows. When duplicate checking was performed, only the most trusted observations (and a unique set among duplicates) were selected for the ERA-40 3D-FGAT-analysis.

Database PREODB contains only four tables: desc, hdr, body and bufr. The end result of duplicate checking procedure actually uses mainly tables hdr and body to pull out the right BUFR-messages for ERA-40 analysis. A typical SQL was as follows (say, for 19780202, 12UTC):

```sql
SET $andate = 19780202;
SET $antime = 120000;
CREATE VIEW query AS
    SELECT msg@bufr
    UNIQUEBY seqno, andate, antime
    FROM desc, hdr, body, bufr
    WHERE andate = $andate
    AND antime = $antime
    AND duplseqno[1] == 0
;
```

The key element was the item duplseqno[1] == 0 in the WHERE-condition, since it
indicadated whether the selected observation (BUFR-message) was either free from duplicates or was the most trusted observation to be selected into the analysis. Details of how duplicate checking is performed can be seen from figures 22 and 23\textsuperscript{16}.

### 9.2 User defined, temporary databases

In the latest ODB software release it is possible to introduce user defined database layouts into the system relatively easily. This is especially true for the interactive ODB environment.

Figure 21: Use `newodb` command to create your own database library

![Diagram of ODB system components](image)

User defined database differ from permanent databases in such a way that the ODB software has built-in (apriori) knowledge of permanent databases, whereas user defined databases need to be explicitly introduced to the software on-the-fly (technically speaking: through the file `_odb_glue.o`).

The easiest way to introduce user defined database is to

1. Decide the name of your database (say MYDB)\textsuperscript{17}
2. Create data definition layout file `MYDB.ddl` containing at least table definition(s) and linking structure between tables

\textsuperscript{16}Kindly supplied by Vanda Da Costa Bechtold, ECMWF

\textsuperscript{17}Must start with a letter `[A–Z]` optionally followed by letter(s) `[A–Z]` and/or number(s) `[0–9]`. No underscores, lowercase letters or blanks allowed
3. Create database specific library `libMYDB.a` by using command:

```
% newodb MYDB
```

4. Supply SQL-file(s) associated with this database and compile them into library `libMYDB.a` as follows (say you have only two queries):

```
% odbcomp -lMYDB query1.sql query2.sql
```

5. Compile (or just re-link) your ODB-application program via command:

```
% odbf90 myprog.F90 -lMYDB -o myprog.x
```

6. Run

```
% ./myprog.x [optional_input_parameters]
```

When your need to supply another set of SQL-files (and disregard the ones supplied in step#4), you just have to repeat the procedure from step#4. That is to say re-compilation of new set of SQLs is necessary as well as re-linking of the ODB-application program.

The command `newodb` is in fact merger of two separate commands. The following

```
$ odbcomp MYDB.ddl
$ create_odbgldue MYDB
```

is an equivalent (or a short-hand) notation for

```
% newodb MYDB
```

In fact the command `newodb` is the preferred way to access permanent databases, too. So if you need to deal with two or more databases, you should still use the `newodb`-command. There can be a practical situation to extract from usually a large ECMA-database (i.e. permanent database) data into a much smaller database `MYDB` in order to create a more manageable "subset" or work-database. Yet your ODB-application needs to be aware of that. The command dialogue is simply:

```
$ newodb ECMA.sch MYDB.ddl
# Queries associated with database MYDB
$ odbcomp -lMYDB query1.sql query2.sql
# A query associated with database ECMA
$ odbcomp -lECMA extract.sql
$ odbf90 myprog.F90 -lECMA -lMYDB -o myprog.x
```
Note that we used ECMA.sch, not ECMA.ddl-file for data layout file of the permanent database, since we assume that this database exists (i.e. database files accompanied its metadata). Actually, we may not have a trace (clue) of the original ECMA.ddl-file at all\(^\text{18}\).

Note also that we compiled two sets of queries: one for database ECMA and the other for MYDB. And then in the linking of the ODB-application we supply both database libraries. Note that the single command

\[
\text{% newodb ECMA.sch MYDB.ddl}
\]

could have been replaced with the following (now three) lines:

\[
\begin{align*}
\$ \text{odbcomp ECMA.sch} \\
\$ \text{odbcomp MYDB.ddl} \\
\$ \text{create_odbglue ECMA MYDB}
\end{align*}
\]

Naturally the one-liner looks more appealing.

### 9.3 Adding new databases in permanent basis

It maybe necessary to create a new database layout and make it a permanent database in the same manner as database ECMA. Lets call the new database DB. This is what you need to do, if you are under the ECMWF's ClearCase system\(^\text{19}\):

1. Go to /ccvobs/odb:
   \[
   \text{% cd /ccvobs/odb}
   \]

2. Create new (ClearCase) directory for database DB:
   \[
   \text{% ct mkdir ddl.DB}
   \]

3. Change (ClearCase) protections:
   \[
   \text{% ct protect -chown rdx -chgrp rd -chmod 555 ddl.DB}
   \]

After this you must create data definition layout file (DB.ddl) under /ccvobs/odb/ddl.DB and SQL-files still under generic place /ccvobs/odb/ddl. See more chapter 8.

\(^{18}\)On some very old ODB-databases, like PREODB even its PREODB.sch does not exist, but we can always re-create it from the data dictionary file PREODB.dd using utility dd2ddl.x

\(^{19}\)if not, then replace /ccvobs with $SRC
9.3 Adding new databases in permanent basis

Figure 22: Duplicate checking with PREODB, part#1

SYNOP, TEMPILOT, AIREP, DKBU, EDRAS, SETDB and SATEM observations
for a specific analysis data and time

Use station id?  No, AIREP, EDRAS, SETDB, SATEM

Yes (SYNOP, TEMPILOT, AIREP)

For each record/obs extract:

Station id, obs subtype, lat/lon

Station id = "_...99999" ?

Yes

Convert station id "99999" to a code based on lat/lon which is recognized
by assimilation system (e.g. 20.04, 20E = 02001200)

3 numeric digits

station id ?

Yes

Use WMO ?

Yes (SYNOP and TEMPILOT)

Reference WMO station id = "...WMO00"
Where _ is blank and WMO WMO block number

Check regions AND
SYNOP AND (subtype 1 OR 3) OR
TEMPILOT AND (subtype 91 OR 101)?

Yes
Figure 23: Duplicate checking with PREODB, part#2
10 ODB applications

Apart from user ODB-applications, there are a few ODB-applications that are capable of retrieving data automatically from ODB, or being able to create a database from user input (either from BUFR or text files). In addition there is a test version of an ODB-to-NetCDF translator.

10.1 BUFR2ODB

This decodes observations in BUFR-format and translates them into ODB databases ECMA or ECMASCR. BUFR-messages are expanded from the Input_file and required parameters are written into ODB format. Usage of the command `bufr2odb` is as follows:

```
bufr2odb -i input_file [-p add_pool]
            [-s npools] [-n] [-d thinning] [-l dbname]
```

The options denote:

- `-i input_file` Input BUFR-file prefix. A suffix ".poolno" will be added, so the actual BUFR-files to be opened will be `input_file.1`, `input_file.2`, etc.
- `-p add_pool` Add `add_pool` number of new pools on the top of existing. Default is 0.
- `-n` New database will be created regardless whether old one already exists.
-s npools  Number of pools required for new database.
-d thinning Rudimentary thinning. By default the thinning-parameter is set to 1 i.e. every observational position will be included. However, if thinning > 1, only every thinning’th spot will be passed to database.
-l dbname  Database name. By default it is ECMA, but it also could be ECMASCR.

The following environment variables are honoured:

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_MESS</td>
<td>No. of BUFR messages to process</td>
<td>-1</td>
</tr>
<tr>
<td>ODB_IO_METHOD</td>
<td>Choose I/O-method : 1 or 4</td>
<td>1</td>
</tr>
</tbody>
</table>

This program only loads the required (known) BUFR-subtypes. Valid BUFR subtype to be handled are:

<table>
<thead>
<tr>
<th>BUFR subtype</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SYNOP LAND</td>
</tr>
<tr>
<td>3</td>
<td>SYNOP LAND (AUTOMATIC)</td>
</tr>
<tr>
<td>9</td>
<td>SYNOP SHIP (ABBREVIATED)</td>
</tr>
<tr>
<td>11</td>
<td>SYNOP SHIP</td>
</tr>
<tr>
<td>13</td>
<td>SYNOP SHIP (AUTOMATIC)</td>
</tr>
<tr>
<td>19</td>
<td>SYNOP SHIP (REDUCED)</td>
</tr>
<tr>
<td>21</td>
<td>SURFACE BUOY</td>
</tr>
<tr>
<td>22</td>
<td>SURFACE BATHY</td>
</tr>
<tr>
<td>23</td>
<td>SURFACE TESAC</td>
</tr>
<tr>
<td>51</td>
<td>TOVS 120 KM RESOLUTION</td>
</tr>
<tr>
<td>52</td>
<td>TOVS 80 KM RESOLUTION</td>
</tr>
<tr>
<td>53</td>
<td>RTOVS</td>
</tr>
<tr>
<td>54</td>
<td>TOVS1C( CALIBRATED IN ECMWF)</td>
</tr>
<tr>
<td>55</td>
<td>ATOVS ( CALIBRATED IN UK MET OFFICE)</td>
</tr>
<tr>
<td>57</td>
<td>AIRS</td>
</tr>
<tr>
<td>65</td>
<td>SATEM</td>
</tr>
<tr>
<td>75</td>
<td>DMSP SATEM</td>
</tr>
<tr>
<td>82</td>
<td>SATOB WINDS</td>
</tr>
<tr>
<td>83</td>
<td>SATOB WINDS</td>
</tr>
<tr>
<td>87</td>
<td>HIGH DENSITY WINDS</td>
</tr>
<tr>
<td>89</td>
<td>GEOSTATIONARY RADIANCES</td>
</tr>
<tr>
<td>91</td>
<td>PILOT LAND</td>
</tr>
</tbody>
</table>
10.2 ODB2BUFR

Figure 25: ODB/ECMA to feedback-BUFR translation procedure

A program to read ECMA (or ECMASCRIPT) ODB-database and create BUFR feedback. This program creates new style analysis feedback files. There are four files created with the following file extensions:
.main containing original bufr observations and summary quality control
.qc containing analysis variables and quality control information
.fos containing analysis variables and first order statistics information
.dep containing analysis variables and anlysis departures information

A suffix ".poolno" will be added at the end of these extensions. Usage of the command 
odb2bufr is as follows:

```
```

The options denote:

- `-o output_file` Output BUFR-file prefix. A suffix ".poolno" will be added, so the actual (series of) BUFR-files written will be `output_file.main.1, output_file.qc.1, output_file.fos.1, output_file.dep.1, output_file.main.2, output_file.qc.2, output_file.fos.2, output_file.dep.2`, etc.
- `-l dbname` Database name. By default it is ECMA, but it also could be EC-MASCR.
- `-u nupdates` Number of updates. IFS scripts normally are set from environment variable $MXUP_TRAJ.
- `-T time_window` Time window (in hours) to which apply the BUFR-feedback creation. By default = 6. Time window starts from given `basetime`.
- `-B basetime` Basetime for time-windowed BUFR-feedback creation. Must be in format YYYYMMDDHH, e.g. for 12UTC on 1st of April 2003 basetime would be 2003040112. If not given, BUFR-feedback will be created for all data found in the ODB-database, which may/may not be against archive policy.
- `-I inc_subtype` Only these subtypes will be processed. Option can/should appear multiple times. Value for `inc_subtype` should generally be between 1 and 255 (both inclusive), but value 0 ensures that all subtypes not in exclusion list via `-E`-option will be processed.
- `-E exc_subtype` These subtypes will not be processed unless subsequent `-I`-option overrides. Value for `exc_subtype` should be between 1 and 255.

The same subtypes are supported as in `bufr2odb`. The following environment variables are honoured:
### 10.3 ODB-viewer

The ODB-viewer was originally designed to be just a debugging tool to cross-check that database contains desired data items. However, it has evolved into a more mature product, which enables a textual report to be produced for each data query. And if latitude and longitude information is present, then data coverage plot can be produced (using Magics software).

To start ODB-viewer do the following (after typing "use odb"):

```bash
$ cd CCMA
$ odbviewer -f CCMA.sch CCMA
```

When the prompt comes, you just type a view name (say *myview* – without any suffix). An editor session (using your preferred editor through the `ODB_EDITOR` environment variable) opens a window to a file `myview.sql` for you to edit. There you specify column and table names and `WHERE`-condition to extract data. Come out from editor and ODB-viewer will actually compile and execute the query. A new window will be opened and a textual result of your query appears. From that window, by choosing *Plot* and *Coverage plot* you can visualize the data coverage.

The next thing is to modify your query or launch another query. And the same thing happens over again. Pressing ctrl-D (end of file) or typing a period-character (.) terminates ODB-viewer session.

#### 10.4 simulobs2odb

In the latest ODB version there is a prototype version for loading a database directly from a text file. This can be a useful option when developing software or loading own databases and BUFR-definitions (for example) are not yet fixed.

This prototype has already been used successfully in migrating radio-occultation observations into the IFS code (by Sean Healy, ECMWF) before their BUFR-layout was even decided.

---

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUMMARY</td>
<td>Produce <code>.main-part</code></td>
<td>true</td>
</tr>
<tr>
<td>QC</td>
<td>Produce <code>.qc-part</code></td>
<td>true</td>
</tr>
<tr>
<td>FOS</td>
<td>Produce <code>.fos-part</code></td>
<td>true</td>
</tr>
<tr>
<td>DEP</td>
<td>Produce <code>.dep-part</code></td>
<td>true</td>
</tr>
<tr>
<td>MXUP_TRAJ</td>
<td>Maximum no. of updates</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Another prototype has also been tested. It allows a pre-compiled SQL data query to extract data from a database and write it into NetCDF format complying with specifications from ...
10.6 odb2mysql

Similar to the odb2netcdf-program, it is possible to extract data from ODB into a popular relational MySQL-database. This software is still under development.
A  Environment variables

This appendix describes every environment variable used directly or indirectly by the ODB software.

ODB-specific environment variables

Environment variables listed here are given in alphabetical order.

**ODB_CATCH_SIGNALS**

It is possible to catch error conditions through the ODB software by setting this environment variable to 1. This means, that when some unexpected memory violation occurs, the ODB system tries to catch signals sent by the Unix-system and produce a traceback in order to pin-point the file name and line number, where the error occurred.

See also ODB_ERRTRA (see p. 191) and ODB_TRACE_PROC (see p. 195).

This environment variable has become obsolete in CY28R2 due to introduction of Dr.Hook, which has a more robust way of catching errors.

**ODB_CCMA_CREATE_DIRECT**

This variable is used in connection with ECMWF’s 4D-Var-system. If set to 1, it tells the system to create directly the CCMA-database from the ECMA-database while IFS/Screening is still running, thus saving some important I/O-time. During development it is sometimes necessary to decouple the direct CCMA creation. Also some memory savings (in expense of higher I/O-cost) can be achieved, when this variable is set to 0 (which is also the default). See also routine $SRC/ifs/var/writeoba.F90.

**ODB_CCMA_IGNORE_TSLOTS**

When this variable is set to 1 during the CCMA-database creation, you can opt for ignoring 4D-Var timeslot information. This would produce a geographically load balanced database disregarding time information. See also routine $SRC/ifs/var/writeoba.F90.

**ODB_CCMA_OBS_RANDOMIZE**

When the CCMA-database is created, a geographical mapping of observational areas is performed. Each area contains approximately the same number of observations taking into account the cost of calculations. Sometimes it is hard to predict the cost and post-mapping (of observations) needs to performed. This environment variable allows randomization of the original "good" geographical mapping by redistributing observations in a round-robin fashion across all available data pools.

Set this to 1 if you want the randomization, otherwise set to 0. The default value is computer platform specific.
ODB_CONSIDER_TABLES

This variable is used only in connection with ODB_IO_METHOD (see p. 191). This environment variable sets ("considers") which database tables will actually be read in when ODB_open or ODB_close calls routine MSGPASS_OBSDATA (in $SRC/odb/lib/msgpass_obsdata.F90) with I/O-method equal to #4.

If not defined, then all TABLEs will be processed. This corresponds to condition:

```
$ export ODB_CONSIDER_TABLES="*"
```

Processing all TABLEs maybe an overkill for some post-processing applications, where just by looking at the SQL-query’s FROM-list one could determine the list of necessary TABLEs. If you know in advance that you will definitely should process only (say) TABLE hdr and body, then you need to give list of TABLEs as a slash (/) separated list as follows:

```
$ export ODB_CONSIDER_TABLES="/hdr/body/"
```

If you want to work with all tables, but not (say) bufr-table, you can specify this as follows:

```
$ export ODB_CONSIDER_TABLES="* except /bufr/"
```

ODB_CTX_DEBUG

Prints one line of information to stderr every time any subroutine in the ODB interface to IFS is entered or exited. This allows you to keep track of when and what kind of retrievals are executed and possible to understand erroneous situations.

This variable is by default turned off i.e. corresponds to value of 0. You can activate it for a particular MPI-task (1_m!m n$o(p&q&r) or for all MPI-tasks in one go by setting it to -1. If running with more than one OpenMP-threads, a line of information for each thread is printed.

See also ODB_ERRTRA (see p. 191) and ODB_TRACE_PROC (see p. 195).

ODB_DATAPATH

The generic root directory for database binary files. Usually the same as ODB_SRCPATH (see p. 194), but if different, then it indicates that binary files are possibly stored on a different disk file system to the metadata.

If not defined, defaults to the current directory. See also ODB_DATAPATH_$dbname (see p. 191).

ODB_DATAPATH_${dbname}

Database specific root directory for database binary files. When the location of the root directory is determined in ODB_open, this is the environment variable which will be interpreted first, just before ODB_DATAPATH (see p.190).

ODB_FLPCHECK

When this variable is set to 1, the ODB software will check that upon reading/unpacking or writing/packing every floating point column has got IEEE 754 conformant numbers. By default set to 0 i.e. turned OFF, and should stay like this, since the checking may increase execution time considerably. See also ODB_INTCHECK (see p.191).

ODB_ERRTRA

Attempt to activate subroutine/function call traceback writing in case of an error condition. The default value is 0 and no traceback will be written. Any other value activates traceback.

See also how to activate signal handler in ODB via ODB_CATCH_SIGNALS (see p.189). For more comprehensive debugging try ODB_TRACE_PROC (see p.195) or with IFS ODB_CTX_DEBUG (see p.190).

This environment variable has become obsolete in CY28R2 due to introduction of Dr.Hook, which has a more robust way of printing traces.

ODB_IGNORE_POOLMASKING

By default value is 1 and indicates that any poolmasking (i.e. selection of active pools on temporary basis) should be skipped. Set this option to 0 to allow poolmasking to be enabled. Note that the use of this option does not affect what has been defined through the variable ODB_PERMANENT_POOLMASK.

ODB_INTCHECK

When set to 1, this variable tries to guard against integer overflows. Can be very expensive when turned ON (=1). See also ODB_FLPCHECK (see p.191) for floating point checks. The default is 0.

ODB_IO_FILESIZE

Defines an approximate maximum allowed filesize in megabytes for concatenated HC32-format files. See also for I/O-method#4 in ODB_IO_METHOD (see p.191).

ODB_IO_METHOD
Chooses the format of ODB database binary files and the way data will be scheduled to the application program. This applies to all database TABLE-files i.e. you cannot choose an I/O-method on a per TABLE-basis.

ODB currently supports four I/O-methods, of which only two (method#1 and #4) are properly tested and considered robust. The I/O-methods are:

1. The default value. Creates one file per every TABLE on a pool basis. Uses the CMA I/O-routines with the standard C I/O-library (i.e. fopen, fread, fwrite and fclose).

2. The same as method#1, but after file opening bypasses the standard C I/O-library by using system I/O-routines (read and fwrite) directly. Not very well tested.

3. $qtar$ method, where an external ODB-specific utility (similar to $tar$) is invoked to store and extract data. One $QTAR$-file per pool is created i.e. all TABLEs will be saved into a single file on a pool basis. Reduces number of files considerably, but can be slow due to regular invocations of the external tool, and is also not very well tested.

4. This is the $HC32$-format file, where "HC" stands for horizontal concatenation and "32" stands for 32-bits and means that a single file in this scheme can span up to 4GB in size. This I/O-method is switched on as default in ECMWF scripts from IFS cycle CY26R1 onwards, since it offers substantial reduction in the number of TABLE-files produced. In this method each similar TABLE-file for a number of consecutive pools are essentially concatenated together to achieve the maximum configured filesize given via $ODB\_IO\_FILESIZE$ (see p.191).

The drawback of using this method is that all the required TABLEs (as specified via $ODB\_CONSIDER\_TABLES$ (see p.190) need to be loaded into the (distributed) memory during the $ODB\_open$. This may currently crash the post-processing application for a large database, unless sufficient number of parallel MPI-processors and/or enough memory is available. There is a work-around for this problem: utility $hcat$ can split the database TABLE-files by essentially transforming $HC32$-format files into format used by I/O-method#1. Other means to solve this memory problem are under development. One of the most prominent ways is to load data pools in chunks, when post-processing. This requires the use of environment variable $ODB\_PERMANENT\_POOLMASK$ (see p.193).

See also $ODB\_IO\_FILESIZE$ (see p.191), $ODB\_IO\_GRPSIZE$ (see p.??), $ODB\_WRITE\_EMPTY\_FILE$ (see p.196), $ODB\_PACKING$ (see p.192), $ODB\_PERMANENT\_POOLMASK$ (see p.193) and $ODB\_CONSIDER\_TABLES$ (see p.190).

**ODB\_PACKING**

By default ODB data files are internally packed. That is to say, the ODB packing algorithm is applied to every column of each TABLE before data is written to disk. There can
be some situations – especially during development – when one wants to turn packing off, or use just a fixed packing method.

This environment variable can have the following values

-1 The default i.e. packing is ON and every packable column gets packed before it is written to disk.

0 Turns packing OFF i.e. data turns out to disk as big-endian binary, where floating point number follow IEEE 754 standard.

n Packing method is set to a fixed number (n), which has to be between 1–255, with currently recognized packing methods being 1, 2, 5, 9, 11–19, 21–29. By using fixed packing method you will ask ALL packable columns to get compressed.

In addition to this environment variable, particular packing methods can be mapped to use alternative packing method. See more detail under ODB_PACKING_MAP_$n (see p.193).

ODB_PACKING_MAP_$n

This variable enables you to map an existing packing method to use some other recognized packing method or to turn OFF packing. When should this advanced feature be used? Suppose you do not trust some newly developed packing method and want to turn that off, but leave columns using other packing methods untouched. Then you can switch off the suspected packing method, or map it to use another, perhaps a more robust or one with better packing capabilities.

An example: Suppose you want to switch off a newly invented packing method#50 and map another packing method#60 to use the existing (robust) method#3. You would define:

```bash
$ export ODB_PACKING_MAP_50=0
$ export ODB_PACKING_MAP_60=3
```

See also ODB_PACKING (see p.192).

ODB_PERMANENT_POOLMASK

This option enables you to select a permanent set of pools to work with. This can be useful in post-processing a large amount of observations spread over many pools – and when the I/O-method had been set to 4. The use of this environment variable essentially limits the pool range(s) that an ODB application can access. You can choose a single pool, a pool range or many pool ranges. The following example illustrates different usages:
# Work with pool 10 only
$ export ODB_PERMANENT_POOLMASK=10

# Work with pools 5 to 7 only
$ export ODB_PERMANENT_POOLMASK="5-7"
$ export ODB_PERMANENT_POOLMASK="5,6,7"

# Work with pools 1, 5-7 and 10-12 only
$ export ODB_PERMANENT_POOLMASK="1,5-7,10-12"

See also ODB_IGNORE_POOLMASKING (see p. 191)

**ODB_REPRODUCIBLE_SEQNO**

This environment variable instructs ECMWF’s MERGEODB-process to generate unique sequence numbers for each observation across MPI-tasks. There are a few different approaches, of which number 2 is the default at ECMWF.

**ODB_SRCPATH**

Generic location of database metadata. If not defined, defaults to "." i.e. the current directory while in function ODB_open. See also ODB_DATAPATH (see p. 190).

**ODB_SRCPATH$_dbname**

The location of database metadata for the specific database $dbname$. When the location of metadata directory is determined in ODB_open, this is the first environment variable to be checked, and if it has not been defined, then ODB_SRCPATH (see p. 194) will be checked.

**ODB_STATIC_LINKING**

Enables (=1) or disables (=0) static linking mode. If static linking mode is OFF, then any database layout (even not a recognized/permanent one) gets loaded into the running executable during the runtime. Also SQL-queries can be supplied during the runtime, making the ODB-system very flexible.

The ODB was developed on SGI (Silicon Graphics) and dynamic linking worked (works) perfectly there for years, but unfortunately the mode doesn’t work as expected on IBM-systems, making the interactive ODB-system quite painful to use. This discrepancy will be addressed in the future by adding an interpretive-like query processing system, which would make ODB data requests independent of platform specific dynamic linking features.

Currently dynamic linking is a deprecated feature and should be avoided.

ODB_TEST_INDEX_RANGE

For every SQL-query while in ODB_select, the ODB builds a list of indices in order to gather data to user arrays while ODB_get is called. If you suspect that this index structure may have been corrupted, you can turn ON the range checking by setting this variable to 1. Activating this option will increase CPU-time consumption considerably.

ODB_TRACE_PROC

You can obtain very detailed information about ODB data transactions. You can choose to set this variable to 0 and you get no information (the default), or to n and you get information from MPI-task number n, or set it to -1 and every MPI-task produces information.

Information is written to files – one for each MPI-task – as defined through the variable ODB_TRACE_FILE (see p.195). Information can be flushed to this file at every ODB_TRACE_FLUSH_FREQ (see p.195) written lines.

ODB_TRACE_FILE

When ODB tracing is turned ON as described in ODB_TRACE_PROC (see p.195), you must define the output file for every MPI-task needing this information. This can be accomplished by setting this environment variable. In the following, the MPI-task is referred via %d and use of environment variables to be parsed by the ODB software, are accepted. For example:

```
$ export ODB_TRACE_PROC=-1
$ export ODB_TRACE_FILE=/specific/path/filename.%d
# or just for the first MPI-task
$ export ODB_TRACE_PROC=1
$ export ODB_TRACE_FILE=$WDIR/filename.%d
```

Trace transactions are written out by default for every line i.e. by using linebuffering mode. If you want less frequent output i.e. more efficient tracing, you can use variable ODB_TRACE_FLUSH_FREQ (see p.195).

ODB_TRACE_FLUSH_FREQ

Chooses the frequency of how often a number of trace transaction information will be written to ODB_TRACE_FILE (see p.195), when ODB_TRACE_PROC (see p.195) is ON. The default is to flush after every line, but you can override this; for example every 1000 lines:
$ export ODB_TRACE_FLUSH_FREQ=1000

**ODB_WRITE_EMPTY_FILES**

The default value is 0 and indicidates that when a TABLE does not contain any data rows, writing the file will be automatically switched off. When attempting to read such a non-existent file, the ODB system will automatically set the number of rows to zero.

If you still want to write zero rows of data, you can do it by setting this environment variable to 1. The file itself will not be completely empty, but will contain a few bytes of header information only.
B Miscellaneous observation type codes

This appendix lists all relevant BUFR and CMA type codes and satellite sensor identifications used in IFS.

B.1 BUFR observation types and its (ECMWF) subtypes

The following table summarizes the recognized BUFR subtypes (ECMWF local numbering) and whether they are currently fed into database ECMA (or ECMASCR) through the BUFR2ODB-procedure.

Table 41: BUFR-types and subtypes

<table>
<thead>
<tr>
<th>bufrtype@hdr</th>
<th>subtype@hdr</th>
<th>Supported by BUFR2ODB?</th>
</tr>
</thead>
<tbody>
<tr>
<td>No.</td>
<td>Name</td>
<td>No.</td>
</tr>
<tr>
<td>0</td>
<td>Land surface</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>140</td>
</tr>
<tr>
<td>1</td>
<td>Sea surface</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21</td>
</tr>
<tr>
<td></td>
<td></td>
<td>22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>23</td>
</tr>
<tr>
<td>2</td>
<td>Upper-air soundings</td>
<td>91</td>
</tr>
<tr>
<td></td>
<td></td>
<td>92</td>
</tr>
<tr>
<td></td>
<td></td>
<td>95</td>
</tr>
<tr>
<td></td>
<td></td>
<td>96</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
</tr>
<tr>
<td></td>
<td></td>
<td>102</td>
</tr>
<tr>
<td></td>
<td></td>
<td>103</td>
</tr>
<tr>
<td></td>
<td></td>
<td>106</td>
</tr>
<tr>
<td>3</td>
<td>Satellite soundings</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>53</td>
</tr>
<tr>
<td></td>
<td></td>
<td>54</td>
</tr>
<tr>
<td></td>
<td></td>
<td>55</td>
</tr>
</tbody>
</table>

\(^{20}\)European and Japanese wind profilers distinguished from each other by their station identifier

\(^{21}\)Phased out in CY28R2

continued on next page …
Table 41: BUFR-types and subtypes ...continued

<table>
<thead>
<tr>
<th>bufrtype@hdr</th>
<th>subtype@hdr</th>
<th>Supported by BUFR2ODB ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>No.</td>
<td>Name</td>
<td>No.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>Low-level temperature SATEM</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>High-level temperature SATEM</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>PWC SATEM</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>Merged SATEM</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>Low-level temperature TOVS</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>High-level temperature TOVS</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>PWC TOVS</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>Merged TOVS</td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>AIREP</td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>COLBA</td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>AMDAR</td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>ACARS</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>Temperature and wind</td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>Wind only</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>Surface temperature</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>Clouds temperature</td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>High-resolution VIS wind</td>
<td></td>
</tr>
<tr>
<td>87</td>
<td>AMV + quality control ??</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>ERS 1</td>
<td></td>
</tr>
<tr>
<td>122</td>
<td>ERS 2 Wind scatterometer</td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>SSM/I BT</td>
<td></td>
</tr>
<tr>
<td>253</td>
<td>PAOB</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>MODIS, GEOS, AIRS, ...</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>Geostationary radiances</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>Radiances</td>
<td></td>
</tr>
<tr>
<td>189</td>
<td>Geostationary radiances</td>
<td></td>
</tr>
</tbody>
</table>


B.2 CMA observation types and its codetypes

The following table summarizes the recognized CMA observation types and codetypes used in IFS, and also whether they are currently fed into database ECMA (or ECMASCR) through the BUFR2ODB-procedure.
### B.2 CMA observation types and its codetypes

Table 42: CMA observation types and codetypes

<table>
<thead>
<tr>
<th>obstype@hdr</th>
<th>codetype@hdr</th>
</tr>
</thead>
<tbody>
<tr>
<td>No.</td>
<td>Name</td>
</tr>
<tr>
<td>1</td>
<td>SYNOP</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>AIREP</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SATOB</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>DRIBU</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>TEMP</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>PILOT</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>SATEM</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*continued on next page...*
Table 42: CMA observation types and codetypes ... continued

<table>
<thead>
<tr>
<th>obstype@hdr</th>
<th>codetype@hdr</th>
</tr>
</thead>
<tbody>
<tr>
<td>No.</td>
<td>Name</td>
</tr>
<tr>
<td>202</td>
<td>GTS BUFR retrieved profiles/ clear radiances</td>
</tr>
<tr>
<td>210</td>
<td>ATOVS</td>
</tr>
<tr>
<td>211</td>
<td>RTOVS</td>
</tr>
<tr>
<td>212</td>
<td>TOVS</td>
</tr>
<tr>
<td>215</td>
<td>SSMI</td>
</tr>
<tr>
<td>8</td>
<td>PAOB</td>
</tr>
<tr>
<td>9</td>
<td>Scatterometer</td>
</tr>
</tbody>
</table>

8 | Scatterometer 1 |
122 | Scatterometer 2 |
210 | Scatterometer 3 |

B.3 Mapping between CMA and BUFR types

B.3.1 Mapping from CMA to BUFR types

Please note that going from CMA-world to BUFR-world, there is not always corresponding BUFR-codes available.

Table 43: CMA obs.types mapped into BUFR types

<table>
<thead>
<tr>
<th>CMA-world</th>
<th>BUFR-world</th>
</tr>
</thead>
<tbody>
<tr>
<td>obstype@hdr</td>
<td>codetype@hdr</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>24</td>
</tr>
<tr>
<td>2</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>141</td>
</tr>
<tr>
<td></td>
<td>142</td>
</tr>
<tr>
<td></td>
<td>144</td>
</tr>
<tr>
<td></td>
<td>145</td>
</tr>
<tr>
<td></td>
<td>241</td>
</tr>
</tbody>
</table>

continued on next page ...
### B.3 Mapping between CMA and BUFR types

Table 43: CMA obs.types mapped into BUFR types  

<table>
<thead>
<tr>
<th>CMA-world</th>
<th>BUFR-world</th>
</tr>
</thead>
<tbody>
<tr>
<td>obstype@hdr</td>
<td>codetype@hdr</td>
</tr>
<tr>
<td>3</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td>89</td>
</tr>
<tr>
<td></td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>188</td>
</tr>
<tr>
<td>4</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>165</td>
</tr>
<tr>
<td>5</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>135</td>
</tr>
<tr>
<td></td>
<td>137</td>
</tr>
<tr>
<td>6</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>34</td>
</tr>
<tr>
<td>7</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>184</td>
</tr>
<tr>
<td></td>
<td>185</td>
</tr>
<tr>
<td></td>
<td>186</td>
</tr>
<tr>
<td></td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>201</td>
</tr>
<tr>
<td></td>
<td>202</td>
</tr>
<tr>
<td></td>
<td>210</td>
</tr>
<tr>
<td></td>
<td>211</td>
</tr>
<tr>
<td></td>
<td>212</td>
</tr>
<tr>
<td></td>
<td>215</td>
</tr>
<tr>
<td>8</td>
<td>180</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>122</td>
</tr>
<tr>
<td></td>
<td>210</td>
</tr>
</tbody>
</table>

### B.3.2 Mapping from BUFR to CMA types
Table 44: BUFR obs.types mapped into CMA types

<table>
<thead>
<tr>
<th>BUFR-world</th>
<th>CMA-world</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>bufrtype@hdr</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>140</td>
</tr>
<tr>
<td>1</td>
<td>9, 11</td>
</tr>
<tr>
<td></td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>23</td>
</tr>
<tr>
<td>2</td>
<td>91</td>
</tr>
<tr>
<td></td>
<td>92</td>
</tr>
<tr>
<td></td>
<td>95</td>
</tr>
<tr>
<td></td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>103</td>
</tr>
<tr>
<td></td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>105</td>
</tr>
<tr>
<td></td>
<td>106</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>61–63, 65</td>
</tr>
<tr>
<td></td>
<td>71–73, 75</td>
</tr>
<tr>
<td>4</td>
<td>142</td>
</tr>
<tr>
<td></td>
<td>143</td>
</tr>
<tr>
<td></td>
<td>144</td>
</tr>
<tr>
<td></td>
<td>145</td>
</tr>
<tr>
<td>5</td>
<td>82–85</td>
</tr>
<tr>
<td></td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>87</td>
</tr>
<tr>
<td></td>
<td>89</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>122</td>
</tr>
<tr>
<td></td>
<td>127</td>
</tr>
</tbody>
</table>

*continued on next page...*
Table 44: BUFR obs.types mapped into CMA types ... continued

<table>
<thead>
<tr>
<th>BUFR-world</th>
<th>CMA-world</th>
</tr>
</thead>
<tbody>
<tr>
<td>bufrtype@hdr</td>
<td>subtype@hdr</td>
</tr>
<tr>
<td>253</td>
<td>164</td>
</tr>
</tbody>
</table>

B.4 Satellite identification codes

See also file $SRC/ifs/module/yomtvradF90$.

B.4.1 Satellite sensor

Table 45: SATOB computational methods

<table>
<thead>
<tr>
<th>Variable label</th>
<th>Index</th>
<th>Name (CHSENSOR)</th>
<th>Channel offset</th>
<th>Channel count</th>
<th>By default perform cloudy RTCALC</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSENSOR_HIRS</td>
<td>0</td>
<td>HIRS</td>
<td>0</td>
<td>20</td>
<td>YES</td>
</tr>
<tr>
<td>MSENSOR_MSU</td>
<td>1</td>
<td>MSU</td>
<td>20</td>
<td>4</td>
<td>YES</td>
</tr>
<tr>
<td>MSENSOR_SSU</td>
<td>2</td>
<td>SSU</td>
<td>24</td>
<td>3</td>
<td>NO</td>
</tr>
<tr>
<td>MSENSOR_AMSUA</td>
<td>3</td>
<td>AMSUA</td>
<td>27</td>
<td>15</td>
<td>YES</td>
</tr>
<tr>
<td>MSENSOR_AMSUB</td>
<td>4</td>
<td>AMSUB</td>
<td>42</td>
<td>5</td>
<td>NO</td>
</tr>
<tr>
<td>MSENSOR_SSMI</td>
<td>6</td>
<td>SSMI</td>
<td>0</td>
<td>7</td>
<td>YES</td>
</tr>
<tr>
<td>MSENSOR_VTPR1</td>
<td>7</td>
<td>VTPR1</td>
<td>0</td>
<td>8</td>
<td>YES</td>
</tr>
<tr>
<td>MSENSOR_VTPR2</td>
<td>8</td>
<td>VTPR2</td>
<td>8</td>
<td>8</td>
<td>YES</td>
</tr>
<tr>
<td>MSENSOR_AIRS</td>
<td>11</td>
<td>AIRS</td>
<td>0</td>
<td>2378</td>
<td>YES</td>
</tr>
<tr>
<td>MSENSOR_METEOSAT</td>
<td>20</td>
<td>METEOSAT</td>
<td>0</td>
<td>2</td>
<td>YES</td>
</tr>
<tr>
<td>MSENSOR_MSG</td>
<td>21</td>
<td>MSG</td>
<td>0</td>
<td>8</td>
<td>NO</td>
</tr>
<tr>
<td>MSENSOR_GOESIMG</td>
<td>22</td>
<td>GOESIMG</td>
<td>0</td>
<td>4</td>
<td>NO</td>
</tr>
</tbody>
</table>
B.4.2 SATOB computational methods

Table 46: SATOB computational methods

<table>
<thead>
<tr>
<th>Variable label</th>
<th>Index</th>
<th>Name (CHMETHOD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMETHOD_WVCL</td>
<td>0</td>
<td>WVCL</td>
</tr>
<tr>
<td>MMETHOD_IR</td>
<td>1</td>
<td>IR</td>
</tr>
<tr>
<td>MMETHOD_VIS</td>
<td>2</td>
<td>VIS</td>
</tr>
<tr>
<td>MMETHOD_WVMIX</td>
<td>3</td>
<td>WVMIX</td>
</tr>
<tr>
<td>MMETHOD_COM_SPEC</td>
<td>4</td>
<td>COM_CHAN</td>
</tr>
<tr>
<td>MMETHOD_WVMW</td>
<td>5</td>
<td>WVMW</td>
</tr>
</tbody>
</table>

B.4.3 Geostationary satellite names

Table 47: Geostationary satellite names

<table>
<thead>
<tr>
<th>Variable label</th>
<th>Index</th>
<th>Name (CHGEOSATN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MGEOSATN_METEOSAT</td>
<td>0</td>
<td>METEOSAT</td>
</tr>
<tr>
<td>MGEOSATN_GMS</td>
<td>1</td>
<td>GMS</td>
</tr>
<tr>
<td>MGEOSATN_Goes</td>
<td>2</td>
<td>GOES</td>
</tr>
<tr>
<td>MGEOSATN_MSG</td>
<td>3</td>
<td>MSG</td>
</tr>
<tr>
<td>MGEOSATN_INDSAT</td>
<td>4</td>
<td>INDSAT</td>
</tr>
<tr>
<td>MGEOSATN_MODIS</td>
<td>7</td>
<td>MODIS</td>
</tr>
</tbody>
</table>

B.4.4 Satellite series

Table 48: Satellite series

<table>
<thead>
<tr>
<th>Variable label</th>
<th>Index</th>
<th>Name (CHSERIES)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSERIES_NOAA</td>
<td>1</td>
<td>NOAA</td>
</tr>
<tr>
<td>MSERIES_DMSP</td>
<td>2</td>
<td>DMSP</td>
</tr>
<tr>
<td>MSERIES_METEOSAT</td>
<td>3</td>
<td>METEOSAT</td>
</tr>
<tr>
<td>MSERIES_Goes</td>
<td>4</td>
<td>GOES</td>
</tr>
<tr>
<td>MSERIES_GMS</td>
<td>5</td>
<td>GMS</td>
</tr>
<tr>
<td>MSERIES_NASA</td>
<td>6</td>
<td>NASA</td>
</tr>
</tbody>
</table>
This appendix describes all known observation variable numbers (varnobody) and how they map into various variable number mnemonics found in ODB file $SRC/odb/ddl/varno.h (see also D.1) and to IFS-variable NVNUMB (initialized in $SRC/ifs/setup/suvnmb.F90).

Table 49: Observation variable numbers

<table>
<thead>
<tr>
<th>NVNUMB index</th>
<th>varnobody</th>
<th>ODB mnemonic</th>
<th>Description</th>
<th>Units [or WMO code]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>$u$</td>
<td>u-component of wind</td>
<td>m/s</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>$v$</td>
<td>v-component of wind</td>
<td>m/s</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>$z$</td>
<td>Geopotential (Z)</td>
<td>(m/s)^2</td>
</tr>
<tr>
<td>4</td>
<td>57</td>
<td>$dz$</td>
<td>Thickness (DZ)</td>
<td>(m/s)^2</td>
</tr>
<tr>
<td>5</td>
<td>29</td>
<td>$rh$</td>
<td>Relative humidity (RH)</td>
<td>Numeric</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>$spwc$</td>
<td>Precipitable water content (PWC)</td>
<td>kg/m^2</td>
</tr>
<tr>
<td>7</td>
<td>58</td>
<td>$rh2m$</td>
<td>2metre relative humidity (RH_{2m})</td>
<td>Numeric</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>$t$</td>
<td>Upper air temperature (T)</td>
<td>K</td>
</tr>
<tr>
<td>9</td>
<td>59</td>
<td>$std$</td>
<td>Upper air dew point (T_d)</td>
<td>K</td>
</tr>
<tr>
<td>10</td>
<td>39</td>
<td>$t2m$</td>
<td>2metre temperature (T_{2m})</td>
<td>K</td>
</tr>
<tr>
<td>11</td>
<td>40</td>
<td>$std2m$</td>
<td>2metre dew point (T_{d2m})</td>
<td>K</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
<td>$sts$</td>
<td>Surface temperature (T_s)</td>
<td>K</td>
</tr>
<tr>
<td>13</td>
<td>30</td>
<td>$ptend$</td>
<td>Pressure tendency (p_t)</td>
<td>Pa/3h</td>
</tr>
<tr>
<td>14</td>
<td>60</td>
<td>$sw$</td>
<td>Past weather (W)</td>
<td>[4561]</td>
</tr>
<tr>
<td>15</td>
<td>61</td>
<td>$ww$</td>
<td>Present weather (WW)</td>
<td>[4677]</td>
</tr>
<tr>
<td>16</td>
<td>62</td>
<td>$vv$</td>
<td>Visibility (V)</td>
<td>[4300]</td>
</tr>
<tr>
<td>17</td>
<td>63</td>
<td>$sch$</td>
<td>Type of high clouds (C_H)</td>
<td>[0509]</td>
</tr>
<tr>
<td>18</td>
<td>64</td>
<td>$cm$</td>
<td>Type of middle clouds (C_M)</td>
<td>[0515]</td>
</tr>
<tr>
<td>19</td>
<td>65</td>
<td>$cl$</td>
<td>Type of low clouds (C_L)</td>
<td>[0513]</td>
</tr>
<tr>
<td>20</td>
<td>66</td>
<td>$nh$</td>
<td>Cloud base height (N_h)</td>
<td>m</td>
</tr>
<tr>
<td>21</td>
<td>67</td>
<td>$nn$</td>
<td>Low cloud amount (N)</td>
<td>[2700]</td>
</tr>
<tr>
<td>22</td>
<td>68</td>
<td>$shhs$</td>
<td>Additional cloud group height (h_ih_s)</td>
<td>m</td>
</tr>
<tr>
<td>23</td>
<td>69</td>
<td>$sc$</td>
<td>Additional cloud group type(C)</td>
<td>[0500]</td>
</tr>
<tr>
<td>24</td>
<td>70</td>
<td>$ns$</td>
<td>Additional cloud group amount (N_i)</td>
<td>[27000]</td>
</tr>
<tr>
<td>25</td>
<td>71</td>
<td>$sdepth$</td>
<td>Snow depth (S_d)</td>
<td>m</td>
</tr>
<tr>
<td>26</td>
<td>72</td>
<td>$se$</td>
<td>State of ground (E)</td>
<td>[0901]</td>
</tr>
<tr>
<td>27</td>
<td>73</td>
<td>$stgtg$</td>
<td>Ground temperature (T_sT_g)</td>
<td>K</td>
</tr>
<tr>
<td>28</td>
<td>74</td>
<td>$spsp1$</td>
<td>Special phenomena#1 (S_pS_p)</td>
<td>[3778]</td>
</tr>
</tbody>
</table>

continued on next page...
Table 49: Observation variable numbers  ...continued

<table>
<thead>
<tr>
<th>NVNUMB index</th>
<th>varno@body mnemonic</th>
<th>ODB mnemonic</th>
<th>Description</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>75</td>
<td>$s_{psp2}$</td>
<td>Special phenomena#2 ($s_{p}\cdot s_{p}$)</td>
<td>[3778]</td>
</tr>
<tr>
<td>30</td>
<td>76</td>
<td>$s_{r}$</td>
<td>Ice code type ($R_{s}$)</td>
<td>[3551]</td>
</tr>
<tr>
<td>31</td>
<td>77</td>
<td>$s_{ses}$</td>
<td>Ice thickness ($E_{1}\cdot E_{2}$)</td>
<td>[1751]</td>
</tr>
<tr>
<td>32</td>
<td>78</td>
<td>$s_{is}$</td>
<td>Ice ($I_{s}$)</td>
<td>[1751]</td>
</tr>
<tr>
<td>33</td>
<td>79</td>
<td>$s_{trr}$</td>
<td>Time period of rain information ($t_{r}\cdot t_{r}$)</td>
<td>h</td>
</tr>
<tr>
<td>34</td>
<td>80</td>
<td>$s_{rr}$</td>
<td>6h rain amount</td>
<td>kg/m²</td>
</tr>
<tr>
<td>35</td>
<td>81</td>
<td>$s_{jj}$</td>
<td>Maximum temperature ($J_{J}$)</td>
<td>K</td>
</tr>
<tr>
<td>36</td>
<td>82</td>
<td>$s_{vs}$</td>
<td>Ship speed ($V_{s}$)</td>
<td>m/s</td>
</tr>
<tr>
<td>37</td>
<td>83</td>
<td>$s_{ds}$</td>
<td>Ship direction ($D_{s}$)</td>
<td>Degree</td>
</tr>
<tr>
<td>38</td>
<td>84</td>
<td>$s_{whw}$</td>
<td>Wave height ($H_{w}\cdot H_{w}$)</td>
<td>m</td>
</tr>
<tr>
<td>39</td>
<td>85</td>
<td>$s_{wpw}$</td>
<td>Wave period ($P_{w}\cdot P_{w}$)</td>
<td>s</td>
</tr>
<tr>
<td>40</td>
<td>86</td>
<td>$s_{dwdw}$</td>
<td>Wave direction ($D_{w}\cdot D_{w}$)</td>
<td>Degree</td>
</tr>
<tr>
<td>41</td>
<td>87</td>
<td>$s_{gclg}$</td>
<td>General cloud group</td>
<td>[20012]</td>
</tr>
<tr>
<td>42</td>
<td>88</td>
<td>$s_{rhlc}$</td>
<td>Relative humidity from low clouds</td>
<td>Numeric</td>
</tr>
<tr>
<td>43</td>
<td>89</td>
<td>$s_{rhmc}$</td>
<td>Relative humidity from middle clouds</td>
<td>Numeric</td>
</tr>
<tr>
<td>44</td>
<td>90</td>
<td>$s_{rhhc}$</td>
<td>Relative humidity from high clouds</td>
<td>Numeric</td>
</tr>
<tr>
<td>45</td>
<td>91</td>
<td>$s_{n}$</td>
<td>Total cloud amount</td>
<td>[20011]</td>
</tr>
<tr>
<td>46</td>
<td>92</td>
<td>$s_{sfall}$</td>
<td>6h snowfall</td>
<td>m</td>
</tr>
<tr>
<td>47</td>
<td>110</td>
<td>$s_{ps}$</td>
<td>Surface pressure ($p_{s}$)</td>
<td>Pa</td>
</tr>
<tr>
<td>48</td>
<td>111</td>
<td>$s_{dd}$</td>
<td>Wind direction ($D_{d}$)</td>
<td>Degree</td>
</tr>
<tr>
<td>49</td>
<td>112</td>
<td>$s_{ff}$</td>
<td>Wind force</td>
<td>m/s</td>
</tr>
<tr>
<td>50</td>
<td>119</td>
<td>$s_{rawbt}$</td>
<td>Brightness temperature ($T_{b}$)</td>
<td>K</td>
</tr>
<tr>
<td>51</td>
<td>120</td>
<td>$s_{rawra}$</td>
<td>Raw radiance</td>
<td>K</td>
</tr>
<tr>
<td>52</td>
<td>121</td>
<td>$s_{satcl}$</td>
<td>Cloud amount from satellite</td>
<td>%</td>
</tr>
<tr>
<td>53</td>
<td>122</td>
<td>$s_{scatss}$</td>
<td>Backscatter ($\sigma_{0}$)</td>
<td>dB</td>
</tr>
<tr>
<td>54</td>
<td>5</td>
<td>$s_{du}$</td>
<td>Wind shear u-component ($\frac{\partial u}{\partial z}$)</td>
<td>1/s</td>
</tr>
<tr>
<td>55</td>
<td>6</td>
<td>$s_{dv}$</td>
<td>Wind shear v-component ($\frac{\partial v}{\partial z}$)</td>
<td>1/s</td>
</tr>
<tr>
<td>56</td>
<td>41</td>
<td>$s_{u10m}$</td>
<td>10metre u-component of wind ($u_{10m}$)</td>
<td>m/s</td>
</tr>
<tr>
<td>57</td>
<td>42</td>
<td>$s_{v10m}$</td>
<td>10metre v-component of wind ($v_{10m}$)</td>
<td>m/s</td>
</tr>
<tr>
<td>58</td>
<td>19</td>
<td>$s_{rhlay}$</td>
<td>Layer relative humidity</td>
<td>Numeric</td>
</tr>
<tr>
<td>59</td>
<td>200</td>
<td>$s_{auxil}$</td>
<td>Auxiliary variable</td>
<td>Numeric</td>
</tr>
</tbody>
</table>

continued on next page ...
Table 49: Observation variable numbers ...continued

<table>
<thead>
<tr>
<th>NVNUMB</th>
<th>varno@body</th>
<th>ODB mnemonic</th>
<th>Description</th>
<th>Units [or WMO code]</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>123</td>
<td>scllqw</td>
<td>Cloud liquid water ($Q_l$)</td>
<td>kg/kg</td>
</tr>
<tr>
<td>61</td>
<td>124</td>
<td>scatdd</td>
<td>Ambiguous v-component</td>
<td>m/s</td>
</tr>
<tr>
<td>62</td>
<td>125</td>
<td>scatff</td>
<td>Ambiguous u-component</td>
<td>m/s</td>
</tr>
<tr>
<td>63</td>
<td>7</td>
<td>$q$</td>
<td>Specific humidity ($Q$)</td>
<td>kg/kg</td>
</tr>
<tr>
<td>64</td>
<td>126</td>
<td>scatwd</td>
<td>Ambiguous wind direction</td>
<td>Degree</td>
</tr>
<tr>
<td>65</td>
<td>127</td>
<td>scatws</td>
<td>Ambiguous wind speed</td>
<td>m/s</td>
</tr>
<tr>
<td>66</td>
<td>8</td>
<td>vsp</td>
<td>Vertical speed</td>
<td>m/s</td>
</tr>
<tr>
<td>67</td>
<td>56</td>
<td>vt</td>
<td>Virtual temperature ($T_v$)</td>
<td>K</td>
</tr>
<tr>
<td>68</td>
<td>130</td>
<td>so3lay</td>
<td>Ozone layer density</td>
<td>kg/m^2</td>
</tr>
<tr>
<td>69</td>
<td>156</td>
<td>height</td>
<td>Generic height</td>
<td>m</td>
</tr>
<tr>
<td>70</td>
<td>215</td>
<td></td>
<td>SSM/I 1DVAR multilevel variable</td>
<td></td>
</tr>
</tbody>
</table>

It is not very difficult to add new variable numbers. This can be accomplished as follows: ...
D Important files for reference

This appendix contains listings of some important files referred in this document.

D.1 Database layouts

$SRC/odb/ddl.CCMA/CCMA.ddl

```c
#define CCMA
#include "cma.h"
```

$SRC/odb/ddl.ECMA/ECMA.ddl

```c
#define ECMA
#include "cma.h"
```

$SRC/odb/ddl.ECMASCR/ECMASCR.ddl

```c
#define ECMASCR
#define ECMA
#include "cma.h"
```

$SRC/odb/ddl/cma.h

```c
// Data layout (schema) for CY28R2
// ECMASCR & ECMA table hierarchy:
// 1 desc
// 2 --> ddrs
// 3 --> poolmask
// 4 --> timeslot_index
// 5 | --> index
// 6 | --> hdr
// 7 | | --> sat
// 8 | | --> real
// 9 v v v v --> satob
// 10 | | | --> scatt
// 11 | | | --> scatt_body
// 12 | | --> ssmi
// 13 | | --> ssmi_body
// 14 | --> atovs
// 15 | --> atovs_pred
```
// Variables

SET $NMXUPD = 3; // Maximum number of updates supported with this layout
SET $NUMAUX = 2; // No. of auxiliary obvalue's per body; auxl --> aux[$NUMAUX]
SET $NUMTHBOX = 3; // No. of thinning boxes (see also ifs/module/yomdb.F90)

// Aligned tables (contain the same no. of rows when requested over the @LINK)
RESET ALIGN;
ALIGN(body,errstat,update[1:$NMXUPD],scatt_body,ssmi_body);
ALIGN(atovs,atovs_pred);

// @LINKs with maximum jump of one ("one-loopers")
// Rows in these tables have one-to-one correspondence over the @LINK
RESET ONELOOPER;
ONELOOPER(index,hdr);
ONELOOPER(hdr,sat);
ONELOOPER(sat,atovs,ssmi,scatt,satob,reo3);

// Define shared links (new option; not available through command line)
// (these aren't working properly ... yet (as of 29/8/2001 SS)
//SHAREDLINK(body,errstat,update[1:$NMXUPD]));
#include "mdi.h"
#include "obstype.h"
#include "varno.h"
#include "vertco_type.h"
#include "ppcode.h"
#include "sensor.h"

SET $BG = 1;
SET $ADJ = 2;

//
D.1 Database layouts

-- Type definitions

//CREATE TYPE obschar_t AS {
// codetype bit10,    // OBS. CODE TYPE
// instype bit10,    // INS. TYPE
// retrtype bit6,    // RETRIEVAL TYPE
// datagroup bit6,   // DATA PROCESSING GROUP
//};

CREATE TYPE report_rdbflag_t AS {
  lat_humon bit1,
  lat_QCsub bit1,
  lat_override bit1,
  lat_flag bit2,
  lat_HQC_flag bit1,
  lon_humon bit1,
  lon_QCsub bit1,
  lon_override bit1,
  lon_flag bit2,
  lon_HQC_flag bit1,
  date_humon bit1,
  date_QCsub bit1,
  date_override bit1,
  date_flag bit2,
  date_HQC_flag bit1,
  time_humon bit1,
  time_QCsub bit1,
  time_override bit1,
  time_flag bit2,
  time_HQC_flag bit1,
  stalt_humon bit1,
  stalt_QCsub bit1,
  stalt_override bit1,
  stalt_flag bit2,
  stalt_HQC_flag bit1,
};

CREATE TYPE status_t AS {
  active bit1, // ACTIVE FLAG
  passive bit1, // PASSIVE FLAG
  rejected bit1, // REJECTED FLAG
  blacklisted bit1, // BLACKLISTED
#ifdef ECMA
  // BLACKLIST FAILCODE BITS
  monthly bit1,
  constant bit1,
  experimental bit1,
  whitelist bit1,
#endif /* ECMA */
};

CREATE TYPE instrm_t = pkint;

CREATE TYPE datum_rdbflag_t AS {
  press_humon bit1,
  press_QCsub bit1,
  press_override bit1,
  press_flag bit2,
  press_HQC_flag bit1,
  press_judged_prev_an bit2,
  press_used_prev_an bit1,
};
CREATE TYPE datum_flag_t AS (
    final bit4, // FINAL FLAG
    fg bit4, // FIRST GUESS FLAG
    depar bit4, // DEPARTURE FLAG
    varQC bit4, // VARIATIONAL QUALITY FLAG
    blacklist bit4, // BLACKLIST FLAG
    ups bit1, // d'utilisation par analyse de pression de surface
    uvt bit1, // d'utilisation par analyse de vent et temperature
    uhu bit1, // d'utilisation par analyse d'humidite
    ut2 bit1, // d'utilisation par analyse de temperature a 2m
    uh2 bit1, // d'utilisation par analyse d'humidite a 2m
    uvl bit1, // d'utilisation par analyse de vent a 10m
    urr bit1, // d'utilisation par analyse de precipitations
    usn bit1, // d'utilisation par analyse de neige
    usst bit1, // d'utilisation par analyse de temperature de surface de la mer
);

#ifndef ECMA
CREATE TYPE level_t AS (
    id bit9, // PILOT LEV. ID.
    maxwind bit1, // MAX WIND LEVEL
    tropopause bit1, // TROPOPAUSE
    D_part bit1, // D PART
    C_part bit1, // C PART
    B_part bit1, // B PART
    A_part bit1, // A PART
    surface bit1, // SURFACE LEVEL
    signwind bit1, // SIGNIFICANT WIND LEVEL
    signtemp bit1, // SIGNIFICANT TEMPR. LEVEL
);
#endif /* ECMA */

CREATE TYPE report_event1_t AS (
    no_data bit1,
    all_rejected bit1,
    bad_practice bit1,
    rdb_rejected bit1,
    rdb_activated bit1,
    whitelist_activated bit1,
    horipos_outrange bit1,
    vertpos_outrange bit1,
    time_outrange bit1,
    redundant bit1,
    land bit1,
    sea bit1,
    stalt_missing bit1,
    modsurf_stalt_distance bit1,
    namelist_rejected bit1,
    QC_failed bit1,
);
#ifdef ECMA
CREATE TYPE report_event2_t = pk1int;

CREATE TYPE report_blacklist_t AS (
  obstype bit1,
  statid bit1,
  codetype bit1,
  instype bit1,
  date bit1,
  time bit1,
  lat bit1,
  lon bit1,
  stalt bit1,
  scanpos bit1,
  retrtype bit1,
  QI_1 bit1,
  QI_2 bit1,
  QI_3 bit1,
  modoro bit1,
  lsmask bit1,
  rlsmask bit1,
  rlmask bit1,
  modPS bit1,
  modTS bit1,
  modT2M bit1,
  modtop bit1,
  sensor bit1,
  fov bit1,
  satza bit1,
  andate bit1,
  antime bit1,
);
#endif /* ECMA */

CREATE TYPE datum_event1_t AS (
  vertco_missing bit1,
  obsvalue_missing bit1,
  fg_missing bit1,
  rdb_rejected bit1,
  rdb_activated bit1,
  whitelist_activated bit1,
  bad_practice bit1,
  vertpos_outrange bit1,
  reflevel_outrange bit1,
  fg2big bit1,
  depar2big bit1,
  obs_error2big bit1,
  datum_redundant bit1,
  not_analysis_varno bit1,
  land bit1,
  sea bit1,
  not_analysis_varno bit1,
  duplicate bit1,
  levels2many bit1,
  multilevel_check bit1,
  vertco_consistency bit1,
  vertco_type_changed bit1,
  namelist_rejected bit1,
  combined_flagging bit1,
  report_rejected bit1,
  varQC_performed bit1,
  contam_cld_flag bit1, // cloud contamination
contam_rain_flag bit1, // rain contamination
);

#ifdef ECMA
CREATE TYPE datum_event2_t = pk1int;
CREATE TYPE datum_blacklist_t AS {
   varno bit1,
   vertco_type bit1,
   press bit1,
   press_rl bit1,
   ppcode bit1,
   obsvalue bit1,
   fg_depar bit1,
   obs_error bit1,
   fg_error bit1,
   winchan_depal bit1,
   obs_t bit1,
};
#else /* ECMA */
CREATE TYPE varno_presence_t AS {
   // presence of (any) ...
   other bit1, // bit#0 for non-categorized variable no.
   ps bit1, // 1 : surface pressure
   wind bit1, // 2 : wind-components
   t bit1, // 3 : temperature
   z bit1, // 4 : geopotential
   rh bit1, // 5 : relative humidity
   q bit1, // 6 : specific humidity
   obs2m bit1, // 7 : 2m observations
   snow bit1, // 8 : snow obs.
   rain bit1, // 9 : rain (liquid/solid)
   rawb bit1, // 10 : brightness temperature
   rawr bit1, // 11 : raw radiance
   ozone bit1, // 12 : ozone
   dz bit1, // 13 : thickness
   pwc bit1, // 14 : pwc
   time_period bit1, // 15 : time period (of rain etc.)
};
#endif /* ECMA */

CREATE TABLE desc AS {
   expver string,
   andate YYYYMMDD,
   antime HHMMSS,
   // creation date, time and created by whom (username)
   creadate YYYYMMDD,
   creatime HHMMSS,
   creaby string,
   // modification date, time and modified by whom (username)
   moddate YYYYMMDD,
   modtime HHMMSS,
   modby string,
   // max. no. of updates for this database (= $MXUP_TRAJ)
   mxup_traj pk1int,
D.1 Database layouts

```sql
CREATE TABLE ddrs AS (  
    ddrno pk1int,  
    wordno pk1int,  
    bulkdata pk9real,  
);

CREATE TABLE poolmask AS (  
    tslot pk1int,  
    obstype pk1int,  
    codetype pk1int,  
    sensor pk1int,  
    bufrtype pk1int,  
    subtype pk1int,  
    satinst pk1int,  
    poolno pk1int,  
    hdr_count pk1int,  
    body_count pk1int,  
    max_bodylen pk1int,  
);

CREATE TABLE timeslot_index AS ( // More direct access to time slot data  
    timeslot pk1int,  
    index @LINK,  
);

CREATE TABLE index AS (  
    target pk1int,  
    procid pk1int, // normally the same as pool number, too  
    tslot pk1int,  
    kset pk1int,  
    abnob pk1int,  
    mapomm pk1int,  
    hdr @LINK,  
);

CREATE TABLE hdr AS (  
    seqno pk1int, // OBSERVATION SEQUENCE  
    obstype pk1int, // OBSERVATION TYPE  
    // obschar OBSCHAR_t, // OBSERVATION CHARACTERISTICS  
    codetype pk1int, // OBSERVATION CODE TYPE  
    insttype pk1int, // OBSERVATION INSTRUMENT TYPE  
    retrtype pk1int, // OBSERVATION RETRIEVAL TYPE  
    areatype pk1int, // OBSERVATION AREA TYPE  
    zone pk9int, // Used by Meteo France  
    date YYYYMMDD, // OBS. DATE  
    time HHHHMMSS, // OBS. EXACT TIME  
    status STATUS_t, // REPORT'S STATUS  
    event1 REPORT_EVENT1_t, // REPORT'S EVENTS (PART 1)  
    sortbox pk1int, // SORTING BOX  
    sitdep pk1int, // site dependent  
    source string, // Source ID of obs. (CHARACTER*8) : Re-analysis purposes  
    statid string, // STATION ID (CHARACTER*8)  
    ident pk1int, // Integer equivalent of STATION ID (or zero)  
    lat pk9real, // LATITUDE
);  
```

```
lon pk9real, // LONGITUDE
stalt pk9real, // ALTITUDE
modoro pk9real, // MODEL'S OROGRAPHY
trlat pk9real, // TRANSFORMED LAT.
trlon pk9real, // TRANSFORMED LON.
instspec INSTRM_t, // INST. SPEC. WORD POSITIONS
anemoht pk9real, // HEIGHT OF ANEMOMETER
baroht pk9real, // HEIGHT OF BAROMETRE
sensor pk1int, // SATELLITE SENSOR INDICATOR
numlev pk1int, // No. of distinct pressure levels in bodies
numactiveb pk1int, // No. of active body entries (i.e. status.active@body == 1)
checksum pk9real, // A check sum of obsvalues where press and
varno_presence varno_presence_t, // Variable presence flag
sat @LINK,
body @LINK,
errstat @LINK,
update[1:$NMXUPD] @LINK,
rdbflag REPORT_RDBFLAG_t, // BUFFER subtype for reference
subtype pk1int, // BUFR-type
satinst pk1int, // Satellite sensor/instrument in BUFR/WMO-world
satname[2] string, // Satellite name (2 words, up to 16 bytes)
#ifdef ECMA
thinningkey[$NUMTHBOX] pk9real, // Thinning key
thinning_timekey pk9real, // Thinning time key
blacklist REPORT_BLACKLIST_t, // REPORT'S BLACKLIST EVENTS
event2 REPORT_EVENT2_t, // REPORT EVENTS (PART 2) WORD POS.
bufr @LINK, // Link to BUFR-data
subsetno pk9int, // Multisubset number in BUFR-msg (=0 for single subset)
station_type pk1int, // SYNOP/SHIPs (needed to find out if DRIBU)
sonde_type pk1int, // In order to do bias corr. of TEMPs
satem_instdata pk1int, // INS. DATA USED IN PROC. for old-SATEM
satem_dataproc pk1int, // DATA PROC. TECHNIQUE USED for old-SATEM
#endif /* ECMA */
);
#ifdef ECMA
CREATE TABLE bufr AS ( msg bufr, // Just a set of 4-byte sequences after each other )
);
#endif
CREATE TABLE sat AS ( satid pk1int, // Integer equivalent to statid@hdr or 0; same as ident@hdr
atovs @LINK,
reo3 @LINK,
satob @LINK,
#ifdef ECMA
scatt @LINK,
ssmi @LINK,
#endif /* ECMA */
);
CREATE TABLE reo3 AS ( instrument_type pk1int, // SATELLITE INSTRUMENT
product_type pk1int, // PRODUCT TYPE FOR RETRIEVED ATMOSPHERIC GASES
lat_fovcorner[1:4] pk9real, // LATITUDE FIELD OF VIEW CORNER 1-4
lon_fovcorner[1:4] pk9real, // LONGITUDE FIELD OF VIEW CORNER 1-4
solar_elevation pk9real, // SOLAR ELEVATION
fov pk1int, // FIELD OF VIEW NUMBER
);
D.1 Database layouts

```
cloud_cover pk9real, // CLOUD COVER
cloud_top_press pk9real, // PRESSURE AT TOP OF CLOUD
quality_retrieval pk1int, // QUALITY OF RETRIEVAL
number_layers pk1int, // NUMBER OF RETRIEVED LAYERS
);

-- The following depends on RTTOV, so watch for it!
SET $mx_satob_weight = 43;
CREATE TABLE satob AS (
  comp_method pk1int, // CLOUD MOTION COMP. METHOD
  instdata pk1int, // INS. DATA USED IN PROC.
  datproc pk1int, // DATA PROC. TECHNIQUE USED
  QI[3] pk1int, // Quality indicators
  segment_size_x pk9real, // RESOLUTION, x-direction
  segment_size_y pk9real, // RESOLUTION, y-direction
  chan_freq pk9real, // Satellite Channel Centre Frequency [Hz] (02197) for subtype=8
  t pk9real, // Temperature at SATOB p
  zenith pk9real, // Satellite zenith angle
  shear pk9real, // Diff. in speed 50hPa above/below
  t200 pk9real, // 200 hPa temperature
  t500 pk9real, // 500 hPa temperature
  top_mean_t pk9real, // Mean temperature between 80 hPa & p
  dt_by_dp pk9real, // Diff in temp. 50hPa above/below
  p_best pk9real, // "Best fit" pressure
  u_best pk9real, // U at "best fit" pressure
  v_best pk9real, // V at "best fit" pressure
  p_old pk9real, // Originally assigned pressure
  u_old pk9real, // U at old pressure
  v_old pk9real, // V at old pressure
  weight[$mx_satob_weight] pk9real, // Weight for obs operator (note hardwiring - see jpmx_satob_weight in module parcma)
);

CREATE TABLE atovs AS {
  #ifdef ECMA
  scanline pk1int, // SCAN LINE NUMBER
  fov pk1int, // FIELD OF VIEW NUMBER
  station_height pk9real, // HEIGHT OF STATION
  zenith pk9real, // SATELLITE ZENITH ANGLE
  bearing_azimuth pk9real, // BEARING OR AZIMUTH
  solar_zenith pk9real, // SOLAR ZENITH ANGLE
  solar_azimuth pk9real, // SOLAR AZIMUTH
  cldtop[1:3] pk9real, // CLOUD TOP PRESS. FROM HIRS RADIANCES
  cldne[1:3] pk9real, // CLOUD EMISSIVITY FROM HIRS RADIANCES
  vertsign[3:4] pk1int, // LAND/SEA QUALIFIER
  landsurf_height pk9real, // HEIGHT OF LAND SURFACE
  skintemp pk9real, // SKIN TEMPERATURE
  press[2:2] pk9real, // 1D VAR ITERATION NUMBER
  iterno_ldvar pk1int, // 1D VAR ERROR(S)
  channel_ldvar[0:1] pk1int,
  presat_flags pk1int,
  iterno_conv_ldvar pk1int,
  failure_ldvar pk1int,
  landsea_obs pk1int, // land/sea surface from obs report
  #endif /* ECMA */
  #ifdef CCMA
  #endif COMA
  fov pk1int, // FIELD OF VIEW NUMBER
```
zenith pk9real,  // SATELLITE ZENITH ANGLE
#endif /* CCMA */

CREATE TABLE atovs_pred AS (  
predictor[16] pk9real,  // SKIN TEMPERATURE, per NRESUPD
  skintemp[0NMXUPD] pk9real,
);

#define ECMA
CREATE TABLE scatt AS (  
track pk1int,  // SATELLITE TRACK
  cellno pk1int,  // CELL NO.
  prodflag pk1int,  // PRODUCT FLAG
  uwi_esa[2] pk1int,  // UWI ESA FLAG (2 x actually 12 bit words)
  scatt_body @LINK,
);
#endif/* ECMA */

#define ECMA
CREATE TABLE ssmi AS (  
orbit pk1int,  // ORBIT NO.
  scanline pk1int,  // SCAN LINE NO.
  typesurf pk1int,  // TYPE OF SURFACE
  vertsign pk1int,  // VERTICAL SIGNIFICANCE
  zenith pk9real,  // ZENITH ANGLE
  iterno_conv_1dvar pk1int,  // NO. OF ITERATIONS
  failure_ldvar pk1int,  // FAILURE INDICATOR
  surfpress[1] pk9real,  // from slev, SFC PRESSURE
  skintemp[1] pk9real,  // from slev, SKIN TEMPERATURE
  u10m[1] pk9real,  // from slev, U-WIND 10M
  v10m[1] pk9real,  // from slev, V-WIND 10M
  prec_st[2] pk9real,  // new , SFC STRAT. PRECIP.
  prec_cv[2] pk9real,  // new , SFC CONV. PRECIP.
  ssmi_body @LINK,
);
#endif/* ECMA */

CREATE TABLE body AS (  
varno pk1int,  // VARIABLE NUMBER
  vertco_type pk1int,  // VERTICAL COORDINATE TYPE
  anflag DATUM_FLAG_t,  // OBSERVATION FLAGS
  status STATUS_t,  // OBSERVATION STATUS
  event1 DATUM_EVENT1_t,  // OBSERVATION EVENTS (PART 1)
  entryno pk1int,  // ENTRY SQ. NO.
  press pk9real,  // VERTICAL COORDINATE REFERENCE 1
  press_r1 pk9real,  // VERTICAL COORDINATE REFERENCE 2
  obsvalue pk9real,  // OBSERVED VARIABLE
  aux[0NUMAUX] pk9real,  // Auxiliary observables;
  biascorr pk9real,  // RADIANCE BIAS CORRECTION
  rdbflag DATUM_RDBFLAG_t,  // OBSERVATION FLAGS (RDB)
#ifdef ECMA
  blacklist DATUM_BLACKLIST_t,  // OBSERVATION BLACKLIST EVENTS
  event2 DATUM_EVENT2_t,  // DATUM EVENTS (PART 2) WORD POS.
#endif /* ECMA */
D.1 Database layouts

```c
ppcode pk1int, // PRESSURE CODE
level LEVEL_t, // PILOT LEVEL ID.
#endif /* ECMA */
an_depar pk9real, // OBSERVED MINUS ANALYSED VALUE
gf_depar pk9real, // OBSERVED MINUS FIRST GUESS VALUE
gf_check[2] pk9real,
an_sens_obs pk9real, // analysis sensitivity to the obs
cld_fg_depar pk9real, // FG departure for cloudy radiances
surfemiss pk9real, // surface emissivity
csr_pclear pk9real, // percentage of clear pixels in mean CSR
csr_pcloudy pk9real, // percentage of cloudy pixels in mean CSR
rank_cld pk9real, // channel ranking for cloud detection
rank_an pk9real, // channel ranking for analysis
#ifdef CANARI
mf_vertco_type pk1int, // vertical coordinate type
mf_log_p pk9real, // pressure used in CANARI (Log P)
mf_stddev pk9real, // obs. std dev at bottom layer
#endif /* CANARI */
);

CREATE TABLE errstat AS ( final_obs_error pk9real, // FINAL OBSERVATION ERROR obs_error pk9real, // OBSERVATION ERROR repres_error pk9real, // REPRESENTATIVENESS ERROR pers_error pk9real, // PERSISTENCE ERROR fg_error pk9real, // FIRST GUESS ERROR )
);

// The following declares up to $NMXUPD update-tables with
// the naming convention update_1, update_2, ..., update_<$NMXUPD>
// Each of them has got exactly the same attributes.
// Note: It is up to the software to decide how many of these tables will
// actually be filled !!
CREATE TABLE update[1:$NMXUPD] AS ( initial pk9real, // OBS. MINUS UPD. U INITIAL VALUE hires pk9real, // OBS. MINUS UPD. U HIGH RES. VALUE lores pk9real, // OBS. MINUS UPD. U LOW RES. VALUE final pk9real, // OBS. MINUS UPD. U FINAL VALUE )
);
#ifdef ECMA
```
clw[2] pk9real, // new , CLOUD WATER MIX. RATIO
clw[2] pk9real, // new , CLOUD ICE MIX. RATIO
cc [2] pk9real, // new , CLOUD COVER, PROF.
);
#endif /* ECMA */

#ifdef ECMA
CREATE TABLE scatt_body AS (  
  azimuth pk9real, // BEAM AZIMUTH ANGLE
  incidence pk9real, // BEAM INCIDENCE ANGLE
  Kp pk9real, // Kp (INSTRUMENT NOISE; Antenna Noise)
  invresid[2] pk9real, // DIRECTIONAL SKILL
  dirskill pk9real,
  mpc pk1int, // MISSING PACKET COUNTER
  wvc_qf pk1int, // WIND VECTOR CELL QUALITY FLAG
  nretr_amb pk1int, // NUMBER OF RETRIEVED AMBIGUITIES
);
#endif /* ECMA */

$SRC/odb/ddl/mdih.h

SET $MDI = 2147483647; // Absolute value of the Missing Data Indicator

$SRC/odb/ddl/obstype.h

// CMA observation types (obstype@hdr) :

SET $SYNOP = 1;
SET $AIREP = 2;
SET $SATOB = 3;
SET $DRIBU = 4;
SET $BUYO = 4;
SET $TEMP = 5;
SET $PILOT = 6;
SET $SATEM = 7;
SET $PAOB = 8;
SET $SCATT = 9;

// CMA codetypes (used to be in obschar.codetype@hdr) for $SATEM :

SET $REO3 = 206;
SET $ATOV5 = 210;
SET $KTOS = 211;
SET $TOVS = 212;
SET $SSMI = 215;

SET $satem500 = 86;
SET $satem250 = 186;
SET $rad1c = 210;

$SRC/odb/ddl/ppcode.h
D.1 Database layouts

// Synop pressure codes (ppcode@body):
SET $psealev = 0; // press@body is SEA LEVEL PRESSURE
SET $pstalev = 1; // press@body is STATION LEVEL PRESSURE
SET $g850hpa = 2; // 850MB GEOPOTENTIAL
SET $g700hpa = 3; // 700MB GEOPOTENTIAL
SET $p500gpm = 4; // 500GPM PRESSURE
SET $p1000gpm = 5; // 1000GPM PRESSURE
SET $p2000gpm = 6; // 2000GPM PRESSURE
SET $p3000gpm = 7; // 3000GPM PRESSURE
SET $p4000gpm = 8; // 4000GPM PRESSURE
SET $g900hpa = 9; // 900MB GEOPOTENTIAL
SET $g1000hpa = 10; // 1000MB GEOPOTENTIAL
SET $g500hpa = 11; // 500MB GEOPOTENTIAL
SET $g925hpa = 12; // 925MB GEOPOTENTIAL

$SRC/odb/ddl/sensor.h

// sensor id's (sensor@hdr):
SET $hirs = 0;
SET $msu = 1;
SET $ssu = 2;
SET $amsua = 3;
SET $amsub = 4;
SET $meteosat = 20;

$SRC/odb/ddl/varno.h

// Variable numbers (varno@body):
SET $u = 3; // UPPER AIR U COMPONENT = NVNUMB(1)
SET $v = 4; // UPPER AIR V COMPONENT = NVNUMB(2)
SET $z = 1; // GEOPOTENTIAL = NVNUMB(3)
SET $dz = 57; // THICKNESS = NVNUMB(4)
SET $rh = 29; // UPPER AIR REL. HUMIDITY = NVNUMB(5)
SET $pwc = 9; // PWC = NVNUMB(6)
SET $rh2m = 58; // 2M REL. HUMIDITY = NVNUMB(7)
SET $t = 2; // UPPER AIR TEMPERATURE = NVNUMB(8)
SET $td = 59; // UPPER AIR DEW POINT = NVNUMB(9)
SET $t2m = 39; // 2M TEMPERATURE = NVNUMB(10)
SET $td2m = 40; // 2M DEW POINT = NVNUMB(11)
SET $ts = 11; // SURFACE TEMPERATURE = NVNUMB(12)
SET $ptend = 30; // PRESSURE TENDENCY = NVNUMB(13)
SET $w = 60; // PAST WEATHER (W) = NVNUMB(14)
SET $ww = 61; // PRESENT WEATHER (WW) = NVNUMB(15)
SET $vv = 62; // VISIBILITY = NVNUMB(16)
SET $ch = 63; // TYPE OF HIGH CLOUDS (Ch) = NVNUMB(17)
SET $cm = 64; // TYPE OF MIDDLE CLOUDS (Cm) = NVNUMB(18)
SET $cl = 65; // TYPE OF LOW CLOUDS (Cl) = NVNUMB(19)
SET $nh = 66; // CLOUD BASE HEIGHT (Nh) = NVNUMB(20)
SET $nn = 67; // LOW CLOUD AMOUNT (N) = NVNUMB(21)
SET $shs = 68; // ADDITIONAL CLOUD GROUP HEIGHT (hh) = NVNUMB(22)
SET $c = 69; // ADDITIONAL CLOUD GROUP TYPE (C) = NVNUMB(23)
SET $ns = 70; // ADDITIONAL CLOUD GROUP AMOUNT (Ns) = NVNUMB(24)
// Vertical coordinate types (vertco_type@body):

SET $pressure = 1; // press@body is pressure (in Pa)
SET $gheight = 2; // press@body is geopotential height
SET $tovs_cha = 3;
SET $scat_cha = 4;
D.1 Database layouts

//=== TYPE and TABLE definitions for the ERA-40 PREODB ===
//=== As of 28-Sep-1999 : by ECMWF/ERA-40 group
//===
//=== The present TABLE hierarchy
//=== -----------------------------------------------
//===
//=== ---+--> bufr
//===   |
//===   |
//===   desc --> hdr --+--> body
//===
//===
//=== === Global $-variables ===
//===
#include "mdi.h"
#include "obstype.h"
#include "varno.h"
#include "vertco_type.h"
#include "ppcode.h"

// For fixed dimensioning of some arrays in data layout :
SET $MAXDESC = 10; // No. of string-words (x8 bytes) per fulldesc
SET $MAXDUPL = 16; // Max. no. of different duplicate criterias (<= 32)

//=== TYPE definitions ===
//===
TYPEDEF FLAG_t AS {
  invalid bit1,
  invalid_date bit1,
  invalid_time bit1,
  invalid_lat bit1,
  invalid_lon bit1,
  invalid_statid bit1,
  invalid_statid bit1,
  passive bit1,
  duplicate bit1,
  _reserved bit6,
  dupl[$MAXDUPL] bit1,
};

TYPEDEF obschar_t = {

codetype bit9, // OBS. CODE TYPE
instype bit10, // INS. TYPE
retrtype bit6, // RETRIEVAL TYPE
geoarea bit6, // GEOGRAPHICAL AREA
);

// === TABLE definitions ===
// CREATE TABLE desc AS ( source string, version pk9int, acl pk9int, // Access control list (0=no, 0>has restrictions) andate YYYYMMDD, antime HHMMSS, creadate YYYYMMDD, creatime HHMMSS, creaby string, moddate YYYYMMDD, modtime HHMMSS, modby string, // Links hdr @LINK, 
// Make sure the $MAXDESC equals to no. of "fulldesc_" -entries fulldesc[$MAXDESC] string, 
);

CREATE TABLE hdr AS ( sortbox pk9int, // Box-id pre-calculated from (lat,lon) on 1x1 grid subclass pk9int, // 2ndry source seqno int, // OBSERVATION SEQUENCE obstype pk2int, // OBSERVATION TYPE (CMA) obschar OBSCHAR_t, // OBSERVATION CHARACTERISTICS (CMA) RDBtype pk5int, // RDB-type (BUFR) subtype pk5int, // Sub-type (BUFR) bufrtype pk5int, // BUFR-type (BUFR) date YYYYMMDD, // OBS. DATE time HHMMSS, // OBS. EXACT TIME statid string, // STATION ID lat real4, // LATITUDE (in degrees: -90..+90) lon real4, // LONGITUDE (in degrees: -180..+180) stalt pk9real, // ALTITUDE flag FLAG_t, // PREODB-flags numlev pk5int, // No. of distinct pressure levels in bodies duplseqno[$MAXDUPL] pk9int, // If > 0, seqno of the duplicate // Links body @LINK,
D.2 ODB interface to IFS

CREATE TABLE body AS (
  varno pk5int, // VARIABLE NUMBER
  press real8,  // VERTICAL COORDINATE (pressure or geop. ht)
  obsvalue real8, // OBSERVED VARIABLE
  aux1 pk9real, // Derived quantity: WSPEED, WDIR, etc..
  vertco_type pk5int, // VERTICAL COORDINATE TYPE
  prcode pk9int, // SYNOP PRESSURE CODE
);}

CREATE TABLE bufr AS (msg bufr,);

D.2 ODB interface to IFS

$SRC/ifs/module/yomdb.F90

MODULE YOMDB
USE PARKIND1, ONLY : JPIM, JPRB
USE PARCMA, ONLY : JPMXUP, JPXTSL, JPMX_SATOB_WEIGHT
USE YOMOML
IMPLICIT NONE
SAVE
PUBLIC

LOGICAL :: LODB = .TRUE.
CHARACTER(LEN=*), PARAMETER :: CLEVELDB = '28R2'
INTEGER(KIND=JPIM), PARAMETER :: MDIDB = 2147483647
!-- UNDEFDB moved to /cc/rd/odb/module/odbshared.F90 on 20-Mar-2002 by SS
!!INTEGER_M , PARAMETER :: UNDEFDB = -1953789045 ! "ODB_UNDEF" ; same as 0x8B8B8B8B
INTEGER(KIND=JPIM), PARAMETER :: JPMAXSIMVIEWS = 16
INTEGER(KIND=JPIM), PRIVATE :: i, j
INTEGER(KIND=JPIM), PARAMETER :: JPMAXSIMDB = 5
INTEGER(KIND=JPIM), PARAMETER :: NUMSIMDB = 0
INTEGER(KIND=JPIM) :: NHANDLEDB(JPMAXSIMDB) = (/ 0, i=1,JPMAXSIMDB)/
INTEGER(KIND=JPIM), PARAMETER :: NPOOLSLDB(JPMAXSIMDB) = (/ 0, i=1,JPMAXSIMDB)/
INTEGER(KIND=JPIM), PARAMETER :: NACTIVEDB = 0 ! Index to NHANDLEDB() for active DB
CHARACTER(Len=20) :: COPENDB(JPMAXSIMDB) = (/(' ', i=1,JPMAXSIMDB)/)
CHARACTER(Len=20) :: CSTATUSDB(JPMAXSIMDB) = (/(' ', i=1,JPMAXSIMDB)/)
CHARACTER(Len=20) :: CACTIVEDB = ' ' ! Say 'BUFRBASE', 'ECMA' or 'CCMA'
*** Disable UNMAPDB-function
LOGICAL :: LUNMAPDB_DO = .TRUE.

!!! Database handles
INTEGER(KIND=JPIM), PARAMETER :: JPMAXSIMDB = 5 ! Max. no. of simultaneously opened DBs
INTEGER(KIND=JPIM) :: NUMSIMDB = 0 ! No. of presently opened DBs
INTEGER(KIND=JPIM) :: NHANDLEDB(JPMAXSIMDB) = (/ 0, i=1,JPMAXSIMDB)/
INTEGER(KIND=JPIM), PARAMETER :: NPOOLSLDB(JPMAXSIMDB) = (/ 0, i=1,JPMAXSIMDB)/
INTEGER(KIND=JPIM), PARAMETER :: NACTIVEDB = 0 ! Index to NHANDLEDB() for active DB
CHARACTER(Len=20) :: COPENDB(JPMAXSIMDB) = (/(' ', i=1,JPMAXSIMDB)/)
CHARACTER(Len=20) :: CSTATUSDB(JPMAXSIMDB) = (/(' ', i=1,JPMAXSIMDB)/)
CHARACTER(Len=20) :: CACTIVEDB = ' ' ! Say 'BUFRBASE', 'ECMA' or 'CCMA'
!!! NPOOLS_ for most common databases
INTEGER(KIND=JPIM) :: NPOOLS_CCMA = 0
INTEGER(KIND=JPIM) :: NPOOLS_ECMA = 0
INTEGER(KIND=JPIM) :: NPOOLS_ECMASCR = 0

!*** Miscellaneous
LOGICAL :: L_DEBUG = .FALSE.
LOGICAL :: L_SWAPOUT = .FALSE.
LOGICAL :: L_CANARI = .FALSE.
LOGICAL :: L_CANALT = .FALSE.
LOGICAL :: L_REDUC = .FALSE.
LOGICAL :: L_DEGREES = .FALSE.
LOGICAL :: L_ACTIVE = .FALSE.

TYPE yomdb_t
  SEQUENCE
    #include "yomdb_vars.h"
  END TYPE yomdb_t

TYPE(yomdb_t), allocatable :: o_(:)

END MODULE YOMDB

$SRC/ifs/common/openmp_obs.h

#include "yomdb_vars.h"
#include "itdef.h"

$SRC/ifs/common/yomdb_vars.h

!*** Note: MAKE SURE ALL VARIABLES ARE IN lowercase !!
! Please make sure that variable lengths do not exceed 31 chars (a F90-std limit) !
! If you want to know what is what, please look at /cc/rd/odb/cma2odb/initmdb.F90

CHARACTER(LEN=128) :: cactiveretr
logical :: lactiveretr
logical :: allocated_satpred

!*** Observational array lengths
INTEGER(KIND=JPIM) :: nrows_robhdr
INTEGER(KIND=JPIM) :: ncols_robhdr
INTEGER(KIND=JPIM) :: nrows_satpred
INTEGER(KIND=JPIM) :: ncols_satpred
INTEGER(KIND=JPIM) :: nrows_satbody
INTEGER(KIND=JPIM) :: ncols_satbody
INTEGER(KIND=JPIM) :: nrows_sathdr
INTEGER(KIND=JPIM) :: ncols_sathdr
INTEGER(KIND=JPIM) :: nrows_mobody
INTEGER(KIND=JPIM) :: ncols_mobody
INTEGER(KIND=JPIM) :: nrows_mobhdr
INTEGER(KIND=JPIM) :: ncols_mobhdr
INTEGER(KIND=JPIM) :: nrows_satpred
INTEGER(KIND=JPIM) :: ncols_satpred
INTEGER(KIND=JPIM) :: nrows_robody
INTEGER(KIND=JPIM) :: ncols_robody
REAL(KIND=JPRB), pointer :: robhdr(:, :)
REAL(KIND=JPRB), pointer :: satpred(:,);
REAL(KIND=JPRB), pointer :: satbody(:,);
REAL(KIND=JPRB), pointer :: sathdr(:,); ! sometimes TARGET
INTEGER(KIND=JPIM), pointer :: mobody(:,);
INTEGER(KIND=JPIM), pointer :: mobhdr(:,);
REAL(KIND=JPRB), pointer :: robody(:,);
!*** for obs-setup retrievals obs_boxes, obatabs, mkglobstab, suobarea, eceset, smtov
INTEGER(KIND=JPIM), pointer :: mobsu(:,);
REAL(KIND=JPRB), pointer :: robsu(:,);
!*** link between header & body
INTEGER(KIND=JPIM), pointer :: mlnkh2b();
!*** dhrs
REAL(KIND=JPRB), pointer :: robddr(:,);
!*** dynamic mdb-pointers
INTEGER(KIND=JPIM) :: mdbhdrno_at_ddrs ! ‘ddrno@ddrs’
INTEGER(KIND=JPIM) :: mdbhwordno_at_ddrs ! ‘wordno@ddrs’
INTEGER(KIND=JPIM) :: mdbhbulkdata_at_ddrs ! ‘bulkdata@ddrs’
INTEGER(KIND=JPIM) :: mdbhandate_at_desc ! ‘andate@desc’
INTEGER(KIND=JPIM) :: mdbhantime_at_desc ! ‘antime@desc’
INTEGER(KIND=JPIM) :: mdbhexpver_at_desc ! ‘expver@desc’
INTEGER(KIND=JPIM) :: mlink_desc2ddrs(2)
INTEGER(KIND=JPIM) :: mdbhseqno ! ‘seqno@hdr’
INTEGER(KIND=JPIM) :: mdbhtype ! ‘obstype@hdr’
!INTEGER_M :: mdbhchar ! ‘obschar@hdr’
INTEGER(KIND=JPIM) :: mdbhdate ! ‘date@hdr’
INTEGER(KIND=JPIM) :: mdbhetm ! ‘time@hdr’
INTEGER(KIND=JPIM) :: mdbhflag ! ‘rdbflag@hdr’
INTEGER(KIND=JPIM) :: mbhsstatus ! ‘status@hdr’
INTEGER(KIND=JPIM) :: mdbhver1 ! ‘event1@hdr’
INTEGER(KIND=JPIM) :: mdbhblacklist ! ‘blacklist@hdr’
INTEGER(KIND=JPIM) :: mdbhsortbox ! ‘sortbox@hdr’
INTEGER(KIND=JPIM) :: mdbhstd ! ‘sitedep@hdr’
INTEGER(KIND=JPIM) :: mbhsid ! ‘statid@hdr’
INTEGER(KIND=JPIM) :: mbhlat ! ‘lat@hdr’
INTEGER(KIND=JPIM) :: mbhlon ! ‘lon@hdr’
INTEGER(KIND=JPIM) :: mbhstalt ! ‘stalt@hdr’
INTEGER(KIND=JPIM) :: mbhmodoro ! ‘modoro@hdr’
INTEGER(KIND=JPIM) :: mbhtrlat ! ‘trlat@hdr’
INTEGER(KIND=JPIM) :: mbhtrlon ! ‘trlon@hdr’
INTEGER(KIND=JPIM) :: mbhinstspec ! ‘instspec@hdr’
INTEGER(KIND=JPIM) :: mbhver2 ! ‘event2@hdr’
INTEGER(KIND=JPIM) :: mbhanoeh ! ‘anemoht@hdr’
INTEGER(KIND=JPIM) :: mbhanoet ! ‘anemoht@hdr’
INTEGER(KIND=JPIM) :: mbhanoamt ! ‘anemoht@hdr’
INTEGER(KIND=JPIM) :: mbhanoep ! ‘anemoht@hdr’
INTEGER(KIND=JPIM) :: mbhobaroth ! ‘baroht@hdr’
INTEGER(KIND=JPIM) :: mbhobaro ! ‘baroht@hdr’
INTEGER(KIND=JPIM) :: mbhobaroat ! ‘baroht@hdr’
INTEGER(KIND=JPIM) :: mbshcomp_method ! ‘comp_method@satob’
INTEGER(KIND=JPIM) :: mbshinstdata ! ‘instdata@satob’
INTEGER(KIND=JPIM) :: mbshdataproc ! ‘dataproc@satob’
INTEGER(KIND=JPIM) :: mbshqi1 ! ‘qi_1@satob’
INTEGER(KIND=JPIM) :: mbshqi2 ! ‘qi_2@satob’
INTEGER(KIND=JPIM) :: mbshqi3 ! ‘qi_3@satob’
INTEGER(KIND=JPIM) :: mbstbth_at_satob ! ‘th@satob’
INTEGER(KIND=JPIM) :: mbstb_t_at_satob ! ‘t@satob’
INTEGER(KIND=JPIM) :: mbstbzenith_at_satob ! ‘zenith@satob’
INTEGER(KIND=JPIM) :: mbstbshear_at_satob ! ‘shear@satob’
INTEGER(KIND=JPIM) :: mbstb200_at_satob ! ‘t200@satob’
INTEGER(KIND=JPIM) :: mbstb500_at_satob ! ‘t500@satob’
INTEGER(KIND=JPIM) :: mbstbtop_mean_t_at_satob ! ‘top_mean_t@satob’
INTEGER (KIND=JPIM) :: mdb_top_wv_at_satob  ! 'top_wv@satob'
INTEGER (KIND=JPIM) :: mdb_dt_by_dp_at_satob  ! 'dt_by_dp@satob'
INTEGER (KIND=JPIM) :: mdb_p_best_at_satob  ! 'p_best@satob'
INTEGER (KIND=JPIM) :: mdb_u_best_at_satob  ! 'u_best@satob'
INTEGER (KIND=JPIM) :: mdb_v_best_at_satob  ! 'v_best@satob'
INTEGER (KIND=JPIM) :: mdb_p_old_at_satob  ! 'p_old@satob'
INTEGER (KIND=JPIM) :: mdb_u_old_at_satob  ! 'u_old@satob'
INTEGER (KIND=JPIM) :: mdb_v_old_at_satob  ! 'v_old@satob'
INTEGER (KIND=JPIM) :: mdb_weight_at_satob(jpmx_satob_weight)  ! 'weight@satob'
INTEGER (KIND=JPIM) :: mdbssia  ! 'sensor@hdr'
INTEGER (KIND=JPIM) :: mdbscla  ! 'scanline@atovs'
INTEGER (KIND=JPIM) :: mdbfova  ! 'fov@atovs'
INTEGER (KIND=JPIM) :: mdbsthia  ! 'station_height@atovs'
INTEGER (KIND=JPIM) :: mdbseaza  ! 'zenith@atovs'
INTEGER (KIND=JPIM) :: mdbseaba  ! 'bearing_azimuth@atovs'
INTEGER (KIND=JPIM) :: mbdsoza  ! 'solar_zenith@atovs'
INTEGER (KIND=JPIM) :: mdbsoba  ! 'solar_azimuth@atovs'
INTEGER (KIND=JPIM) :: mdbvs3a  ! 'vertsign_3@atovs'
INTEGER (KIND=JPIM) :: mdbvsqa  ! 'landsea@atovs'
INTEGER (KIND=JPIM) :: mdbhlsa  ! 'landsurf_height@atovs'
INTEGER (KIND=JPIM) :: mdbskta  ! 'skintemp@atovs'
INTEGER (KIND=JPIM) :: dbpp2a  ! 'press_2@atovs'
INTEGER (KIND=JPIM) :: mdbvs4a  ! 'vertsign_4@atovs'
INTEGER (KIND=JPIM) :: mdbldita  ! 'iterno_1dvar@atovs'
INTEGER (KIND=JPIM) :: mdbldera  ! 'error_1dvar@atovs'
INTEGER (KIND=JPIM) :: mdbldcua  ! 'channel_1dvar_0@atovs'
INTEGER (KIND=JPIM) :: mdbldcula  ! 'channel_1dvar_1@atovs'
INTEGER (KIND=JPIM) :: mdbldpfa  ! 'presat_flags@atovs'
INTEGER (KIND=JPIM) :: mdbldina  ! 'iterno_conv_1dvar@atovs'
INTEGER (KIND=JPIM) :: mdbldfia  ! 'failure_1dvar@atovs'
INTEGER (KIND=JPIM) :: mbdcp1a  ! 'predictor_1@atovs_pred'
INTEGER (KIND=JPIM) :: mbdcp2a  ! 'predictor_2@atovs_pred'
INTEGER (KIND=JPIM) :: mbdcp3a  ! 'predictor_3@atovs_pred'
INTEGER (KIND=JPIM) :: mbdcp4a  ! 'predictor_4@atovs_pred'
INTEGER (KIND=JPIM) :: mbdcp5a  ! 'predictor_5@atovs_pred'
INTEGER (KIND=JPIM) :: mbdcp6a  ! 'predictor_6@atovs_pred'
INTEGER (KIND=JPIM) :: mbdcp7a  ! 'predictor_7@atovs_pred'
INTEGER (KIND=JPIM) :: mbdcp8a  ! 'predictor_8@atovs_pred'
INTEGER (KIND=JPIM) :: mbdcp9a  ! 'predictor_9@atovs_pred'
INTEGER (KIND=JPIM) :: mbdcp10a  ! 'predictor_10@atovs_pred'
INTEGER (KIND=JPIM) :: mbdcp11a  ! 'predictor_11@atovs_pred'
INTEGER (KIND=JPIM) :: mbdcp12a  ! 'predictor_12@atovs_pred'
INTEGER (KIND=JPIM) :: mbdcp13a  ! 'predictor_13@atovs_pred'
INTEGER (KIND=JPIM) :: mbdcp14a  ! 'predictor_14@atovs_pred'
INTEGER (KIND=JPIM) :: mbdcp15a  ! 'predictor_15@atovs_pred'
INTEGER (KIND=JPIM) :: mbdcp16a  ! 'predictor_16@atovs_pred'
INTEGER (KIND=JPIM) :: mbdcpts1(jpmxup)  ! skintemp[$NMXUPD]@atovs_pred
INTEGER (KIND=JPIM) :: mbscsat  ! 'track@scatt'
INTEGER (KIND=JPIM) :: mdbscno  ! 'cellno@scatt'
INTEGER (KIND=JPIM) :: mbscpfl  ! 'prodflag@scatt'
INTEGER (KIND=JPIM) :: mdborno  ! 'orbit@ssmi'
INTEGER (KIND=JPIM) :: mdbslno  ! 'scanline@ssmi'
INTEGER (KIND=JPIM) :: mdbpsnas  ! 'scanpos@ssmi'
INTEGER (KIND=JPIM) :: mdbtos  ! 'typesurf@ssmi'
INTEGER (KIND=JPIM) :: mdbvsqg  ! 'vertsign@ssmi'
INTEGER (KIND=JPIM) :: mdbzan  ! 'zenith@ssmi'
INTEGER (KIND=JPIM) :: mdbldmit  ! 'iterno_conv_1dvar@ssmi'
INTEGER (KIND=JPIM) :: mdbldfin  ! 'failure_1dvar@ssmi'
INTEGER (KIND=JPIM) :: mdbbps  ! 'surfpress_1@ssmi'
INTEGER (KIND=JPIM) :: mdbbsta  ! 'skintemp_1@ssmi'
INTEGER (KIND=JPIM) :: mdbhsus  ! 'u10m_1@ssmi'
INTEGER (KIND=JPIM) :: mdbbsvs  ! 'v10m_1@ssmi'
INTEGER(KIND=JPIM) :: mdb1dvps ! 'press@ssmi_body'
INTEGER(KIND=JPIM) :: mdb1dvbt ! 'temp_1@ssmi_body'
INTEGER(KIND=JPIM) :: mdb1dvat ! 'temp_2@ssmi_body'
INTEGER(KIND=JPIM) :: mdb1dvq ! 'q_1@ssmi_body'
INTEGER(KIND=JPIM) :: mdb1dvaq ! 'q_2@ssmi_body'
INTEGER(KIND=JPIM) :: mdb_procid_at_index ! 'procid@index'
INTEGER(KIND=JPIM) :: mdb_target_at_index ! 'target@index'
INTEGER(KIND=JPIM) :: mdb_kset_at_index ! 'kset@index'
INTEGER(KIND=JPIM) :: mdb_tslot_at_index ! 'tslot@index'
INTEGER(KIND=JPIM) :: mdb_abnob_at_index ! 'abnob@index'
INTEGER(KIND=JPIM) :: mdb_mapomm_at_index ! 'mapomm@index'
INTEGER(KIND=JPIM) :: mdb_ident_at_hdr ! 'ident@hdr'
INTEGER(KIND=JPIM) :: mdb_thinningkey_at_hdr(jp_numthbox) ! 'thinningkey@hdr'
INTEGER(KIND=JPIM) :: mdb_thinningtimekey_at_hdr ! 'thinningtimekey@hdr'
INTEGER(KIND=JPIM) :: mlnk_index2hdr(2)
INTEGER(KIND=JPIM) :: mlnk_hdr2body(2)
INTEGER(KIND=JPIM) :: mlnk_hdr2update(2,jpmxup)
INTEGER(KIND=JPIM) :: mlnk_hdr2errstat(2)
INTEGER(KIND=JPIM) :: mlnk_hdr2sat(2)
INTEGER(KIND=JPIM) :: mlnk_sat2atovs(2)
INTEGER(KIND=JPIM) :: mlnk_sat2reo3(2)
INTEGER(KIND=JPIM) :: mlnk_sat2ssmi(2)
INTEGER(KIND=JPIM) :: mlnk_sat2scatt(2)
INTEGER(KIND=JPIM) :: mlnk_sat2stream(2)
INTEGER(KIND=JPIM) :: mlnk_sat2satob(2)
INTEGER(KIND=JPIM) :: mlnk_atovs2atovs_pred(2)
INTEGER(KIND=JPIM) :: mlnk_scatt2scatt_body(2)
INTEGER(KIND=JPIM) :: mdbvnm ! 'varno@body'
INTEGER(KIND=JPIM) :: mdbvco ! 'vertco_type@body'
INTEGER(KIND=JPIM) :: mdbrdfl ! 'rdbflag@body'
INTEGER(KIND=JPIM) :: mdbflg ! 'anflag@body'
INTEGER(KIND=JPIM) :: mdbdsta ! 'status@body'
INTEGER(KIND=JPIM) :: mdbdev1 ! 'event1@body'
INTEGER(KIND=JPIM) :: mdbdev2 ! 'event2@body'
INTEGER(KIND=JPIM) :: mdbsypc ! 'ppcode@body'
INTEGER(KIND=JPIM) :: mdbtelid ! 'level@body'
INTEGER(KIND=JPIM) :: mdbpilid ! 'level@body'
INTEGER(KIND=JPIM) :: mdbtorb ! 'biascorr@body'
INTEGER(KIND=JPIM) :: mdbtorba ! 'biascorr@body'
INTEGER(KIND=JPIM) :: mdbssrb ! 'biascorr@body'
INTEGER(KIND=JPIM) :: mdb1dvc ! 'radcost@ssmi_body'
INTEGER(KIND=JPIM) :: mdbssccf ! 'frequency@ssmi_body'
INTEGER(KIND=JPIM) :: mdbsscbw ! 'bandwidth@ssmi_body'
INTEGER(KIND=JPIM) :: mdbssanp ! 'polarisation@ssmi_body'
INTEGER(KIND=JPIM) :: mdbcbaa ! 'azimuth@scatt_body'
INTEGER(KIND=JPIM) :: mdbcbia ! 'incidence@scatt_body'
INTEGER(KIND=JPIM) :: mdbckp ! 'kp@scatt_body'
INTEGER(KIND=JPIM) :: mdbcirf ! 'invresid_1@scatt_body'
INTEGER(KIND=JPIM) :: mdbirs ! 'invresid_2@scatt_body'
INTEGER(KIND=JPIM) :: mdbscirf ! 'invresid_10@scatt_body'
INTEGER(KIND=JPIM) :: mdb_satid_at_sat ! 'satid@sat'
INTEGER(KIND=JPIM) :: mdb_cldptop1_at_atovs ! 'cldptop1@atovs'
INTEGER(KIND=JPIM) :: mdb_cldne1_at_atovs ! 'cldne1@atovs'
INTEGER(KIND=JPIM) :: mdb_cldptop2_at_atovs ! 'cldptop2@atovs'
INTEGER(KIND=JPIM) :: mdb_cldne2_at_atovs ! 'cldne2@atovs'
INTEGER(KIND=JPIM) :: mdb_cldptop3_at_atovs ! 'cldptop3@atovs'
INTEGER(KIND=JPIM) :: mdb_cldne3_at_atovs ! 'cldne3@atovs'
INTEGER(KIND=JPIM) :: mdbro3it ! 'instrument_type@reo3'
INTEGER(KIND=JPIM) :: mdbro3pt ! 'product_type@reo3'
INTEGER(KIND=JPIM) :: mdbro3la1 ! 'lat_fovcorner_1@reo3'
INTEGER(KIND=JPIM) :: mdbro3lo1 ! 'lon_fovcorner_1@reo3'
INTEGER(KIND=JPIM) :: mdbro3la2 ! 'lat_fovcorner_2@reo3'
INTEGER(KIND=JPIM) :: mdbro3lo2 ! 'lon_fovcorner_2@reo3'
INTEGER(KIND=JPIM) :: mdbro3la3 ! 'lat_fovcorner_3@reo3'
INTEGER(KIND=JPIM) :: mdbro3lo3 ! 'lon_fovcorner_3@reo3'
INTEGER(KIND=JPIM) :: mdbro3la4 ! 'lat_fovcorner_4@reo3'
INTEGER(KIND=JPIM) :: mdbro3lo4 ! 'lon_fovcorner_4@reo3'
INTEGER(KIND=JPIM) :: mdbro3soe ! 'solar_elevation@reo3'
INTEGER(KIND=JPIM) :: mdbro3fov ! 'fov@reo3'
INTEGER(KIND=JPIM) :: mdbro3clc ! 'cloud_cover@reo3'
INTEGER(KIND=JPIM) :: mdbro3cp ! 'cloud_top_press@reo3'
INTEGER(KIND=JPIM) :: mdbro3qr ! 'quality_retrieval@reo3'
INTEGER(KIND=JPIM) :: mdbro3nl ! 'number_layers@reo3'
INTEGER(KIND=JPIM) :: mlnk_desc2hdr(2)
INTEGER(KIND=JPIM) :: mlnk_hdr2bufr(2)
INTEGER(KIND=JPIM) :: mdb_subclass_at_hdr ! 'subclass@hdr'
INTEGER(KIND=JPIM) :: mdb_rdbtype_at_hdr ! 'rdbtype@hdr'
INTEGER(KIND=JPIM) :: mdb_subtype_at_hdr ! 'subtype@hdr'
INTEGER(KIND=JPIM) :: mdb_flag_at_hdr ! 'flag@hdr'
INTEGER(KIND=JPIM) :: mdb_numlev_at_hdr ! 'numlev@hdr'
INTEGER(KIND=JPIM) :: mdb_numactiveb_at_hdr ! 'numactiveb@hdr'
INTEGER(KIND=JPIM) :: mdb_dupseqno_at_hdr(jp_maxdup) ! 'dupseqno@hdr'
INTEGER(KIND=JPIM) :: mdb_mag_at_bufr ! 'mag@bufr'
INTEGER(KIND=JPIM) :: mdb_thinbox_at_hdr ! 'thinbox@hdr'
INTEGER(KIND=JPIM) :: mdb_msec_at_hdr ! 'msec@hdr'
INTEGER(KIND=JPIM) :: mdb_satid_at_hdr ! 'satid@hdr'
INTEGER(KIND=JPIM) :: mdb_orbit_at_hdr ! 'orbit@hdr'
INTEGER(KIND=JPIM) :: mdb_fov_at_hdr ! 'fov@hdr'
INTEGER(KIND=JPIM) :: mdb_scanline_at_hdr ! 'scanline@hdr'
INTEGER(KIND=JPIM) :: mdb_zenith_at_hdr ! 'zenith@hdr'
INTEGER(KIND=JPIM) :: mlnk_hdr2inbufr(2)
INTEGER(KIND=JPIM) :: mdb_msg_at_inbufr ! 'msg@inbufr'
INTEGER(KIND=JPIM) :: mlnk_hdr2outbufr(2)
INTEGER(KIND=JPIM) :: mdb_msg_at_outbufr ! 'msg@outbufr'
INTEGER(KIND=JPIM) :: mdb_varno_presence_at_hdr ! 'varno_presence@hdr'

*** New items added due to direct BUFR2ODB conversion
INTEGER(KIND=JPIM) :: mdb_station_type_at_hdr ! 'station_type@hdr'
INTEGER(KIND=JPIM) :: mdb_sonde_type_at_hdr ! 'sonde_type@hdr'
INTEGER(KIND=JPIM) :: mdb_uwi_esa_at_scatt(2) ! 'uwi_esa@scatt'
INTEGER(KIND=JPIM) :: mdb_mpc_at_scatt_body ! 'mpc@scatt_body'
INTEGER(KIND=JPIM) :: mdb_wvc_qf_at_scatt_body ! 'wvc_qf@scatt_body'
INTEGER(KIND=JPIM) :: mdb_nretr_am@scatt_body ! 'nretr_am@scatt_body'
INTEGER(KIND=JPIM) :: mdb_subsetno_at_hdr ! 'subsetno@hdr'

*** New items added due to Poolmasking
INTEGER(KIND=JPIM) :: mlnk_desc2poolmask(2)
INTEGER(KIND=JPIM) :: mdb_poolno_at_poolmask ! 'poolno@poolmask'
INTEGER(KIND=JPIM) :: mdb_obstype_at_poolmask ! 'obstype@poolmask'
INTEGER(KIND=JPIM) :: mdb_codetype_at_poolmask ! 'codetype@poolmask'
INTEGER(KIND=JPIM) :: mdb_sensor_at_poolmask ! 'sensor@poolmask'
INTEGER(KIND=JPIM) :: mdb_subtype_at_poolmask ! 'subtype@poolmask'
INTEGER(KIND=JPIM) :: mdb_bufrtype_at_poolmask ! 'bufrtype@poolmask'
INTEGER(KIND=JPIM) :: mdb_hdr_count_at_poolmask ! 'hdr_count@poolmask'
INTEGER(KIND=JPIM) :: mdb_body_count_at_poolmask ! 'body_count@poolmask'
INTEGER(KIND=JPIM) :: mdb_max_bodylen_at_poolmask ! 'max_bodylen@poolmask'

*** Commonly needed subwords/members of 'obschar.*@hdr'

*** Additional @satob entries requested by Niels Bormann [20/11/2001]
INTEGER(KIND=JPIM) :: mdb_segment_size_x_at_satob ! 'segment_size_x@satob'
INTEGER(KIND=JPIM) :: mdb_segment_size_y_at_satob ! 'segment_size_y@satob'

*** New item(s)
INTEGER(KIND=JPIM) :: mdb_zone_at_hdr ! 'zone@hdr'
INTEGER(KIND=JPIM) :: mdb_satinst_at_hdr ! 'satinst@hdr'
INTEGER(KIND=JPIM) :: mdb_satinst_at_poolmask ! 'satinst@poolmask'
INTEGER(KIND=JPIM) :: mdb_satname_at_hdr(2) ! 'satname@hdr'

*** Table @satem removed and items moved into @hdr with different names
   (for old SATEM) by Sami Saarinen [26/6/2003]
INTEGER(KIND=JPIM) :: mdb_satem_dataproc_at_hdr ! 'satem_dataproc@hdr'
INTEGER(KIND=JPIM) :: mdb_satem_instdata_at_hdr ! 'satem_instdata@hdr'

*** MDB_AUX1_AT_BODY obsolete (except PREODB)
! Elsewhere replaced by (vector) of MDB_AUX_AT_BODY [23/1/04 SS]
INTEGER(KIND=JPIM) :: mdb_aux_at_body(jp_numaux) ! 'aux@body'
INTEGER(KIND=JPIM) :: mdb_aux1_at_body ! 'aux1@body'

*** latlon_rad@desc added to flag whether (lat,lon) are in radians or not [23/1/04 SS]
INTEGER(KIND=JPIM) :: mdb_latlon_rad_at_desc ! 'latlon_rad@desc'

*** SSM/I reorganization (Peter Bauer/Sami Saarinen) [11/3/04]
INTEGER(KIND=JPIM) :: mdb_prec_st_at_ssmi(2) ! 'prec_st[2]@ssmi'
INTEGER(KIND=JPIM) :: mdb_prec_cv_at_ssmi(2) ! 'prec_cv[2]@ssmi'
INTEGER(KIND=JPIM) :: mdb_rain_at_ssmi_body(2) ! 'rain[2]@ssmi_body'
INTEGER(KIND=JPIM) :: mdb_snow_at_ssmi_body(2) ! 'snow[2]@ssmi_body'
INTEGER(KIND=JPIM) :: mdb_clw_at_ssmi_body(2) ! 'clw[2]@ssmi_body'
INTEGER(KIND=JPIM) :: mdb_ciw_at_ssmi_body(2) ! 'ciw[2]@ssmi_body'
INTEGER(KIND=JPIM) :: mdb_cc_at_ssmi_body(2) ! 'cc[2]@ssmi_body'
INTEGER(KIND=JPIM) :: mdb_rad_cost_at_ssmi_body ! 'rad_cost@ssmi_body'
INTEGER(KIND=JPIM) :: mdb_rad_obs_at_ssmi_body ! 'rad_obs@ssmi_body'
INTEGER(KIND=JPIM) :: mdb_rad_dep_br_at_ssmi_body ! 'rad_dep_br@ssmi_body'
INTEGER(KIND=JPIM) :: mdb_rad_an_depar_at_ssmi_body ! 'rad_an_depar@ssmi_body'
INTEGER(KIND=JPIM) :: mdb_rad_obs_err_at_ssmi_body ! 'rad_obs_err@ssmi_body'
INTEGER(KIND=JPIM) :: mdb_rad_bias_at_ssmi_body ! 'rad_bias@ssmi_body'

*** checksum@hdr-word added to get over reproducibility problems for good [18/3/04 SS]
! contains the SUM(obsvalue) WHERE press IS NOT NULL and obsvalue IS NOT NULL
! just after bufr2odb (in mergeodb; routine odb/cma2odb/update_obsdb.F90)
INTEGER(KIND=JPIM) :: mdb_checksum_at_hdr ! 'checksum@hdr'

!*** Note: make sure that the variable to be redefined is in UPPERCASE
! and MAKE SURE THAT THE REDEFINITION (o_(it)...) IS IN lowercase !!
! If you want to know what is what, please look at /cc/rd/odb/cma2odb/initmdb.F90
#define CACTIVERETR o_(it)%cactiveretr
#define LACTIVERETR o_(it)%lactiveretr

*** Observational array lengths
#define ALLOCATED_SATPRED o_(it)%allocated_satpred
#define NROWS_ROWHDR o_(it)%nrows_rowhdr
#define NCOLS_ROWHDR o_(it)%ncols_rowhdr
#define NROWS_SATPRED o_(it)%nrows_satpred
#define NCOLS_SATPRED o_(it)%ncols_satpred
#define NROWS_SATBODY o_(it)%nrows_satbody
#define NCOLS_SATBODY o_(it)%ncols_satbody
#define NROWS_SATHDR o_(it)%nrows_sathdr
#define NCOLS_SATHDR o_(it)%ncols_sathdr
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
#define NROWS_MOBODY o_(it)%nrows_mobody
#define NCOLS_MOBODY o_(it)%ncols_mobody
D.2 ODB interface to IFS

```c
#define SATBODY o_(it)%satbody
#define SATHDR o_(it)%sathdr
#define MOBODY o_(it)%mobody
#define MOBHDR o_(it)%mobhdr
#define ROBODY o_(it)%robody
#define MOBSU o_(it)%mobsu
#define ROBSU o_(it)%robsu
#define MLNKH2B o_(it)%mlnkh2b
#define ROBDDR o_(it)%robddr

!*** Dynamic MDB-pointers to address obs. array columns
#define MDB_DDRNO_AT_DDRS o_(it)%mdb_ddrno_at_ddrs
#define MDB_WORDNO_AT_DDRS o_(it)%mdb_wordno_at_ddrs
#define MDB_BULKDATA_AT_DDRS o_(it)%mdb_bulkdata_at_ddrs
#define MDB_ANDATE_AT_DESC o_(it)%mdb_andate_at_desc
#define MDB_ANTIME_AT_DESC o_(it)%mdb_antime_at_desc
#define MLNK_DESC2DDR o_(it)%mlnk_desc2ddrs
#define MDBONM o_(it)%mdbonm
#define MDBOTP o_(it)%mdbotp
#define MDBOCH o_(it)%mdboch
#define MDBDAT o_(it)%mdbdat
#define MDBETM o_(it)%mdbetm
#define MDBRFIL o_(it)%mdbrfil
#define MDBRST o_(it)%mdbrst
#define MDBREV1 o_(it)%mdbrev1
#define MDBRBLE o_(it)%mdbrble
#define MDBBOX o_(it)%mdbbox
#define MDBSTD o_(it)%mdbstd
#define MDSID o_(it)%mdsid
#define MDBLAT o_(it)%mdblat
#define MDBLON o_(it)%mdblon
#define MDBALT o_(it)%mdbalt
#define MDBMOR o_(it)%mdbmor
#define MDBTLA o_(it)%mdbtla
#define MDBTLO o_(it)%mdbtlo
#define MDRING o_(it)%mdrings
#define MDRREV2 o_(it)%mdrrev2
#define MDRHOANS o_(it)%mdrhoons
#define MDBHOAND o_(it)%mdbhoand
#define MDBHOANT o_(it)%mdbhoant
#define MDRHOANP o_(it)%mdrhopans
#define MDRHOAS o_(it)%mdrhosas
#define MDRHOBAD o_(it)%mdrhobad
#define MDRHOBAT o_(it)%mdrhobat
#define MDRHOBAP o_(it)%mdrhobap
#define MDSBCMM o_(it)%mdsbcmmm
#define MDSBSTITUP o_(it)%mdsbiup
#define MDSBSBDPT o_(it)%mdbsbdpt
#define MDSBSQI1 o_(it)%mdbsq11
#define MDSBSQIQ2 o_(it)%mdbsqi2
#define MDSBSQIQ3 o_(it)%mdbsqi3
#define MDB_TB_AT_SATOB o_(it)%mdb_tb_at_satob
#define MDB_T_AT_SATOB o_(it)%mdb_t_at_satob
#define MDB_CHAN_FREQ_AT_SATOB o_(it)%mdb_chan_freq_at_satob
#define MDB_ZENITH_AT_SATOB o_(it)%mdb_zenith_at_satob
#define MDB_SHEAR_AT_SATOB o_(it)%mdb_shear_at_satob
#define MDB_T200_AT_SATOB o_(it)%mdb_t200_at_satob
#define MDB_T500_AT_SATOB o_(it)%mdb_t500_at_satob
#define MDR_Top_MEAN_T_AT_SATOB o_(it)%mdb_top_mean_t_at_satob
#define MDR_Top_WV_AT_SATOB o_(it)%mdb_top_wv_at_satob
#define MDR_DT_by_DP_AT_SATOB o_(it)%mdb_dt_by_dp_at_satob
#define MDR_P_BEST_AT_SATOB o_(it)%mdb_p_best_at_satob
```
#define MDB_U_BEST_AT_SATOB o_(it)%mdb_u_best_at_satob
#define MDB_V_BEST_AT_SATOB o_(it)%mdb_v_best_at_satob
#define MDB_P_OLD_AT_SATOB o_(it)%mdb_p_old_at_satob
#define MDB_V_OLD_AT_SATOB o_(it)%mdb_v_old_at_satob
#define MDB_WEIGHT_AT_SATOB o_(it)%mdb_weight_at_satob
#define MDBSSIA o_(it)%mdbssia
#define MDBSCLA o_(it)%mdbscla
#define MDBFOVA o_(it)%mdbfova
#define MDBSTHA o_(it)%mdbstha
#define DBSAZA o_(it)%mdbsaza
#define DBSABA o_(it)%mdbsaba
#define DBS2A o_(it)%mdbs2a
#define DBSABA o_(it)%mdbsaba
#define DBS3A o_(it)%mdbs3a
#define DBLSQA o_(it)%dblsqa
#define DBHLSA o_(it)%dbhlsa
#define DBSKTA o_(it)%dbskta
#define DBFP2A o_(it)%dbfp2a
#define DBV04A o_(it)%dbv04a
#define DBIDITA o_(it)%dbidita
#define DBIDERA o_(it)%dbidera
#define DBIDCUA o_(it)%dbidcua
#define DBIDCU1A o_(it)%dbidcua1
#define DBIDPFA o_(it)%dbidpfa
#define DB11DINA o_(it)%db11dina
#define DB11DFIA o_(it)%db11dfia
#define DB1BPC1A o_(it)%db1bpc1a
#define DB1BPC2A o_(it)%db1bpc2a
#define DB1BPC3A o_(it)%db1bpc3a
#define DB1BPC4A o_(it)%db1bpc4a
#define DB1BPC5A o_(it)%db1bpc5a
#define DB1BPC6A o_(it)%db1bpc6a
#define DB1BPC7A o_(it)%db1bpc7a
#define DB1BPC8A o_(it)%db1bpc8a
#define DB1BPC9A o_(it)%db1bpc9a
#define DB1BPC10A o_(it)%db1bpc10a
#define DB1BPC11A o_(it)%db1bpc11a
#define DB1BPC12A o_(it)%db1bpc12a
#define DB1BPC13A o_(it)%db1bpc13a
#define DB1BPC14A o_(it)%db1bpc14a
#define DB1BPC15A o_(it)%db1bpc15a
#define DB1BPC16A o_(it)%db1bpc16a
#define DB1BPCPTS1 o_(it)%db1bpcpts1
#define DB1BCSAT o_(it)%db1bcsat
#define DB1BCSCAT o_(it)%db1bcsat
#define DB1BCSCNO o_(it)%db1bcscno
#define DB1BSCPFL o_(it)%db1bscpfl
#define DB1BDORN0 o_(it)%db1bdorn0
#define DB1BLSINO o_(it)%db1blsino
#define DB1BPNAS o_(it)%db1bpnas
#define DB1BTSOS o_(it)%db1btsos
#define DB1BSLG o_(it)%db1bslg
#define DB1BSTNIT o_(it)%db1bstnit
#define DB1BDFIN o_(it)%db1bdfin
#define DB1BPS o_(it)%db1bps
#define DB1BSTS o_(it)%db1bsts
#define DB1BSUS o_(it)%db1bsus
#define DB1BSVS o_(it)%db1bsvs
#define DB1DVPS o_(it)%db1dvps
#define DB1DVST o_(it)%db1dvst
#define DB1DVAT o_(it)%db1dvat
#define MDB1DVBQ o_(it)%mdb1dvbq
#define MDB1DVQ o_(it)%mdb1dvq
#define MDB_PROCID_AT_INDEX o_(it)%mdb_procid_at_index
#define MDB_TARGET_AT_INDEX o_(it)%mdb_target_at_index
#define MDB_KSET_AT_INDEX o_(it)%mdb_kset_at_index
#define MDB_TSLOT_AT_INDEX o_(it)%mdb_tslot_at_index
#define MDB_ABNOB_AT_INDEX o_(it)%mdb_abnob_at_index
#define MDB_MAPOMM_AT_INDEX o_(it)%mdb_mapomm_at_index
#define MDB_IDENT_AT_HDR o_(it)%mdb_ident_at_hdr
#define MLNK_INDEX2HDR o_(it)%mlnk_index2hdr
#define MLNK_HDR2BODY o_(it)%mlnk_hdr2body
#define MLNK_HDR2UPDATE o_(it)%mlnk_hdr2update
#define MLNK_HDR2ERRSTAT o_(it)%mlnk_hdr2errstat
#define MLNK_HDR2SAT o_(it)%mlnk_hdr2sat
#define MLNK_ATOV2ATOV2 o_(it)%mlnk_atovs2atovs
#define MLNK_HDR2REO3 o_(it)%mlnk_hdr2reo3
#define MLNK_HDR2SSMI o_(it)%mlnk_hdr2ssmi
#define MLNK_HDR2SSCATT o_(it)%mlnk_hdr2sscatt
#define MLNK_SAT2ATOV2 o_(it)%mlnk_sat2atovs
#define MLNK_SAT2REO3 o_(it)%mlnk_sat2reo3
#define MLNK_SSMI2SSMI o_(it)%mlnk_ssmi2ssmi
#define MLNK_SSMI2SSCATT BODY o_(it)%mlnk_ssmi2sscatt_body
#define MLNK_SCI2SSCATT BODY o_(it)%mlnk_sci2sscatt_body
#define MDBVNM o_(it)%mdbvnm
#define MBBVCO o_(it)%mbbvco
#define MBBDFL o_(it)%mbbdfl
#define MBDFLD o_(it)%mbdfld
#define MBBSTA o_(it)%mbbsta
#define MBDEVL o_(it)%mbdevl
#define MBDBLE o_(it)%mbdle
#define MBDESQX o_(it)%mbdesq
#define MBDFPQ o_(it)%mbdfp
#define MBDFPML o_(it)%mbdfpl
#define MBDBVAR o_(it)%mbvar
#define MBBASO o_(it)%mbaso
#define MBBOMN o_(it)%mbbomn
#define MBBOMF o_(it)%mbbomf
#define MBDFOE o_(it)%mbfoe
#define MBDFOER o_(it)%mbfere
#define MBDFPER o_(it)%mbfper
#define MBDFPER o_(it)%mbdfper
#define MBDFGE o_(it)%mbdfge
#define MBDFGC1 o_(it)%mbdfgc1
#define MBDFGC2 o_(it)%mbdfgc2
#define MBBJOMO o_(it)%mbbjom0
#define MBDFC1 o_(it)%mbdfc1
#define MBDFC2 o_(it)%mbdfc2
#define MBBJOMN o_(it)%mbbjomn
#define MBDBRVC o_(it)%mbbrvc
#define MBDBPVO o_(it)%mbbrpvo
##define MBDBRVE o_(it)%mbbrve
#define MBDBN2 o_(it)%mbdbv2
#define MBDSYPC o_(it)%mbdsypc
#define MBDELID o_(it)%mbdelid
#defined MBBPILID o_(it)%mbbplid
#define MBDBTORB o_(it)%mbdbtorb
#define MBDBTORAM O_(it)%mbdbta
#define MBDBSRB o_(it)%mbbssrb
#define MBDBS1DVC o_(it)%mbb1dvrc
#define MBDBSCTCF o_(it)%mbbscctcf
#define MBDBSSCBW o_(it)%mbbsscbw
#define MBDBSSANP o_(it)%mbbssanp
# define MBDBSCBAA o_(it)%mbbcsba
#define MDBSCBIA o_(it)%mdbscbia
#define MDBSCKP o_(it)%mdbsckp
#define MDBSCRIF o_(it)%mdbscif
#define MDBSCRIS o_(it)%mdbscis
#define MDBSCDIS o_(it)%mdbscdis
#define MDB_SATID_AT_SAT o_(it)%mdb_satid_at_sat
#define MDB_CLDTOP1_AT_ATOV5 o_(it)%mdb_cldtop1_at_atovs
#define MDB_CLDNE1_AT_ATOV5 o_(it)%mdb_cldne1_at_atovs
#define MDB_CLDTOP2_AT_ATOV5 o_(it)%mdb_cldtop2_at_atovs
#define MDB_CLDNE2_AT_ATOV5 o_(it)%mdb_cldne2_at_atovs
#define MDB_CLDTOP3_AT_ATOV5 o_(it)%mdb_cldtop3_at_atovs
#define MDB_CLDNE3_AT_ATOV5 o_(it)%mdb_cldne3_at_atovs
#define MDBRO3IT o_(it)%mdbro3it
#define MDBRO3PT o_(it)%mdbro3pt
#define MDBRO3LA1 o_(it)%mdbro3la1
#define MDBRO3LO1 o_(it)%mdbro3lo1
#define MDBRO3LA2 o_(it)%mdbro3la2
#define MDBRO3LO2 o_(it)%mdbro3lo2
#define MDBRO3LA3 o_(it)%mdbro3la3
#define MDBRO3LO3 o_(it)%mdbro3lo3
#define MDBRO3LA4 o_(it)%mdbro3la4
#define MDBRO3LO4 o_(it)%mdbro3lo4
#define MDBRO3SOE o_(it)%mdbro3soe
#define MDBRO3FOV o_(it)%mdbro3fov
#define MDBRO3CLC o_(it)%mdbro3clc
#define MDBRO3CP o_(it)%mdbro3cp
#define MDBRO3QL o_(it)%mdbro3ql
#define MDBRO3NL o_(it)%mdbro3nl
#define MDB_SOURCE_AT_DESC o_(it)%mdb_source_at_desc
#define MDB_VERSION_AT_DESC o_(it)%mdb_version_at_desc
#define MDB_ACL_AT_DESC o_(it)%mdb_acl_at_desc
#define MDB_CREADATE_AT_DESC o_(it)%mdb_creadate_at_desc
#define MDB_CREATIME_AT_DESC o_(it)%mdb_creatime_at_desc
#define MDB_CREABY_AT_DESC o_(it)%mdb_creaby_at_desc
#define MDB_MODDATE_AT_DESC o_(it)%mdb_moddate_at_desc
#define MDB_MODTIME_AT_DESC o_(it)%mdb_modtime_at_desc
#define MDB_MODBY_AT_DESC o_(it)%mdb_modby_at_desc
#define MLNK_DESC2HDR o_(it)%mlnk_desc2hdr
#define MLNK_HDR2BUFR o_(it)%mlnk_hdr2bufr
#define MDB_SUBCLASS_AT_HDR o_(it)%mdb_subclass_at_hdr
#define MDB_RDBTYPE_AT_HDR o_(it)%mdb_rdbtype_at_hdr
#define MDB_SUBTYPE_AT_HDR o_(it)%mdb_subtype_at_hdr
#define MDB_BUFRTYPE_AT_HDR o_(it)%mdb_bufrtype_at_hdr
#define MDB_FLAG_AT_HDR o_(it)%mdb_flag_at_hdr
#define MDB_NUMLEV_AT_HDR o_(it)%mdb_numlev_at_hdr
#define MDB_MSG_AT_INBUFR o_(it)%mdb_msg_at_inbufr
#define MDB_THINBOX_AT_HDR o_(it)%mdb_thinbox_at_hdr
#define MDB_MSEC_AT_HDR o_(it)%mdb_msec_at_hdr
#define MDB_SATID_AT_HDR o_(it)%mdb_satid_at_hdr
#define MDB_VARNO_PRESENCE_AT_HDR o_(it)%mdb_varno_presence_at_hdr

!*** New items added due to direct BUFR2ODB conversion.
```c
#define MDB_STATION_TYPE_AT_HDR o_(it)%mdb_station_type_at_hdr
#define MDB_SONDE_TYPE_AT_HDR o_(it)%mdb_sonde_type_at_hdr
#define MDB_UWI_ESA_AT_SCATT o_(it)%mdb_uwi_esa_at_scatt
#define MDB_MPC_AT_SCATT_BODY o_(it)%mdb_mpc_at_scatt_body
#define MDB_WVC_QF_AT_SCATT_BODY o_(it)%mdb_wvc_qf_at_scatt_body
#define MDB_NRETR_AMB_AT_SCATT_BODY o_(it)%mdb_nretr_amb_at_scatt_body
#define MDB_SUBSETNO_AT_HDR o_(it)%mdb_subsetno_at_hdr

/* New items added due to Poolmasking */
#define MLNK_DESC2POOLMASK o_(it)%mlnk_desc2poolmask
#define MDB_POOLNO_AT_POOLMASK o_(it)%mdb_poolno_at_poolmask
#define MDB_OBSTYPE_AT_POOLMASK o_(it)%mdb_obstype_at_poolmask
#define MDB_CODETYPE_AT_POOLMASK o_(it)%mdb_codetype_at_poolmask
#define MDB_SENSOR_AT_POOLMASK o_(it)%mdb_sensor_at_poolmask
#define MDB_TSLOT_AT_POOLMASK o_(it)%mdb_tslot_at_poolmask
#define MDB_SUBTYPE_AT_POOLMASK o_(it)%mdb_subtype_at_poolmask
#define MDB_BUFRTYPE_AT_POOLMASK o_(it)%mdb_bufrtype_at_poolmask
#define MDB_HDR_COUNT_AT_POOLMASK o_(it)%mdb_hdr_count_at_poolmask
#define MDB_BODY_COUNT_AT_POOLMASK o_(it)%mdb_body_count_at_poolmask
#define MDB_MAX_BODYLEN_AT_POOLMASK o_(it)%mdb_max_bodylen_at_poolmask

/* New items added due to introduction of ‘timeslot_index’-table */
#define MDB_TIMESLOT_AT_TIMESLOT_INDEX o_(it)%mdb_timeslot_at_timeslot_index
#define MLNK_DESC2TIMESLOTINDEX o_(it)%mlnk_desc2timeslot_index
#define MLNK_TIMESLOT_INDEX2INDEX o_(it)%mlnk_timeslot_index2index

/* Commonly needed subwords/members of ‘obschar.*@hdr’ */
#define MDB_OBSCHAR_D_CODETYPE_AT_HDR o_(it)%mdb_obschar_d_codetype_at_hdr
#define MDB_OBSCHAR_D_INSTYPE_AT_HDR o_(it)%mdb_obschar_d_instype_at_hdr
#define MDB_OBSCHAR_D_RETRTYPE_AT_HDR o_(it)%mdb_obschar_d_retrtype_at_hdr
#define MDB_OBSCHAR_D_DATAGROUP_AT_HDR o_(it)%mdb_obschar_d_datagroup_at_hdr

/* OBSCHAR is replaced by 4 new entries */
#define MDB_CODETYPE_AT_HDR o_(it)%mdb_codetype_at_hdr
#define MDB_INSTTYPE_AT_HDR o_(it)%mdb_insttype_at_hdr
#define MDB_RETRTYPE_AT_HDR o_(it)%mdb_retrtype_at_hdr
#define MDB_AREATYPE_AT_HDR o_(it)%mdb_areatype_at_hdr

/* Additional @satob entries requested by Niels Bormann [20/11/2001] */
#define MDB_SEGMENT_SIZE_X_AT_SATOB o_(it)%mdb_segment_size_x_at_satob
#define MDB_SEGMENT_SIZE_Y_AT_SATOB o_(it)%mdb_segment_size_y_at_satob

/* New entry(s) */
#define MDB_ZONE_AT_HDR o_(it)%mdb_zone_at_hdr
#define MDB_SATEM_DATAPROC_AT_HDR o_(it)%mdb_satem_dataproc_at_hdr

/* New @satob table item suggested by Niels Bormann [7/11/2002] */
#define MDB_CHAN_FREQ_AT_SATOB o_(it)%mdb_chan_freq_at_satob

/* New @body & @atovs entries proposed by Christina Koepken [26/11/2002] */
#define MDB_LANDSEA_OBS_AT_ATOVS o_(it)%mdb_landsea_obs_at_atovs
#define MDB_CLD_PCT_AT_BODY o_(it)%mdb_cld_pct_at_body
#define MDB_CLD_PCT_AT_ATOVS o_(it)%mdb_cld_pct_at_atovs
#define MDB_CLD_PCT_AT_HDR o_(it)%mdb_cld_pct_at_hdr

/* New @body entries proposed by Tony McNally [29/11/2002] */
#define MDB_RANK_CLD_AT_BODY o_(it)%mdb_rank_cld_at_body
#define MDB_RANK_AN_AT_BODY o_(it)%mdb_rank_an_at_body

/* New items added for storing thinning keys */
#define MDB_THINNINGKEY_AT_HDR o_(it)%mdb_thinningkey_at_hdr
#define MDB_THINNINGTIMEKEY_AT_HDR o_(it)%mdb_thinningtimekey_at_hdr

/* Table @satem removed and items moved into @hdr with different names */
#define MDB_SATEM_DATAPROC_AT_HDR o_(it)%mdb_satem_dataproc_at_hdr
```
#define MDB_SATEM_INSTDATA_AT_HDR o_(it)%mdb_satem_instdata_at_hdr

!*** MDB_AUX1_AT_BODY obsolete (except PREODB)
! Elsewhere replaced by (vector) of MDB_AUX_AT_BODY [23/1/04 SS]
#define MDB_AUX_AT_BODY o_(it)%mdb_aux_at_body
#define MDB_AUX1_AT_BODY o_(it)%mdb_aux1_at_body

!*** latlon_rad@desc added to flag whether (lat,lon) are in radians or not [23/1/04 SS]
#define MDB_LATLON_RAD_AT_DESC o_(it)%mdb_latlon_rad_at_desc

!*** SSM/I reorganization (Peter Bauer/Sami Saarinen) [11/3/04]
#define MDB_PREC_ST_AT_SSMI o_(it)%mdb_prec_st_at_ssmi
#define MDB_PREC_CV_AT_SSMI o_(it)%mdb_prec_cv_at_ssmi
#define MDB_RAIN_AT_SSMI_BODY o_(it)%mdb_rain_at_ssmi_body
#define MDB_SNOW_AT_SSMI_BODY o_(it)%mdb_snow_at_ssmi_body
#define MDB_CLW_AT_SSMI_BODY o_(it)%mdb_clw_at_ssmi_body
#define MDB_CIW_AT_SSMI_BODY o_(it)%mdb_ciw_at_ssmi_body
#define MDB_CC_AT_SSMI_BODY o_(it)%mdb_cc_at_ssmi_body
#define MDB_RAD_COST_AT_SSMI_BODY o_(it)%mdb_rad_cost_at_ssmi_body
#define MDB_RAD_OBS_AT_SSMI_BODY o_(it)%mdb_rad_obs_at_ssmi_body
#define MDB_RAD_FG_DEPAR_AT_SSMI_BODY o_(it)%mdb_rad_fg_depar_at_ssmi_body
#define MDB_RAD_AN_DEPAR_AT_SSMI_BODY o_(it)%mdb_rad_an_depar_at_ssmi_body
#define MDB_RAD_OBS_ERR_AT_SSMI_BODY o_(it)%mdb_rad_obs_err_at_ssmi_body
#define MDB_RAD_BIAS_AT_SSMI_BODY o_(it)%mdb_rad_bias_at_ssmi_body

!*** checksum@hdr-word added to get over reproducibility problems for good [18/3/04 SS]
#define MDB_CHECKSUM_AT_HDR o_(it)%mdb_checksum_at_hdr

$SRC/ifs/common/itdef.h

#define it it_
INTEGER(KIND=JPIM) :: it
it = OML_MY_THREAD()

$SRC/ifs/common/inumtdef.h

#define inumt inumt_
INTEGER(KIND=JPIM) :: inumt
#include "itdef.h"
inumt = OML_MAX_THREADS()
E  ODB source files and binary objects

This appendix gives an overview of the ODB source files, which libraries are involved and how to build standalone ODB object binaries from a source code "tarball" – that is, the ODB source code distribution tarfile.

E.1  ODB source file organization and libraries

ODB software is split into many libraries, some of which are very big in terms of object size (order of tens of megabytes). That is because we want to avoid linking unnecessary libraries together and be able to re-create, say, database libraries on-the-fly from database metadata (see more in xxx).

The following libraries are involved:

<table>
<thead>
<tr>
<th>Library name</th>
<th>Object description</th>
</tr>
</thead>
<tbody>
<tr>
<td>libodbsqlcompiler.a</td>
<td>ODB/SQL-compiler</td>
</tr>
<tr>
<td>libodb.a</td>
<td>The core ODB</td>
</tr>
<tr>
<td>libodbport.a</td>
<td>ODB interface to IFS &amp; BUFR2ODB/ODB2BUFR</td>
</tr>
<tr>
<td>libodbmain.a</td>
<td>ODB-tools and other miscellaneous main programs</td>
</tr>
<tr>
<td>libodbdummy.a</td>
<td>Dummy objects</td>
</tr>
<tr>
<td>libCCMA.a</td>
<td>Database CCMA objects (as built from std source)</td>
</tr>
<tr>
<td>libECMA.a</td>
<td>Database ECMA objects</td>
</tr>
<tr>
<td>libECMASCR.a</td>
<td>Database ECMASCR objects</td>
</tr>
<tr>
<td>libPREODB.a</td>
<td>Database PREODB objects</td>
</tr>
<tr>
<td>libifs.a</td>
<td>IFS</td>
</tr>
<tr>
<td>libifsaux.a</td>
<td>IFSAUX</td>
</tr>
<tr>
<td>libobstat.a</td>
<td>OBSTAT (Observation statistics)</td>
</tr>
<tr>
<td>libobsproc.a</td>
<td>OBSPROC (becoming obsolete)</td>
</tr>
<tr>
<td>libobsort.a</td>
<td>OBSORT (becoming obsolete)</td>
</tr>
<tr>
<td>libmpi_serial.a</td>
<td>Serial (non-parallel) MPI-library</td>
</tr>
<tr>
<td>libemos.a</td>
<td>64-bit real, 32-bit integer EMOSLIB (000240)</td>
</tr>
<tr>
<td>libec.a</td>
<td>64-bit real, 32-bit integer ECLIB (y2k timer only)</td>
</tr>
</tbody>
</table>

Also source code selected for these libraries follows certain conventions which should not be broken. The following table lists the libraries.
Table 50: ODB source codes and dependencies per library

<table>
<thead>
<tr>
<th>Library name</th>
<th>Source files</th>
<th>Dependency directories ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>libodbsqlcom timber.a</td>
<td>compiler/<em>.l , compiler/</em>.y , compiler/*.c</td>
<td>$SRC/odb/include</td>
</tr>
<tr>
<td>libodb.a</td>
<td>module/<em>.F90 , lib/[a-z]</em>.F90 , aux/[a-z]<em>.F90 , aux/[a-z]</em>.c</td>
<td>$SRC/odb/include , $SRC/odb/module , $SRC/odb/iface</td>
</tr>
<tr>
<td>libodbport.a</td>
<td>cma2odb/<em>.F90 , bufr2odb/</em>.F90 , preodb/[a-z]*.F90</td>
<td>As for libodb.a plus: $SRC/odb/include , $SRC/odb/module , $SRC/odb/iface</td>
</tr>
<tr>
<td>libodbmain.a</td>
<td>tools/[A-Z]<em>.F90 , tools/</em>.c</td>
<td>As for libodb.a and libodbport.a</td>
</tr>
<tr>
<td>libodbdummy.a</td>
<td>lib/[A-Z]<em>.F90 , lib/[A-Z]</em>.c</td>
<td>As for libodb.a</td>
</tr>
</tbody>
</table>

Table 51: Source codes & dependencies per permanent (recognized) database libraries

<table>
<thead>
<tr>
<th>Library name</th>
<th>Source files</th>
<th>Dependency directories ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>libCCMA.a</td>
<td>ddl.CCMA/CCMA.ddl , ddl.CCMA/*.sql</td>
<td>$SRC/odb/ddl.CCMA</td>
</tr>
<tr>
<td>libECMA.a</td>
<td>ddl.ECMA/ECMA.ddl , ddl.ECMA/*.sql</td>
<td>$SRC/odb/ddl.ECMA</td>
</tr>
<tr>
<td>libECMASCR.a</td>
<td>ddl.ECMASCR/ECMASCR.ddl , ddl.ECMASCR/*.sql</td>
<td>$SRC/odb/ddl.ECMASCR</td>
</tr>
<tr>
<td>libPREODB.a</td>
<td>ddl.PREODB/PREODB.ddl , ddl.PREODB/*.sql</td>
<td>$SRC/odb/ddl.PREODB</td>
</tr>
</tbody>
</table>

The next table is valid for a standalone ODB build only. There we build also some additional libraries (like EMOS and ECLIB), since they may not be available, accessible or up-to-date.
E.2 Building ODB binaries

ODB can be distributed and built without building the whole libifs.a. We still, however, need the prerequisite libraries like libifsaux.a and some selected modules from libifs.a. Also some crucial setup routines are still needed from libraries libobsproc.a and libobsort.a, but this is subject to change in the near future.22

How to build ODB software from a tarfile?

Assuming you have gunzip’ped the ODB source code distribution tarfile (say, the file odb_CY28R2.tar.gz). Copy it to the place where you have some 0.5GB of disk space and untar it. Then run configure and make as follows (assuming Korn-shell):

```
$ cp odb_CY28R2.tar.gz /your/disk
$ cd /your/disk
$ gunzip < odb_CY28R2.tar.gz | tar -xf -
$ cd odb_CY28R2
$ export ARCH=xxxx
$ ./configure
$ make 2>&1 | tee log.make
```

If you had run this in C-shell (csh or tcsh), the to achieve the same log-file, the last command should be:

```
% make & tee log.make
```

where xxxx is ARCH-name as described in table below. Also, please have a look at odb/scripts/make. for $ARCH xxxx.

Be aware that compilation may take well over one hour unless you lower optimization levels.
The label "CY28R2" may vary, depending on shipped version or included features.
You do not need EMOSLIB & ECLIB from ECMWF to satisfy all external references, since there are all the necessary entries found under odb/extras/.
The same holds for MPI, if you don’t have one, the odb/extras/mpi_serial can be "kicked in" (change is possibly needed in file odb/scripts/make.xxxx).

---

22Almost all the necessary functions libobsproc.a have already been moved to libifs.a and observation libobsort.a awaits for similar "treatment" – that is, some source code from OBSORT will be moved under ODB.
For one plotting utility you need MAGICS from ECMWF – or you just rely on default dummies (=> no plots produced).

Before running ./configure, make sure your ARCH setting is set to one of the following (B=Big Endian, L=Little Endian):

<table>
<thead>
<tr>
<th>ARCH</th>
<th>Platform</th>
<th>Endian</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs6000</td>
<td>IBM RS6000 Power3</td>
<td>B</td>
<td>Tested</td>
</tr>
<tr>
<td>ibm_power4</td>
<td>IBM RS6000 Power4</td>
<td>B</td>
<td>Tested</td>
</tr>
<tr>
<td>linux</td>
<td>Pentium/Linux</td>
<td>L</td>
<td>Tested, byte swap ok.</td>
</tr>
<tr>
<td>linuxg95</td>
<td>Pentium/Linux with Gnu95</td>
<td>L</td>
<td>Compiles, doesn’t link yet</td>
</tr>
<tr>
<td>sgimips</td>
<td>SGI MIPS R10000</td>
<td>B</td>
<td>Compiles</td>
</tr>
<tr>
<td>vpp5000</td>
<td>Fujitsu VPP5000</td>
<td>B</td>
<td>To be tested</td>
</tr>
<tr>
<td>hppa</td>
<td>HP-UX 11</td>
<td>B</td>
<td>Compiles, doesn’t link yet</td>
</tr>
<tr>
<td>decalpha</td>
<td>Compaq/Dec Alpha</td>
<td>L</td>
<td>Compiles, but needs /bin/sh -&gt; /bin/ksh</td>
</tr>
</tbody>
</table>

At the moment ODB has not been thoroughly tested on all platforms. To add a new platform, you need to add a file "make.<platform>" under odb/scripts-directory. See for example the files make.linux, make.rs6000 or make.ibm_power4.

The ODB software was originally developed on SGI-system (sgimips) and it ran many years on Fujitsu VPP5000 & VPP700(E) vector-machines at ECMWF, so we have available their file ("make.<platform>") as well, but we cannot (anymore) test them properly.

The ODB software should also run okay on Sun Solaris, HP-UX, Dec/Compaq Alpha and on the new Cray SV2-systems, and NEC SX (with some tweaking).

All systems must be either big or little endians and follow the IEEE-754 floating point arithmetic standard.

Installation directory

For the ODB administrator, it is preferred to do the following when installing the standalone ODB software:

```bash
$ export ARCH=xxxx
$ export ODB_DIR=/your/odb/root/path
$ export ODB_VERSION=mytest
$ make INSTALL_DIR=$ODB_DIR/$ARCH/$ODB_VERSION install
```

As result you will have an ODB installation 'mytest' for machine 'xxxx' under directory $ODB_DIR/$ARCH/$ODB_VERSION. Under this directory you will see subdirectories: lib/, bin/, include/ and module/. They contain libraries, binaries and scripts, various include files and F90 module files, respectively.
E.2 Building ODB binaries

Environment initialization

Things could be wrapped into a script. But in essence the following things must be set to gain access to the installed software (note: you may have to modify scripts use_odb.sh/use_odb to suit local requirements):

```
$ export ARCH=xxxx
$ export ODB_DIR=/your/odb/root/path
$ export ODB_VERSION=mytest
$ . $ODB_DIR/$ARCH/$ODB_VERSION/bin/use_odb.sh
```

or in C-shell (csh/tcsh):

```
% setenv ARCH xxxx
% setenv ODB_DIR /your/odb/root/path
% setenv ODB_VERSION mytest
% source $ODB_DIR/$ARCH/$ODB_VERSION/bin/use_odb
```

Running examples

(This will be completed later)
F  ODB limitations

There are a few limitations in ODB that user needs to be aware of.

F.1  Column entry size

Each column entry can hold a maximum 8-bytes of information. This means that any attempt to store character strings greater than 8 bytes per element are subject to failure. In such circumstances multiple column entries must be allocated, or character (or byte) data needs to be spread across multiple rows. The latter is bad for executing SQL. However BUFR-data – 4 or 8 bytes per element for example – can safely be spread across multiple rows, since the SQL’s WHERE-condition practically never needs to be applied to BUFR-data itself.

F.2  Maximum number of rows per request

A serious limitation could be hit in practice\textsuperscript{23}. The control ($0^{th}$) column of data matrix in \textit{ODB_get} is a made up of compound numbers which must accomodate the pool number and row number from where data was extracted. For increasing the number of pools (say $\gg 100$, like at ECMWF), the maximum number of rows per request that fit into the control column gets smaller and smaller, being approximately $2147483647$ divided by the largest possible pool number.

In future releases this limitation will be fixed by allowing functions \textit{ODB_get} and \textit{ODB_put} not to operate with other than \texttt{REAL(8)} data matrices. This gives 63-bits rather than 31-bits for the control column information.

F.3  Character string limit

The number of bytes in a single cell (an element of a column) is exactly 8 bytes. This limitation is likely to stay like this, since functions \textit{ODB_get} and \textit{ODB_put} can deal at most with 8-byte elements (cells). If you need more than eight bytes for your character string, you can specify multiples of eight as follows:

\begin{verbatim}
CREATE TABLE tbl (  
str[10] string , //  
);
\end{verbatim}

\textsuperscript{23}Before ODB software is changed
This definition will create 10 string-columns (fields from str_1@tbl to str_10@tbl), each having 8 bytes for character data. And yet functions ODB_get and ODB_put work as usual.
G  Data sorting

G.0.1  KEYSORT

Usage of the subroutine KEYSORT is as follows:

```
CALL KEYSORT ( RC, A, N, [KEY], [MULTIKEY] &, &
                [METHOD], [INDEX], [INIT], [TRANSPOSED])
```

This is ECMWF's multi-key sort, which is using fast and vectorizable radix-sort method for reordering the data. This function is used everywhere in IFS what comes to observation ordering. It is also the cornerstone for performing ORDERBY-statement in ODB/SQL. Source code is located in the module file $SRC/ifsaux/module/ecsor.F90. To use this function you need to issue the following statement in your Fortran90 program:

```
USE ecsort
```

This function uses rsort32()-function underneath to perform the fast sorting. This utility is located under $SRC/ifsaux/utilities/rsort32.c and duplicated in file $SRC/odb/lib/rsort32_odb.c. The parameters denote:

**RC**

INTEGER(4), INTENT(INOUT)
Return code, which upon success should return n

**A(:,,:),A(:,:,):**

REAL(4), INTENT(INOUT)
The array or vector to be sorted. If optional argument index is present, then physical reordering is not performed.

**N**

INTEGER(4), INTENT(IN)
Number of rows to be sorted, when transposed-parameter is not given or is set to (default) value .FALSE.. In this case must be \( \leq \) leading dimension of array a. If transposed-parameter is set to .TRUE., then denotes number of columns. In this case must be \( \leq \) than number of columns in array a.

**KEY**

INTEGER(4), INTENT(IN), OPTIONAL
The column number of array a to be used as primary key for sorting (when transposed=.FALSE.). (if transposed=.TRUE., then the row number of a). The value must be either positive number between 1...n and sorting will be ascending, or negative number between \(-n\ldots-1\) and sorting will return a descending sequence. This parameter is mutually exclusive with parameter multikey and they should not be used simultaneously. Also, if array a is a vector, then use of this parameter is not applicable and will result an error.
MULTIKEY(:) INTEGER(4), INTENT(IN), OPTIONAL
This parameter should be used when multiple sorting keys (columns) needs to be applied to array a. The first value multikey(1) denotes the most significant key and the last element denotes the least significant key. Negative numbers are also accepted, as with parameter key. Sorting is performed in a series of calls to rsort32, starting from least significant column down to the most significant key. No check is performed whether the same key is supplied more than once. This parameter is mutually exclusive with parameter key and they should not be used simultaneously. Also, if array a is a vector, then use of this parameter is not applicable and will result an error.

METHOD INTEGER(4), INTENT(IN), OPTIONAL
Change of sorting method. Applicable only for the situation where array a is a vector. Possible choices are (1) radix sort (the default) or (2) heap sort. The multikey-sorting always uses radix sort.

INDEX(:) INTEGER(4), INTENT(INOUT), TARGET, OPTIONAL
Instead of performing physical sort of array a, the routine can return just index to the sorted order. This parameter is initialized with values 1...n, if parameter init is not set (or is set to .TRUE.). If init is set to .FALSE., then user can supply her own initial ordering (values 1...n, but in predefined order). This is useful if one wants to perform a series of keysort-calls after each other, each changing the index-vector.

INIT LOGICAL, INTENT(IN), OPTIONAL
When this parameter is not present or present and set to .TRUE., it will initialize index-vector (if present) with values 1...n.

TRANSPOSED LOGICAL, INTENT(IN), OPTIONAL
You can supply array a as a transposed array without actually transposing it first. This swaps the meaning of rows and columns with each other. The end result is the same. Since this is by far much less economical way of accessing data (stride is now leading dimension of a), it is not recommended to set this parameter to .TRUE.. The default value is .FALSE.
H Dr.Hook instrumentation tool

Dr.Hook is a simple, low-overhead instrumentation tool, which allows you to keep track of
dynamic calling tree of a program and print it in the event of failure. It can also gather perform-
ance profiling information on a per subroutine basis, which can be useful in estimating
computational costs.

The name Dr.Hook stems from Fujitsu VPP’s hook-function – a compiler option which al-
lowed routines to be intercepted upon entry and exit by a user defined "hook"-function. The
Dr.Hook-environment, unlike Fujitsu’s system, is meant to be portable across all Unix-
systems, controlled by environment variables without need to recompile or relink.

In the recent years it has become more difficult to get reliable traceback upon failure. One
reason is our complex environment and use of multiple MPI-tasks and OpenMP-threads.
Error messages from such processes are often lost or can be quite misleading. As result
much extra time has to be put into debugging. We wanted to reduce this laborious debugging
and instead (or in addition to) offer a more direct way of trapping errors.

Since the CY28 of IFS was a cleaning cycle, it was decided to insert automatically calls
to DR_HOOK() -function while entering and leaving any IFS/ARPEGE related subroutine,
a few thousand in total. This way the Dr.Hook-environment could keep track of dynamic
calling tree and upon failure produce informative Dr.Hook-traceback.

As result of these calls we can hopefully fix programming errors much quicker than before
without need to add extra print statements or wonder for several days what may have gone
wrong.

A nice "by-product" of Dr.Hook-instrumentation is that we can also collect profiling infor-
mation on resources we spend in each instrumented routine. That is to say, we can collect
routine call-counts, wall & CPU-times, chase for memory peaks, check paging activities and
so on. Furthermore we can produce a user-friendly report at the end of execution per each
MPI-task spread across every computational OpenMP-thread, if necessary.

A recent addition (CY28R1) to Dr.Hook system is possibility to obtain MFlop/s-rates per
OpenMP-thread and MPI-task on the IBM Power4 machine. This greatly helps us to under-
stand computational costs in IFS/ARPEGE and may even aid benchmarking.

Dr.Hook is currently callable from both Fortran90 and C-routines.

Instrumentation

Every instrumented routine (here: just a generic SUBR) has the following coding norm:

- at the top of the routine: enable access to YOMHOOK-module

USE YOMHOOK, ONLY : LHOOK, DR_HOOK
at the beginning of the routine (before the first executable statement)

```fortran
REAL(KIND=JPRB) :: ZHOOK_HANDLE
IF (LHOOK) CALL DR_HOOK('SUBR',0,ZHOOK_HANDLE)
```

just before END- or before any RETURN- or CONTAINS-statements you must call

```fortran
IF (LHOOK) CALL DR_HOOK('SUBR',1,ZHOOK_HANDLE)
```

The first call to \texttt{DR\_HOOK()} adjusts the current dynamic calling tree by adding a new member to it. The last call to \texttt{DR\_HOOK()} removes routine ‘\texttt{SUBR}’ from this instantaneous calling tree.

If in addition performance profiling is on, then the Dr.Hook-system collects exact (as opposed to sampled) performance profile on behalf of routine ‘\texttt{SUBR}’ and its descendants while in this routine.

If the \texttt{SUBR} is a Fortran90 module procedure, then the name will contain the MODULE-name followed by a colon before the routine name (for example: ‘\texttt{GFL\_SUBS:DEFINE\_GFL\_COMP}’).

It is important to note that the variable \texttt{ZHOOK\_HANDLE} must be a local (stack) variable – i.e. not a \texttt{SAVE}d variable. Its size has to be 8-bytes and the subroutine prototype requires it to be a \texttt{REAL(8)} floating point number. Upon return from the first \texttt{DR\_HOOK()} -call it contains the address information to the Dr.Hook internal data structure, where accounting information for this particular routine (and of this OpenMP-thread) is kept. The last call to \texttt{DR\_HOOK()} in the routine will use the value of variable \texttt{ZHOOK\_HANDLE} as an address to locate and update the routine’s accounting records.

If \texttt{ZHOOK\_HANDLE} gets overwritten between these two calls to \texttt{DR\_HOOK()}, then the second call will pick this it up immediately and raise an abort. This is an indication of either a genuine (stack variable) overwrite, or more often that some descendant routine has \texttt{RETURN}ed before calling second time its \texttt{DR\_HOOK()}. The latter case is easy to fix: just search for alternative exits and \texttt{RETURN}-statements from the descendant routines and check whether any path to the second call of \texttt{DR\_HOOK()} exists at all.

The logical variable \texttt{LHOOK} is by default set to \texttt{.TRUE.} in module \texttt{YOMHOOK}. When the very first call to \texttt{DR\_HOOK()} takes place in a program, then all Dr.Hook-environment variables are examined. If it turns out that Dr.Hook-facility was not turned on, then the logical variable \texttt{LHOOK} is set to \texttt{.FALSE.} and any further dynamic calling tree processing will be disabled.

In the future there will be available a checker-program to ensure that \texttt{DR\_HOOK()} calls are entered correctly.
Environment variables

Environment variables are used to activate and drive Dr.Hook features. By default – if no variables are set – Dr.Hook facility is turned off. All values (except filenames) are case insensitive in the following table.

Table 52: Dr.Hook environment variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Values &amp; remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR_HOOK</td>
<td>Enable/disable Dr.Hook</td>
<td>true or 1 activates Dr.Hook false or 0 is the default</td>
</tr>
</tbody>
</table>
| DR_HOOK_OPT    | A comma-separated list of:| Traceback contains call counts Traceback contains CPU-times Traceback contains wall clock times Times or time Union of cputime and walltime Heap or hwm Heap memory high water mark per routine Stack or stk Stack size monitoring per routine Rss Resident set size monitoring Paging or pag Paging activity monitoring per routine Memory or mem Union of heap, stack, rss & paging All Union of memory and times Prof or wallprof Activates wall clock time based profiling Imply also calls and walltime Deactivates cpuprof Cpuprof Activates CPU-time based profiling Imply also calls and cputime Deactivates wallprof Hpmprof or hpm or mflops Activates MFlop/s profiling Imply wallprof Trim Allow case insensitive input of routine name Very expensive! Don’t use this! Self Include Dr.Hook in the profile output, too Noself Exclude Dr.Hook altogether from the profile By default Dr.Hook overhead is always, 

*continued on next page*
Table 52: Dr.Hook environment variables …continued

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Values &amp; remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>nosize or nosizeinfo</td>
<td>noprop or npropagate</td>
<td>accounted for, but not printed in the profile output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Do not display size-information</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Do not propagate signals outside Dr.Hook</td>
</tr>
<tr>
<td>cluster or clusterid</td>
<td></td>
<td>Display cluster-id information</td>
</tr>
<tr>
<td>DR_HOOK_HPMSTOP</td>
<td>Stops HPM-counting for the given routine</td>
<td>Set this to value</td>
</tr>
<tr>
<td></td>
<td>after ncalls if mflops-rate has not been achieved</td>
<td>ncalls or ncalls,mflops</td>
</tr>
</tbody>
</table>

Currently the options DR_HOOK_OPT="stack,paging,mflops" are only available for IBM Power4 machines, but ports to Cray X1 (for example) are underway. As of writing this document, Cray X1 hardware performance monitor for obtaining MFlop/s-rates has just been implemented.

In addition the following environment variables are also available to Dr.Hook:

Table 53: More Dr.Hook environment variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Values &amp; remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR_HOOK_PROFILE</td>
<td>Filename(s) for the profile</td>
<td>One profile-file per MPI-task</td>
</tr>
<tr>
<td></td>
<td>Use full path with .%d for MPI-task: /path/file.%d,</td>
<td>The default: drhook.prof.%d If the suffix (.%d) is missing, it will be added</td>
</tr>
<tr>
<td></td>
<td>%d = 1..$NPES $</td>
<td>automatically</td>
</tr>
<tr>
<td></td>
<td>The default = -1 i.e. all MPL-tasks</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The default = -10.0 (!)</td>
<td></td>
</tr>
<tr>
<td>DR_HOOKPROFILEPROC</td>
<td>MPL-task that produces profile</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Routines that consume at least this many percentages of (self-time resources will appear in the profile</td>
<td></td>
</tr>
<tr>
<td>DR_HOOKPROFILELIMIT</td>
<td>A comma separated list of additional signals to be caught</td>
<td>The default = 0 i.e. no effect. Value = -1 activates all possible signals</td>
</tr>
<tr>
<td></td>
<td>A comma separated list of signals not to be caught</td>
<td>The default = 0 i.e. no effect. Value = -1 deactivates all possible signals</td>
</tr>
</tbody>
</table>

continued on next page…
Table 53: More Dr.Hook environment variables …continued

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Values &amp; remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR_HOOK_HASHBITS</td>
<td>No. bits for hashing algorithm</td>
<td>The default = 15 i.e. allocates a hash data structure of $2^{15}$ elements. Between 1..24. Increase only if you have thousands of routines to monitor</td>
</tr>
</tbody>
</table>

**Caught signals**

The following table shows what Unix-signals are caught by Dr.Hook-system. Please note that signal **SIGXPU** (i.e. CPU-time exceeded) does not produce any profiling output, since it would cause Dr.Hook to catch **SIGXPU** infinitely, since there is no CPU-time left! Otherwise the profile is written even if a signal has been caught, since we believe it still can contain valuable information.

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Description</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGABRT</td>
<td>Abort execution</td>
<td></td>
</tr>
<tr>
<td>SIGBUS</td>
<td>Bus error</td>
<td></td>
</tr>
<tr>
<td>SIGSEGV</td>
<td>Segmentation violation</td>
<td></td>
</tr>
<tr>
<td>SIGILL</td>
<td>Illegal instruction</td>
<td></td>
</tr>
<tr>
<td>SIGEMT</td>
<td>EMT instruction</td>
<td></td>
</tr>
<tr>
<td>SIGSTKFLT</td>
<td>Stack fault</td>
<td></td>
</tr>
<tr>
<td>SIGFPE</td>
<td>Floating-point exception</td>
<td></td>
</tr>
<tr>
<td>SIGTRAP</td>
<td>Trace trap</td>
<td></td>
</tr>
<tr>
<td>SIGINT</td>
<td>Interrupt</td>
<td></td>
</tr>
<tr>
<td>SIGQUIT</td>
<td>Quit</td>
<td></td>
</tr>
<tr>
<td>SIGTERM</td>
<td>Termination</td>
<td></td>
</tr>
<tr>
<td>SIGIO</td>
<td>I/O now possible</td>
<td></td>
</tr>
<tr>
<td>SIGXCPU</td>
<td>CPU limit exceeded</td>
<td></td>
</tr>
<tr>
<td>SIGSYS</td>
<td>Bad system call</td>
<td></td>
</tr>
</tbody>
</table>

For actual signal numbers, please refer to your `man signal-command` or have a look at your Unix-system’s include file `/usr/include/signal.h` (often in `/usr/include/sys/signal.h`). These are needed if you want to use environment variables `DR_HOOK_CATCH_SIGNALS` or `DR_HOOK_IGNORE_SIGNALS` to activate/deactivate some signals.

Very often catching a **SIGSEGV** means serious memory overwrite, a **SIGBUS** that some subroutine argument is missing or stack is corrupted, a **SIGILL** means that illegal instruction is executed, a **SIGFPU** means a floating point arithmetic on uninitialized or invalid numbers, a **SIGINT** means that control-C is pressed (for interactive jobs), and so on.
Overhead

When the dynamic calling tree is activated (but nothing else) the overhead of DR_HOOK() - calls seem to be around 1% in the current IFS/ARPEGE configurations on ECMWF’s IBM Power4. This seems to be a low price to pay for enabling correct tracebacks, and thus reducing need for extensive debugging and head-lice scratching!

When the wall clock time profiling is on, overheads seem to be around 2%. And if the MFlop/s-rate counters are activated, the overhead is still bearable 10-15%. And all these extra profiling options can be turned on via environment variables, without need to recompile or relink anything. Please note that the HPM-overhead can now be effectively minimized by setting environment value DR_HOOK_HPMSTOP to (say) 20,500000. This means: stop calling HPM-counters for any given routine after 20 calls unless routines performance exceeds 500000 Mflop/s (500GigaFlop/s!).

Some auxiliary routines

You can activate Dr.Hook’s signal catching feature to be able to produce at least system specific traceback more reliably by calling routine C_DRHOOK_INIT_SIGNALS():

```fortran
CALL C_DRHOOK_INIT_SIGNALS(1)
```

This instructs Dr.Hook’s signal handling functions to be on the top of the list of the functions to be called in the event of failure. And if Dr.Hook-facility was turned off you will not get its dynamic calling tree, but a system specific traceback, which may or may not be accurate. See for example ifs/setup/sumpin.F90.

When Dr.Hook has been activated, you can print an instantaneous calling tree at any moment to a Fortran I/O-unit by calling routine C_DRHOOK_PRINT():

```fortran
INTEGER(4) :: IOUNIT, ITID, IOPT, INDENT
INTEGER(4),EXTERNAL :: get_thread_id
IOUNIT = 0   ! Fortran I/O-unit, say stderr
ITID = get_thread_id( ) ! 1 .. # of OpenMP-threads
IOPT = 2     ! Print current calling tree
INDENT = 0   ! Indentation; modified during the call
CALL C_DRHOOK_PRINT(IOUNIT, ITID, IOPT, INDENT)

! The variable INDENT now equals to no. of routines seen in the traceback
```

C-interface

Also C-routines can be instrumented under Dr.Hook-system. The interface (in CY28R2) to C is as follows:
#include "drhook.h" /* from "ifsaux/include/drhook.h" */

void subname()
{
    /* Variable declarations */

    { /* Begin block (optional) */
        /* The following is like the first call to DR_HOOK() in Fortran90 */

        DRHOOK_START(subname);

        /* or as a constant (compile-time evaluable) character string:
        DRHOOK_START_BY_STRING("subname"); */

        /* The first actual executable statement */

        /* The body of the routine "subname" goes here */

        /* The last thing;
        the zero (0) below can be replaced with some size-information,
        like the number of bytes processed or so */

        DRHOOK_END(0);
    } /* End block (optional, if the Begin block also omitted) */
    return;
}

You can see examples in files odb/lib/codb.c and odb/include/codb.h. Note that symbols DRHOOK_START, DRHOOK_START_BY_STRING and DRHOOK_END are all simple and cheap C-macros, which take care of declaring the Fortran90 _HOOK_HANDLE-equivalent inside the macros.

**Source code and libraries**

In CY28R2, Dr.Hook resides in IFSAUX, files ifsaux/support/drhook.c and include file ifsaux/include/drhook.h and is therefore naturally built into library libifsaux.a. The Fortran90-part resides in files ifsaux/module/yomhook.F90 and ifsaux/support/dr_hook_*.F90.

Timers and memory routines needed are found mostly under the ifsaux/utilities/-directory. Some other resource related routines are still found (for historical reasons) under odb/aux/util_ccode.c, but can also be found in file ifsaux/utilities/util_timers.c in CY28R2.
Dummies for HPM (High Performance Monitoring) for IBM are found under odb/lib/Dummies.c, in case you hit problems with unsatisfied externals.

On an IBM Power4 machine in order to activate the HPM-feature, you need to compile the file ifsaux/support/drhook.c with option -DHPM and link executable either with dummies (-lodbdummy i.e. no runtime HPM) or with HPM-libraries (at ECMWF called $LIBHPM)

- L/usr/pmapi/lib -lpmapi.

**Enhancements**

After CY28R2 we will investigate ways to incorporate profiling of data movement. That is to say, we want to get clearer picture on amounts of data transferred across MPI-processes and to be able to trace I/O-bottlenecks. This has been partially done via GSTATS-subroutine (in IFSAUX).

Furthermore we want to be able to show the filename of the routine being called. This is important especially for C-routines, where there maybe several routines sitting the same file making searching process more difficult.

The Fortran90-interface to DR_HOOK() contains two more arguments: size information (an INTEGER(4)) and the (source) filename parameters. In fact, the C-interface in cycle CY28R1 already contains this functionality and some ODB C-routines already benefit from these two extra arguments. As an example, you could pass the number of bytes retrieved from a database, or decide that you want to see how many rows of data satisfy a particular SQL-query condition. In both cases you could also opt for source filename of the routine in concern.

The basic Dr.Hook without fancy MFlop/s-monitors, but with traceback feature, will run at least on the following machines in CY28R2: IBM Power4 & Power3, Pentium/Linux and Silicon Graphics. We still have to check how Fujitsu VPP5000 version will behave. Port to Cray X1 is nearly completed and it is now already possible to easily compare this machine’s performance with IBM Power4 rather realistically.

**Known bugs or features**

Dr.Hook-system cannot handle recursive function calls correctly i.e. profiling information for such routines (and calling tree before the routine) will be incorrect. This will be fixed in the future. Meantime, remove Dr.Hook from potentially recursive routines.

When a failure occurs on a non-master OpenMP-thread (i.e. > 1), the dynamic calling tree incorrectly prints the full current calling tree of the master-thread before proceeding to print thread’s own tree. Although this doesn’t matter too much, you will still see slightly mislead-
DRAFT : 1st edition

H : Dr.Hook instrumentation tool [Version: 7th May 2004]

257

ing traceback and duplicate output of some routines that were active in both master and slave
thread during failure.
The signal SIGIOT was mistakenly typed as SIGIO. The SIGIOT normally equals to SIGABRT. If you belive that this incorrect coding causes problems and you want to remove SIGIO
(normally #29) from the list of Dr.Hook-catchable signals, you can ignore this signal via environment variable as follows (Korn-shell):
$ export DR_HOOK_IGNORE_SIGNALS=29
This bug has now been fixed in CY28R2.
Acknowledgements
Thanks to Bob Walkup from IBM Watson Research for information about how to read the
HPM-counters on IBM Power4 machine.
Thanks to Bob Carruthers from Cray UK for supplying HPM-counters for Cray X1 machines.
Examples of Dr.Hook output from IFS:


## Dr. Hook for T511 forecast

<table>
<thead>
<tr>
<th>#</th>
<th>Time (sec)</th>
<th>Dwell (sec)</th>
<th>Total (sec)</th>
<th># of calls</th>
<th>MID</th>
<th>MILF</th>
<th>Div*</th>
<th>Routing (octet)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>22.12</td>
<td>147.001</td>
<td>147.001</td>
<td>187.794</td>
<td>2190</td>
<td>371</td>
<td>4.3</td>
<td>018F0006010</td>
</tr>
<tr>
<td>2</td>
<td>22.12</td>
<td>147.001</td>
<td>146.696</td>
<td>126.032</td>
<td>112200</td>
<td>370</td>
<td>4.2</td>
<td>018F0006010</td>
</tr>
<tr>
<td>3</td>
<td>22.99</td>
<td>147.001</td>
<td>146.249</td>
<td>126.072</td>
<td>112006</td>
<td>370</td>
<td>4.2</td>
<td>018F0006010</td>
</tr>
<tr>
<td>4</td>
<td>22.99</td>
<td>147.001</td>
<td>146.616</td>
<td>126.762</td>
<td>112784</td>
<td>371</td>
<td>4.3</td>
<td>018F0006010</td>
</tr>
<tr>
<td>5</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>6</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>7</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>8</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>9</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>10</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>11</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>12</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>13</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>14</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>15</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>16</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>17</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>18</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>19</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>20</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>21</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>22</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>23</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>24</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>25</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>26</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
<tr>
<td>27</td>
<td>3.28</td>
<td>234.235</td>
<td>26.739</td>
<td>26.739</td>
<td>371</td>
<td>370</td>
<td>4.3</td>
<td>01F00006010</td>
</tr>
</tbody>
</table>
## Glossary

This appendix provides a short description of acronyms used. Some of them are not directly used in the document, but are closely related to high performance computing, databases, observation processing, satellites or 4D-Var analysis – thus worth mentioning.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D-Var</td>
<td>One Dimensional Variational Assimilation (usually the vertical profile).</td>
</tr>
<tr>
<td>3D-FGAT</td>
<td>3D-Var with first guess at appropriate time.</td>
</tr>
<tr>
<td>3D-Var</td>
<td>Three Dimensional Variational Assimilation.</td>
</tr>
<tr>
<td>4D-Var</td>
<td>Four Dimensional Variational Assimilation (includes time as fourth dimension).</td>
</tr>
<tr>
<td>ACARS</td>
<td>Aircraft Communication Addressing and Reporting System.</td>
</tr>
<tr>
<td>Active data</td>
<td>All data that contribute (have a non-zero weight) in the analysis.</td>
</tr>
<tr>
<td>AD</td>
<td>Adjoint (the transpose of the tangent-linear operator).</td>
</tr>
<tr>
<td>AIREP</td>
<td>Aircraft weather report.</td>
</tr>
<tr>
<td>AIRS</td>
<td>Atmospheric Infrared Sounder.</td>
</tr>
<tr>
<td>AMDAR</td>
<td>Aircraft Meterological Data Acquisition and Relay.</td>
</tr>
<tr>
<td>AMSU (-A, -B)</td>
<td>Advanced Microwave Sounding Unit (AMSU–A, AMSU–B).</td>
</tr>
<tr>
<td>AMV</td>
<td>Atmospheric Motion Vectors (SATOBS).</td>
</tr>
<tr>
<td>Analysis variables</td>
<td>.</td>
</tr>
<tr>
<td>AQUA</td>
<td>NASA’s Earth Observation system (EOS) mission to study water vapour and cloud.</td>
</tr>
<tr>
<td>ARPEGE</td>
<td>Action de Recherche Petite Echelle Grande Echelle (a collaboration with ECMWFs IFS code development).</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange.</td>
</tr>
<tr>
<td>ATOVS</td>
<td>Advanced TOVS.</td>
</tr>
<tr>
<td>Background</td>
<td>Prior information, often in the form of climatology or (better) a short-range forecast.</td>
</tr>
<tr>
<td>BATHY</td>
<td>Report of bathythal observation.</td>
</tr>
<tr>
<td>Bias</td>
<td>Mean error.</td>
</tr>
<tr>
<td><strong>Bit</strong></td>
<td>Stands for binary digit, or a number that can have only two values: 0 or 1.</td>
</tr>
<tr>
<td><strong>Blacklist</strong></td>
<td>A set of instructions defining which observations not to use.</td>
</tr>
<tr>
<td><strong>BUFR</strong></td>
<td>WMO Code for Binary Universal Form for data Representation.</td>
</tr>
<tr>
<td><strong>BUFR2ODB</strong></td>
<td>A conversion software package to translate BUFR-data into ODB-format (database ECMA in particular) prior to 4D-Var-analysis. See also ODB2BUFR.</td>
</tr>
<tr>
<td><strong>Byte</strong></td>
<td>Eight bits and can hold a value from 0 to 255 (i.e. $2^8 - 1$), or -128 to 127.</td>
</tr>
</tbody>
</table>

| **CCMA** | ODB-database name for used observations (Compressed Central Memory Array). |
| **/ccvobs** | Root directory for files under ClearCase version handling system at ECMWF. |
| **ClearCase** | Software version handling system in use at ECMWF to manage source code modifications. |
| **COH** | Comprehensive Observation Handling. See as/see also MAKECMA. |
| **Control variable** | The set of variables that are adjusted (analyzed) during variational assimilation. |
| **Cost function** | The "objective function" measuring the distance (or misfit) to observations and prior information. |
| **CPU** | Central Processing Unit. |

| **DDL** | Data Definition Language or Layout. |
| **Departure** | The difference between observation and model fields (either background or analysis). |
| **DMSP** | Defense Meteorological Satellite Program (USA). |
| **Dr.Hook** | A portable instrumentation tool developed at ECMWF, which enables reliable traceback upon failure and produces per subroutine, per OpenMP-thread profiling information about resource usages, like wall-clock time, CPU-time and on some computer platforms even MFlops- and MIPS-rates. |
| **DRIBU** | WMO code for drifting buyos. |
### ECMA
ODB-database name for all used and monitored observations (Extended Central Memory Array).

### ECMASCR
Normally the same as ECMA-database, but used as a temporary database.

### ECMWF
European Centre for Medium-Range Weather Forecasts.

### ENVISAT
Environmental Satellite (ESA).

### ERA-15
ECMWF Re-Analysis for 15 years.

### ERA-40
ECMWF Re-Analysis for 40 years.

### ESA
European Space Agency.

### EUMETSAT
European Organisation for the Exploitation of Meteorological Satellites.

### First guess
Starting point for an iterative process (such as the minimization of the variational cost function). First guess can be different from the Background.

### FDB
Fields DataBase.

### Fragmentation
Operating systems allocate disc space in chunks as they create and expand files. When files are purged, these chunks are released for re-use. Over time the disc space may end up "fragmented" into many small pieces, which can slow the performance and the reliability of the system. The same analogy holds for performance degradation of a program, which constantly allocates and de-allocates memory during the execution: free space becomes fragmented.

### Gigabyte
A gigabyte (GB) is a measure of memory space equal to 1,024 megabytes or 1,073,741,824 (i.e. $2^{30}$) bytes.

### GOES
U.S. programme of geostationary satellites.

### GPS
Global Positioning System.

### GRIB
Gridded data in Binary Form (a WMO code).

### HIRS
High Resolution Infra-Red Sounder.
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPM</td>
<td>High Performance Monitor – a process or computer hardware that enables to measure MFlops-rates, thus to monitor application performance.</td>
</tr>
<tr>
<td>IASI</td>
<td>Infrared Atmospheric Sounding Interferometer.</td>
</tr>
<tr>
<td>IFS</td>
<td>Integrated Forecasting System.</td>
</tr>
<tr>
<td>Increment</td>
<td>The initial-condition correction (w.r.t. the Background) computed by assimilation system.</td>
</tr>
<tr>
<td>Incremental</td>
<td>A technique in which the control variable is the increment rather than the full initial state – the increment may be at reduced resolution to save computational cost.</td>
</tr>
<tr>
<td>I/O</td>
<td>Input and/or output operation.</td>
</tr>
<tr>
<td>IR</td>
<td>Infrared.</td>
</tr>
<tr>
<td>Jacobian</td>
<td>Matrix of first derivatives.</td>
</tr>
<tr>
<td>Jb</td>
<td>The background term in variational assimilation.</td>
</tr>
<tr>
<td>Jc</td>
<td>Constraints added as additional terms to the variational cost function.</td>
</tr>
<tr>
<td>Jo</td>
<td>Rge observation term in variational assimilation.</td>
</tr>
<tr>
<td>Kilobyte</td>
<td>A kilobyte (KB) is a memory space measure equal 1,024, or $2^{10}$ bytes.</td>
</tr>
<tr>
<td>L31, L60, L90, ...</td>
<td>31, 60, 90, ... levels in ECMWF’s forecasting models (its vertical discretization).</td>
</tr>
<tr>
<td>MAKECMA</td>
<td>An observation pre- and postprocessing software (now obsolescent at ECMWF) to create files in CMA-format holding 4D-Var-analysis observation data. Also known as OBSPROC or COH.</td>
</tr>
<tr>
<td>MAP</td>
<td>Mesoscale Alpine Programme.</td>
</tr>
<tr>
<td>MARS</td>
<td>Meteorological Archival and Retrieval System.</td>
</tr>
</tbody>
</table>
MATCHUP A software to update ECMA ODB-database with 4D-Var-information from CCMA ODB-database.

Megabyte A megabyte (MB) is a memory space measure equal to $1,048,576$ ($1024 \times 1024$), or $2^{20}$ bytes. $1,024$ megabytes is a gigabyte.

MERGEODB A software to merge multiple ECMA ODB-databases (which were split per satellite sensor in BUFR2ODB) into one large virtual database to give impression for 4D-Var about single database view only. This process is normally embedded into IFS-code and is run just before any observation processing begins in a 4D-Var-run.

METVIEW Generalized data access, manipulation and visualization system for meteorological data.

MF, M-F Meteo-France.

MFlops Megaflops per second, millions of floating point operations per second. A performance measure to show how efficient a particular computer is on numerically intensive applications.

Minimization An iterative algorithm to find the minimum of a function (e.g. M1QN3 and Conjugate Gradient).

MIPAS Michelson Interferometer for Passive Atmospheric Sounding.

MIPS Millions of instructions per second. Another measure about computer performance in particular in scalar intensive computations.

MKCMARPL An abbreviation for MAKECMA replacement. A large set of subroutines in IFS-code to preprocess observations, before they actually enter into analysis. Used to be a separate software package called MAKECMA (also known as OBSPROC or COH) before BUFR2ODB software was available.

Model variables The predictive variables of the forecast model (e.g. $lnP_s$, $T_v$, $VO$, $Div$, $O_3$, $C_{lw}$, $Ci$, $Ca$ and the surface variables).

MODIS Moderate Resolution Imaging Spectroradiometer.

MPI Message Passing Interface, a system dependent software library, which can be used in parallel meteorological application programs. Also denotes Max Planck Institute.

MPI I/O Message Passing Interface, I/O-extension.
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPL</td>
<td>Message Passign Library is a software layer on top of MPI developed at ECMWF.</td>
</tr>
<tr>
<td>MSG</td>
<td>METEOSAT Second Generation.</td>
</tr>
<tr>
<td>MSLP</td>
<td>Mean Sea Level Pressure.</td>
</tr>
<tr>
<td>MSU</td>
<td>Microwave Sounding Unit.</td>
</tr>
<tr>
<td>NCAR</td>
<td>U.S. National Centre for Atmospheric Research.</td>
</tr>
<tr>
<td>NESDIS</td>
<td>National Environmental Satellite, Data and Information Service, USA.</td>
</tr>
<tr>
<td>NOAA</td>
<td>National Oceanic and Atmospheric Administration, USA.</td>
</tr>
</tbody>
</table>

**Observation operator**

The calculations required to go from model variables to the equivalents of the observed quantities. Includes horizontal and vertical interpolation to the observation locations. In 4D-Var the forecast model can usefully be seen as part of the observation operator.

**Observed variables**

Observed quantities such as $T$, $u$, $v$, $u_{10m}$, $v_{10m}$, $r_{h_{2m}}$, $P_s$, IR and microwave radiances, TCWV etc.

**OBSPROC**

See MAKECMA.

**ODB**

Observational DataBase is a database software developed at ECMWF to manage large amount of (especially) satellite observations through the 4D-Var system in efficient manner. Its core parts consist of ODB/SQL data query engine and Fortran90 interface layer.

**ODBSHUFFLE**

A software, which translates active observational data from ECMA into CCMA ODB-database for use by 4D-Var minimization process.

**ODB2ECMA**

A conversion software package to translate ECMA ODB-database back to so called feedback BUFR-format after 4D-Var completion.

**ODB/SQL**

A very-subset of SQL-language and is applicable to ODB software only to perform data queries from its databases.

**OI**

Optimum Interpolation.

**OML**

OpenMP support library developed at ECMWF to allow OpenMP-multithreaded computation in high-performance meteorological applications.
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OpenMP</strong></td>
<td>A portable application program interface aimed at parallel programs on shared-memory machines.</td>
</tr>
<tr>
<td><strong>OSE</strong></td>
<td>Observing System Experiment.</td>
</tr>
<tr>
<td><strong>PAOB</strong></td>
<td>WMO code for surface pressure pseudo-observations (from Australia).</td>
</tr>
<tr>
<td><strong>Passive data</strong></td>
<td>Data that are processed for monitoring purposes, but not actively used in the assimilation – observation minus background departures are computed.</td>
</tr>
<tr>
<td><strong>PILOT</strong></td>
<td>Upper-wind report from a fixed land station.</td>
</tr>
<tr>
<td><strong>Preconditioning</strong></td>
<td>A technique that makes it faster/easier/less expensive to minimize the variational cost-function by incorporating some knowledge about its shape (its second derivative).</td>
</tr>
<tr>
<td><strong>PREODB</strong></td>
<td>ODB-database for ERA-40 created from input (conventional) BUFR-data from various data sources in order to perform easier duplicate checking with ODB software before input data actually gets selected for re-analysis. Contains over 40 years and more than 200GB of ODB-data online, split into 6 hourly time windows. ERA-40 assimilation has used PREODB from September 1957 till December 2001. Also the very first ODB-application/implementation.</td>
</tr>
<tr>
<td><strong>PREPIFS</strong></td>
<td>Software to PREPare IFS research experiments and eventually submit them to run on a supercomputer.</td>
</tr>
<tr>
<td><strong>QC</strong></td>
<td>Quality Control.</td>
</tr>
<tr>
<td><strong>QuikScat</strong></td>
<td>NASA's dedicated mission to fly the SeaWinds scatterometer.</td>
</tr>
<tr>
<td><strong>RDB</strong></td>
<td>Reports Data Base.</td>
</tr>
<tr>
<td><strong>REO3</strong></td>
<td>Retrieved ozone layers.</td>
</tr>
<tr>
<td><strong>Rejected data</strong></td>
<td>The data that will not be used actively in the assimilation.</td>
</tr>
<tr>
<td><strong>RISC</strong></td>
<td>Reduced Instruction Set Computer.</td>
</tr>
<tr>
<td><strong>RTOVS</strong></td>
<td>Replacement TOVS.</td>
</tr>
<tr>
<td><strong>RT-codes</strong></td>
<td>Radiative Transfer code (e.g. RTTOV, OPTRAN, RTI-ASI, SYNSATRAD, ...), providing the observation operator for infrared and microwave radiances.</td>
</tr>
<tr>
<td><strong>RTTOV</strong></td>
<td>RT model for TOVS, ATOVS and several other atmospheric sounders.</td>
</tr>
<tr>
<td><strong>SATOB</strong></td>
<td>WMO code for Satellite Observation report (Cloud track winds).</td>
</tr>
<tr>
<td><strong>SBUV</strong></td>
<td>Solar Backscatter Ultraviolet Radiometer.</td>
</tr>
<tr>
<td><strong>SCIAMACHY</strong></td>
<td>Scanning Imaging Absorption Spectrometer for Atmospheric Cartography.</td>
</tr>
<tr>
<td><strong>Schema</strong></td>
<td>.</td>
</tr>
<tr>
<td><strong>Screening</strong></td>
<td>A set of quality control and data selection procedures prior to the analysis (uses the Background).</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>Structured Query Language is international standard for making data queries from (especially) relational databases. See also ODB/SQL.</td>
</tr>
<tr>
<td><strong>$SRC</strong></td>
<td>Root directory of the source code in this document. At ECMWF this refers to directory /ccvobs, when source files are under ClearCase software version handling system. Otherwise it can be any file system holding a sub-tree of directories and sourcefiles.</td>
</tr>
<tr>
<td><strong>SSM/I, SSMI</strong></td>
<td>Special Sensor Microwave Imager.</td>
</tr>
<tr>
<td><strong>SSMIS</strong></td>
<td>Special Sensor Microwave Imager Sounder.</td>
</tr>
<tr>
<td><strong>SST</strong></td>
<td>Sea Surface Temperature.</td>
</tr>
<tr>
<td><strong>SSU</strong></td>
<td>Stratospheric Sounding Unit.</td>
</tr>
<tr>
<td><strong>SYNOP</strong></td>
<td>WMO code for upper air observations.</td>
</tr>
<tr>
<td><strong>TCWV</strong></td>
<td>Total Column Water Vapour.</td>
</tr>
<tr>
<td><strong>TEMP</strong></td>
<td>WMO code for surface weather observations.</td>
</tr>
<tr>
<td><strong>Terabyte</strong></td>
<td>A terabyte (TB) is a measure of memory space equal to 1,024 gigabytes or 1,099,511,627,776 (i.e. 2⁴⁰) bytes.</td>
</tr>
<tr>
<td><strong>Thinning</strong></td>
<td>A procedure to reduce the density of data to be comparable with the model resolution.</td>
</tr>
<tr>
<td><strong>TL</strong></td>
<td>Tangent Linear – the linear model that best approximates the corresponding non-linear model for a given atmospheric state.</td>
</tr>
<tr>
<td><strong>TOVS</strong></td>
<td>TIROS Operational Vertical Sounder.</td>
</tr>
<tr>
<td><strong>Trajectory</strong></td>
<td>A time sequence of atmospheric states (produced by the non-linear forecast model), around which perturbations are propagated by the TL model.</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>TRMM</strong></td>
<td>Tropical Rainfall Measuring Mission.</td>
</tr>
<tr>
<td><strong>UTC</strong></td>
<td>Universal Time Coordinated.</td>
</tr>
<tr>
<td><strong>UV</strong></td>
<td>Ultraviolet.</td>
</tr>
<tr>
<td><strong>VarQC</strong></td>
<td>Variational Quality Control.</td>
</tr>
<tr>
<td><strong>WMO</strong></td>
<td>World Meteorological Organization based in Geneva, Switzerland.</td>
</tr>
</tbody>
</table>
REFERENCES

References


# List of Figures

1. Observational data in a single table (a matrix or so called flat file) .................. 14
2. Table relations and data normalization in ODB .............................. 15
3. Software layers in ODB ........................................ 21
4. ODB/SQL compilation system ........................................ 24
5. Using `odbccomp -command` to compile data layouts and SQL-queries ... 25
6. Using `odbcopsc -command` to re-create ODB data access library ......... 26
7. Vertical data partitioning ...................................... 27
8. One-to-one relationship between two tables, where data rows are not in the same order 28
9. One-to-one relationship between tables, where not all tables have matching row present in the "sibling" table(s) 29
10. Horizontal data partitioning .................................. 29
11. Database files ............................................... 32
12. Components in CREATE TABLE statement .......................... 40
13. Using the `odbf90 -command` to build a sequential executable .......... 47
14. Using the `odbf90mp -command` to build a MPI-parallel executable ... 48
15. The program flow in the 4D-Var-system at ECMWF .................. 113
16. Observational data flow at ECMWF ................................ 114
17. Software layers in ODB and IFS interface ................................ 114
18. Interfacing and transferring data between IFS and ODB ................ 117
19. Generic database table hierarchy in IFS ................................ 138
20. Satellite database table hierarchy in IFS ................................ 138
21. Use `newodb -command` to create your own database library .......... 176
22. Duplicate checking with PREODB, part#1 ................................ 179
23. Duplicate checking with PREODB, part#2 ................................ 180
24. BUFR to ODB/ECMA translation procedure .......................... 181
25. ODB/ECMA to feedback-BUFR translation procedure .................. 183
26. An example of ODBVIEWER-report output .......................... 186
27. An example of ODBVIEWER-report coverage plot .................. 186
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ODB/SQL operators</td>
<td>36</td>
</tr>
<tr>
<td>2</td>
<td>ODB/SQL keywords</td>
<td>38</td>
</tr>
<tr>
<td>3</td>
<td>ODB/SQL language element syntax</td>
<td>39</td>
</tr>
<tr>
<td>4</td>
<td>ODB/SQL built-in functions</td>
<td>45</td>
</tr>
<tr>
<td>5</td>
<td>Most important ODB Fortran90 functions.</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>Observational arrays in the ODB interface to IFS</td>
<td>115</td>
</tr>
<tr>
<td>7</td>
<td>Datatype obschar_t</td>
<td>139</td>
</tr>
<tr>
<td>8</td>
<td>Datatype datum_flag_t</td>
<td>140</td>
</tr>
<tr>
<td>9</td>
<td>Datatype status_t</td>
<td>140</td>
</tr>
<tr>
<td>10</td>
<td>Datatype report_rdbflag_t</td>
<td>141</td>
</tr>
<tr>
<td>11</td>
<td>Datatype datum_rdbflag_t</td>
<td>142</td>
</tr>
<tr>
<td>12</td>
<td>Datatype level_t</td>
<td>143</td>
</tr>
<tr>
<td>13</td>
<td>Datatype report_event1_t</td>
<td>143</td>
</tr>
<tr>
<td>14</td>
<td>Datatype report_blacklist_t</td>
<td>144</td>
</tr>
<tr>
<td>15</td>
<td>Datatype datum_blacklist_t</td>
<td>145</td>
</tr>
<tr>
<td>16</td>
<td>Datatype varno_presence_t</td>
<td>146</td>
</tr>
<tr>
<td>17</td>
<td>Datatype datum_event1_t</td>
<td>146</td>
</tr>
<tr>
<td>18</td>
<td>Database table desc</td>
<td>148</td>
</tr>
<tr>
<td>19</td>
<td>Database table ddrs</td>
<td>148</td>
</tr>
<tr>
<td>20</td>
<td>Database table poolmask</td>
<td>149</td>
</tr>
<tr>
<td>21</td>
<td>Database table timeslot_index</td>
<td>150</td>
</tr>
<tr>
<td>22</td>
<td>Database table index</td>
<td>150</td>
</tr>
<tr>
<td>23</td>
<td>Database table hdr</td>
<td>151</td>
</tr>
<tr>
<td>24</td>
<td>Database table body</td>
<td>153</td>
</tr>
<tr>
<td>25</td>
<td>Database table errstat</td>
<td>154</td>
</tr>
<tr>
<td>26</td>
<td>Database table update</td>
<td>154</td>
</tr>
<tr>
<td>27</td>
<td>Database table sat</td>
<td>155</td>
</tr>
<tr>
<td>28</td>
<td>Database table reo3</td>
<td>155</td>
</tr>
<tr>
<td>29</td>
<td>Database table satob</td>
<td>156</td>
</tr>
<tr>
<td>30</td>
<td>Database table scatt</td>
<td>157</td>
</tr>
<tr>
<td>31</td>
<td>Database table scatt_body</td>
<td>157</td>
</tr>
<tr>
<td>32</td>
<td>Database table ssmi</td>
<td>158</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>33</td>
<td>Database table <strong>ssmi_body</strong></td>
<td>158</td>
</tr>
<tr>
<td>34</td>
<td>Database table <strong>atovs</strong></td>
<td>159</td>
</tr>
<tr>
<td>35</td>
<td>Database table <strong>atovs_pred</strong></td>
<td>160</td>
</tr>
<tr>
<td>36</td>
<td>Database table <strong>bufr</strong></td>
<td>160</td>
</tr>
<tr>
<td>37</td>
<td>Database table <strong>sat</strong></td>
<td>162</td>
</tr>
<tr>
<td>38</td>
<td>Database table <strong>atovs</strong></td>
<td>162</td>
</tr>
<tr>
<td>39</td>
<td>A checklist how to add a new column entry</td>
<td>163</td>
</tr>
<tr>
<td>41</td>
<td>BUFR-types and subtypes</td>
<td>197</td>
</tr>
<tr>
<td>42</td>
<td>CMA observation types and codetypes</td>
<td>199</td>
</tr>
<tr>
<td>43</td>
<td>CMA obs.types mapped into BUFR types</td>
<td>200</td>
</tr>
<tr>
<td>44</td>
<td>BUFR obs.types mapped into CMA types</td>
<td>202</td>
</tr>
<tr>
<td>45</td>
<td>SATOB computational methods</td>
<td>203</td>
</tr>
<tr>
<td>46</td>
<td>SATOB computational methods</td>
<td>204</td>
</tr>
<tr>
<td>47</td>
<td>Geostationary satellite names</td>
<td>204</td>
</tr>
<tr>
<td>48</td>
<td>Satellite series</td>
<td>204</td>
</tr>
<tr>
<td>49</td>
<td>Observation variable numbers</td>
<td>205</td>
</tr>
<tr>
<td>50</td>
<td>ODB source codes and dependencies per library</td>
<td>240</td>
</tr>
<tr>
<td>51</td>
<td>Source codes &amp; dependencies per permanent (recognized) database libraries</td>
<td>240</td>
</tr>
<tr>
<td>52</td>
<td>Dr.Hook environment variables</td>
<td>251</td>
</tr>
<tr>
<td>53</td>
<td>More Dr.Hook environment variables</td>
<td>252</td>
</tr>
</tbody>
</table>
Index

., 71
*., 36, 59, 71, 92
**, ^, 37
+, 36
+$mdi, 30
-, 36
-$mdi, 30
,, 185, 194
ddl, 36
.sql, 24, 36
/, 37
/ccvobs, 170, 178, 260, 266
/ccvobs/odb/ddl, 178
/ccvobs/odb/ddl,DB, 178
/local/disk/DB, 111
/vol/odb/ODBDIR/use_odb, 81
/vol/odb/ODBDIR/use_odb.sh, 81
<, 37
<= ... <, 37
<> , 37
=, /=, 37
< ... <, 37
<, 37
<= ... <=, 37
>= ... >, 37
>, 37
[0-9], 176
[A-Z], 176
#, 44
$, 73
$-variable, 50, 62, 74, 75, 77, 97, 101
$ARCH, 241
$BASETIME, 111
$MXUP_TRAJ, 184
$NMXUPD, 160
$ODB_SYSPATH/odb98.flags, 26
$SRC, 170, 178, 266
$SRC/ifs/common, 164
$SRC/ifs/common/itdef.h, 114
$SRC/ifs/common/openmp_obs.h, 114
$SRC/ifs/common/yomdbdefs.h, 114, 165–167
$SRC/ifs/common/yomdb_vars.h, 114, 165–167
$SRC/ifs/module, 165
$SRC/ifs/module/parcma.F90, 156
$SRC/ifs/module/yomdb.F90, 113, 118, 164
$SRC/ifs/module/yomtvradF90, 203
$SRC/ifs/obs_preproc, 165
$SRC/ifs/pp_obs, 165
$SRC/ifs/setup/suvmnb.F90, 205
$SRC/ifs/var, 165
$SRC/ifs/var/ecset.F90, 126
$SRC/ifs/var/writeoba.F90, 189
$SRC/ifsaux/module/ecsort.F90, 247
$SRC/ifsaux/utilities/rsort32.c, 247
$SRC/odb, 240
$SRC/odb/aux, 165
$SRC/odb/bufr2odb, 165
$SRC/odb/cma2odb, 164
$SRC/odb/cma2odb/closedb.F90, 125
$SRC/odb/cma2odb/ctxgetdb.F90, 125
$SRC/odb/cma2odb/ctxinitdb.F90, 119,
$SRC/odb/cma2odb/ctxputdb.F90, 117
$SRC/odb/cma2odb/load.F90, 117
$SRC/odb/cma2odb/loadddb.F90, 113
$SRC/odb/cma2odb/mapvardb.F90, 130
$SRC/odb/cma2odb/matchupdb.F90, 132
$SRC/odb/cma2odb/openpdb.F90, 118
$SRC/odb/cma2odb/putatdb.F90, 129, 165, 174
$SRC/odb/cma2odb/putdb.F90, 124, 165
$SRC/odb/cma2odb/setactivedb.F90, 118
$SRC/odb/cma2odb/shuffledb.F90, 117, 132,
INDEX

133
$SRC/odb/cma2odb/store.F90, 117
$SRC/odb/cma2odb/storedb.F90, 131
$SRC/odb/cma2odb/swapout.F90, 117
$SRC/odb/cma2odb/swapoutdb.F90, 130
$SRC/odb/cma2odb/unmapdb.F90, 130
$SRC/odb/cma2odb/xchangedatadb.F90, 132, 133, 165
$SRC/odb/compiler, 165
$SRC/odb/dll, 164–169
$SRC/odb/dll/ECMA, 168
$SRC/odb/dll/ECMA/ECMA.ddl, 147
$SRC/odb/dll/PREODBB, 166
$SRC/odb/dll/dbname, 164, 165
$SRC/odb/dll/, 120
$SRC/odb/dll/cma.h, 119, 139, 165–167, 173
$SRC/odb/dll/varno.h, 205
$SRC/odb/include, 49
$SRC/odb/lib, 165
$SRC/odb/lib/msgpass_obsdata.F90, 117, 190
$SRC/odb/lib/rsort32_odb.c, 247
$SRC/odb/module, 49, 165
$SRC/odb/module/context.F90, 117
$SRC/odb/module/odb.F90, 63
$SRC/odb/tools, 164, 165
$VPATH, 240
$WDIR, 111, 195
$WDIR/ECMA.airs, 111
$all, 121
$codetype, 123
$dbname, 82, 194
$dbname.dll, 82
$hdr_max, 121
$hdr_min, 121
$skset, 122
$mdi, 30
$mx_satob_weight, 156
$nxupdp, 118
$obstype, 122
$sensor, 123
$slot, 122
$s, 43
%., 37
%d, 195
& & , AND, 37
@, 36
_odb_glue.c, 25
_odb_glue.o, 176
0.5GB, 241
1D-Var, 259
3D-FGAT, 175, 259
3D-Var, 259
4D-Var, 4–6, 17, 18, 22, 25, 48, 81, 103, 113, 120, 122, 126, 132, 134, 137, 150, 154, 161, 189, 259, 260, 262–264
A, 52, 59, 69, 122, 130
ABS, 38
ABS(), 42
abs(x), 45
ACARS, 259
acos, 45
Active data, 259
AD, 259
addfile, 170
aggregate functions, 94
AIREP, 259
AIRS, 111, 203, 259
ALIGN, 38
AM DAR, 259
AMSU (-A, -B), 259
AMSU-A, 259
AMSU-B, 259
AMSUA, 203
AMSUB, 203
AMV, 259
Analysis variables, 259
AND, 37, 95
appenddb.F90, 108
appenddb1.out, 109
appenddb2.out, 110
AQUA, 259
ARCH, 242
ARPEGE, 259
AS, 38
as follows, 97, 104
ASC, 38, 42
INDEX

ASCII, 259
asin, 45
AS , = 36
atan, 45
atan2, 45
ATOVS, 259
atovs, 159, 161, 162
atovs_pred, 160, 161
Background, 259
Backus Naur Form, 35
BATHY, 259
BETWEEN, 38
BETWEEN ... AND ..., 37
Bias, 259
bias, 103, 107, 110
big-endian, 193
Bit, 260
Bitfield, 22, 38, 39
Blacklist, 260
BNF, 35
body, 83, 95, 96, 103, 145, 152, 161–163, 175, 190
BUFR, 6, 175, 181, 260, 265
bufr, 160, 175, 190
BUFR-world, 200–203
BUFR2ODB, 4, 165, 197, 198, 239, 260
BUFR2ODB-tools, 6
hdr, 197, 198, 200–203
BY, 42
Byte, 260
cartesian product, 95
CCMA, 3, 118–120, 127, 132–134, 163, 166, 173, 175, 189, 239, 260
CDNAME, 120, 129, 130
ceil, 45
check upd.sql, 107
CHGEOSATN, 204
child, 14, 83
CHMETHOD, 204
CHSENSOR, 203
CHSERIES, 204
circle, 46
ClearCase, 260, 266
CLOSEDB, 125
closedb.F90, 125
CMA, 5, 6, 122, 123, 126
CMA I/O-routines, 192
CMA-world, 200–203
cma.h, 119, 139, 165–167, 173
hdr, 199–203
COH, 260
colget, 69
colpack, 69
colput(:), 104
COM_CHAN, 204
Compilation and linking, 91
complex.inp, 97
complex.sql, 94
configure, 241
context.F90, 117
Control variable, 260
cos, 45
cosh, 45
Cost function, 260
CPP, 44
CPU, 260
CREATE, 53
CREATE TYPE, 39
CREATE VIEW, 52, 54, 57, 59, 66, 69, 76, 78, 91, 92, 102, 103
createdb.F90, 84
createdb.out, 85
createdb.usage, 85
CREATE TABLE, 14, 15, 36, 38
CREATE TYPE, 36, 38
CREATE VIEW, 36, 38, 41
csh, 241
CTXGETDB, 121, 125, 127, 130
ctxgetdb.F90, 125
CTXID, 126
CTXINITDB, 119, 124–126, 172
txinitdb.F90, 120, 165, 170, 173, 174
CTXPUTDB, 127
txputdb.F90, 127
cx.out, 96
cx1, 94
cx2, 94
cx3, 95, 97
cx4, 95
cx5, 96, 97
d, 56–58, 68, 69
Data Definition Layout, 22
data layout, 83
DB, 23, 25, 26, 33, 82, 84, 111, 178
DB.ddl, 22, 26, 82, 84, 85, 178
DB.flags, 26
DB.inp, 83
DB.insertinto, 86
DB.sch, 26
dble, 45
dca, 31, 32
dd2ddl.x, 178
DDL, 260
dl, 166
DDL underscore-file, 23, 26
DDL-file, 22
ddl-file, 62, 75, 77
ddl_, 23
ddr, 115, 148, 161
deg2rad, 46
degrees, 46
depar, 104, 107
Departure, 260
DESC, 38, 42
desc, 148, 161, 175
direct column access, 32
dist(lat1, lon1, lat2, lon2), 46
distance, 46
DMSP, 204, 260
Dr.Hook, 260
DRIBU, 260
dtname, 56, 57, 59–61, 71, 78
ECMA, 3, 118–120, 132–135, 163, 164, 166,
175, 177, 178, 189, 197, 198, 239,
261
ECMA.ddl, 147, 178
ECMA.sch, 178

ECMASCR, 118–120, 134, 166, 175, 197, 198,
239, 261
ECMWF, 63, 247, 261
ECSET, 122, 126
ecset.F90, 126
Endian, 242
everyno, 83
ENVISAT, 261
ERA-15, 261
ERA-40, 261
ERA40, 175
ERA40 reanalysis, 6
erstat, 154, 161
ESA, 261
EUMETSAT, 261
exp, 45
FALSE, 58, 67, 77, 247, 248
FAQ, 4
fclose, 192
FDB, 261
feedback-BUFR, 6
feedrow.F90, 107
file, 67
First guess, 261
float(x), 45
floor(x), 45
fodb.h, 50
following commands, 85, 88, 102
following output, 96, 100
fopen, 192
FOREIGN KEY, 15
Fragmentation, 261
fread, 192
FROM, 23, 36, 38, 53, 94, 174, 190
fwrite, 192
GB, 261
GETATDB, 127–129
getatdb.F90, 128, 165
GETDB, 118, 120, 121, 124, 125, 128, 130
getdb.F90, 121, 165, 173
ghost, 57
Gigabyte, 261
INDEX

GMS, 204
GOES, 204, 261
GOESIMG, 203
GPS, 261
GRIB, 261
gunzip, 31, 241
gzip, 31, 33
handle, 51, 52, 54, 55, 57, 59, 61, 62, 66, 69–72, 75–78
HC32, 191, 192
hcat, 192
hdr, 83, 94–96, 145, 151, 161, 175, 190
HHMMSS, 148, 151, 165
HIPHOP, 172
HIRS, 203, 261
HPM, 262
http://www.ecmwf.int/products/data/software/bufr_–reference_manual.pdf, 198
I, 57, 68, 130–132
I/O, 262
I/O-method, 70
I/O-method#1, 22
I/O-method#4, 22
IASI, 262
ibits, 45
hdr, 137
idx, 78
IEEE 754, 191, 193
IFS, 117, 121, 122, 125, 126, 247, 262
IFS/Screening, 189
IN, 37
in the following, 103
IN, NOT IN, 38
Increment, 262
Incremental, 262
index, 150, 161, 247
INDSAT, 204
INITMDB, 119, 129
initmdb.F90, 119, 165–167
int, 45
INTEGER, 75
INTEGER(4), 50, 56, 68
IOASSIGN, 31, 85
ioassign, 32
IOASSIGN-file, 33
IR, 204, 262, 264
IS NOT NULL, 37, 38
IS NULL, 37, 38
IT, 130
it, 114
itdef.h, 114
Jacobian, 262
Jb, 262
Jc, 262
Jo, 262
JPMAXSIMDB, 117
JPMX_SATOB_WEIGHT, 156
KB, 262
KCODETYPE, 123
key, 83
KEYSORT, 4, 247
Kilobyte, 262
KINFO, 118, 121, 124
KINFO(1), 118, 121
KINFO(2), 118, 121
KINFO(3), 121
KINFOLEN, 118, 121
km, 46
KOBSTYPE, 122
KSENSOR, 123
KSET, 122
KTSLOT, 122
KVERSION, 120, 121, 124
L31, L60, L90, ..., 262
latest, 82
layout, 82
ldexp(x,n), 45
lg, 46
lib/rsort32_odb.c, 247
libCCMA.a, 166, 239, 240
libDB.a, 23, 25, 26, 84
libECMA.a, 166, 239, 240
<table>
<thead>
<tr>
<th>Element</th>
<th>Page 1</th>
<th>Page 2</th>
<th>Page 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>libECMASCR.a</td>
<td>166</td>
<td>239</td>
<td>240</td>
</tr>
<tr>
<td>libemos.a</td>
<td>239</td>
<td></td>
<td></td>
</tr>
<tr>
<td>libifs.a</td>
<td>166</td>
<td>239</td>
<td>241</td>
</tr>
<tr>
<td>libifsaux.a</td>
<td>239</td>
<td>241</td>
<td></td>
</tr>
<tr>
<td>libmpi_serial.a</td>
<td>21</td>
<td>239</td>
<td></td>
</tr>
<tr>
<td>libMYDB.a</td>
<td>177</td>
<td></td>
<td></td>
</tr>
<tr>
<td>libobsort.a</td>
<td>239</td>
<td>241</td>
<td></td>
</tr>
<tr>
<td>libobsproc.a</td>
<td>239</td>
<td>241</td>
<td></td>
</tr>
<tr>
<td>libobstat.a</td>
<td>239</td>
<td></td>
<td></td>
</tr>
<tr>
<td>libodb.a</td>
<td>239</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>libodbdummy.a</td>
<td>239</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>libodbdmain.a</td>
<td>239</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>libodbport.a</td>
<td>166</td>
<td>239</td>
<td>240</td>
</tr>
<tr>
<td>libodbsqlcompiler.a</td>
<td>239</td>
<td>241</td>
<td></td>
</tr>
<tr>
<td>libPREODB.a</td>
<td>239</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>LIKE, RLIKE</td>
<td>96</td>
<td>92</td>
<td></td>
</tr>
<tr>
<td>LIKE, LINKLEN, LINKOFFSET</td>
<td>78</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ln</td>
<td>46</td>
<td></td>
<td></td>
</tr>
<tr>
<td>load.F90</td>
<td>117</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOADDB</td>
<td>131</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>loaddb.F90</td>
<td>132</td>
<td></td>
<td></td>
</tr>
<tr>
<td>log(x)</td>
<td>46</td>
<td></td>
<td></td>
</tr>
<tr>
<td>log10(x)</td>
<td>46</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M1QN3</td>
<td>263</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAGICS</td>
<td>262</td>
<td></td>
<td></td>
</tr>
<tr>
<td>make</td>
<td>241</td>
<td></td>
<td></td>
</tr>
<tr>
<td>make.ibm_power4</td>
<td>242</td>
<td></td>
<td></td>
</tr>
<tr>
<td>make.linux</td>
<td>242</td>
<td></td>
<td></td>
</tr>
<tr>
<td>make.rs6000</td>
<td>242</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAKECMA</td>
<td>262</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAKECMA-replacement</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td>262</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAPVARDB</td>
<td>129</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td>mapvardb.F90</td>
<td>130</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARS</td>
<td>262</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATCHUP</td>
<td>263</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATCHUPDB</td>
<td>134</td>
<td></td>
<td></td>
</tr>
<tr>
<td>matchupdb.F90</td>
<td>132</td>
<td></td>
<td></td>
</tr>
<tr>
<td>max</td>
<td>45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpoolno, maxpoolno</td>
<td>51</td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxrows</td>
<td>45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MB</td>
<td>263</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB</td>
<td>119, 129, 130</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB/MLNK-pointers</td>
<td>115, 147</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB1DFIA</td>
<td>160</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB1DINA</td>
<td>160</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB1BCP1..16A</td>
<td>160</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB1BCPTS1(:)</td>
<td>160</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB1DCU1A</td>
<td>160</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB1DCUA</td>
<td>160</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB1DERA</td>
<td>160</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB1DFIN</td>
<td>158</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB1DITA</td>
<td>160</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB1DNIT</td>
<td>158</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB1DPFA</td>
<td>160</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB1DVAQ</td>
<td>159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB1DVAT</td>
<td>159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB1DVBOQ</td>
<td>159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB1DVBT</td>
<td>159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB1DVPS</td>
<td>159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_ABNOB_AT_INDEX</td>
<td>150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_ANDDATE_AT_DESC</td>
<td>148</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_ANTIME_AT_DESC</td>
<td>148</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_AREATYPE_AT_HDR</td>
<td>151</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_AUX_AT_BODY</td>
<td>153</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_BODY_COUNT_AT_POOLMASK</td>
<td>149</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_BUFRTYPE_AT_HDR</td>
<td>152</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_BUFRTYPE_AT_POOLMASK</td>
<td>149</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_BULKDATA_AT_DDRS</td>
<td>149</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_CC_AT_SSMI_BODY(1:2)</td>
<td>159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_CHAN_FREQ_AT_SATOB</td>
<td>156</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_CHECKSUM_AT_HDR</td>
<td>152</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_CIW_AT_SSMI_BODY(1:2)</td>
<td>159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_CLD_FG_DEPAR_AT_BODY</td>
<td>159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_CLDNE1..3_AT_ATOVS</td>
<td>159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_CLDPSTOP1..3_AT_ATOVS</td>
<td>159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_CLW_AT_SSMI_BODY(1:2)</td>
<td>159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_CLDTYPE_AT_HDR</td>
<td>151</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_CLDTYPE_AT_POOLMASK</td>
<td>149</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_CREABY_AT_DESC</td>
<td>148</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_CREADATE_AT_DESC</td>
<td>148</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB_CREATION_TIME_AT_DESC</td>
<td>148</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
INDEX

MDB_CSR_PCLEAR_AT_BODY, 153
MDB_CSR_PCL CLOUDY_AT_BODY, 153
MDB_DDRNO_AT_DDRS, 148
MDB_DT_BY_DP_AT_SATOB, 156
MDB_HDR_COUNT_AT_POOLMASK, 149
MDB_IDENT_AT_HDR, 151
MDB_INSTTYPE_AT_HDR, 151
MDB_KSET_AT_INDEX, 150
MDB_LANDSEA_OBS_AT_ATOVS, 160
MDB_LATLON_RAD_AT_DESC, 148
MDB_MAPOMM_AT_INDEX, 150
MDB_MAX_BODYLEN_AT_POOLMASK, 149
MDB_MODBY_AT_DESC, 148
MDB_MODDATE_AT_DESC, 148
MDB_MODTIME_AT_DESC, 148
MDB_MPC_AT_SCATT_BODY, 157
MDB_MSG_AT_BUFR, 160
MDB_MXUP_TRAJ_AT_DESC, 148
MDB_NRETR_AMB_AT_SCATT_BODY, 158
MDB_NUMACTIVEB_AT_HDR, 151
MDB_NUMLEV_AT_HDR, 151
MDB_OBS_TYPE_AT_POOLMASK, 149
MDB_P_BEST_AT_SATOB, 157
MDB_P_OLD_AT_SATOB, 157
MDB_POOLNO_AT_POOLMASK, 149
MDB_PREC_CV_AT_SSMI(1:2), 158
MDB_PREC_ST_AT_SSMI(1:2), 158
MDB_PROCID_AT_INDEX, 150
MDB_RAD_AN_DEPAR_AT_SSMI_BODY, 159
MDB_RAD_BIAS_AT_SSMI_BODY, 159
MDB_RAD_COST_AT_SSMI_BODY, 159
MDB_RAD_FG_DEPAR_AT_SSMI_BODY, 159
MDB_RAD_OBS_AT_SSMI_BODY, 159
MDB_RAD_OBS_ERR_AT_SSMI_BODY, 159
MDB_RADCOST_AT_SSMI_BODY, 158
MDB_RAIN_AT_SSMI_BODY(1:2), 159
MDB_RANK_AN_AT_BODY, 153
MDB_RANK_CLD_AT_BODY, 153
MDB_RETTYPE_AT_HDR, 151
MDB_satem_Dataproc_AT_HDR, 152
MDB_satem_Instdata_AT_HDR, 152
MDB_SATID_AT_SAT, 155, 162
MDB_SATINST_AT_HDR, 152
MDB_SATINST_AT_POOLMASK, 149
MDB_SATNAME_AT_HDR(2), 152
MDB_SEGMENT_SIZE_X_AT_SATOB, 156
MDB_SEGMENT_SIZE_Y_AT_SATOB, 156
MDB_SENSOR_AT_POOLMASK, 149
MDB_SHEAR_AT_SATOB, 156
MDB_SHEARING_AT_HDR, 151
MDB_SOURCE_AT_HDR, 151
MDB_STATION_TYPE_AT_HDR, 152
MDB_SUBSETNO_AT_HDR, 152
MDB_SUBTYPE_AT_HDR, 152
MDB_SUBTYPE_AT_POOLMASK, 149
MDB_SURFEMISS_AT_BODY, 153
MDB_SURFPRESS_AT_SSMI(1), 158
MDB_T200_AT_SATOB, 156
MDB_T500_AT_SATOB, 156
MDB_T_AT_SATOB, 156
MDB_TARGET_AT_INDEX, 150
MDB_TB_AT_SATOB, 156
MDB_THINNINGKEY_AT_HDR(:), 152
MDB_THINNINGTIMEKEY_AT_HDR(152)
MDB_TIMESLOT_AT_TIMESLOT_INDEX, 150
MDB_TOP_MEAN_T_AT_SATOB, 156
MDB_TOP_WV_AT_SATOB, 156
MDB_TSLot_AT_INDEX, 150
MDB_TSLot_AT_POOLMASK, 149
MDB_U10M_AT_SSMI(1), 158
MDB_U_BEST_AT_SATOB, 157
MDB_U_OLD_AT_SATOB, 157
MDB_UWI_ESA_AT_SCATT(2), 157
MDB_V10M_AT_SSMI(1), 158
MDB_V_AT_SATOB, 157
MDB_V_BEST_AT_SATOB, 157
MDB_V_OLD_AT_SATOB, 157
MDB_VARNO_PRESENCE_AT_HDR, 152
MDB_WEIGHT_AT_SATOB(:), 156, 157
MDB_WORDNO_AT_DDRS, 149
MDB_WVC_QF_AT_SCATT_BODY, 158
MDB_XYZ_AT_BODY, 163, 164
MDB_ZENITH_AT_SATOB, 156
MDB_ZONE_AT_HDR, 151
MDBALT, 151
MDBBOX, 151
MDBDAT, 151
MDBDUBLE, 153
MDBDEV1, 153
MDBDEV2, 153
MDBST, 153
MDBESQ, 153
MDBETM, 151
MDBFGC1, 153
MDBFGC2, 153
MDBFGE, 154
MDBFL, 153
MDBFOE, 154
MDBFOVA, 159, 162
MDBHLSA, 159
MDBHOAND, 151
MDBHOANP, 151
MDBHOANS, 151
MDBHOBAD, 151
MDBHOBAT, 151
MDBHOBAS, 151
MDBHOAB, 151
MDBHOABAS, 151
MDBI1C1(IUPD), 154
MDBI1C2(IUPD), 154
MDBINS, 151
MDBIOM0(IUPD), 154
MDBIOMN(IUPD), 154
MDBLAT, 151
MDBLON, 151
MDBLSQA, 159
MDBMOR, 151
MDBOER, 154
MDBOMF, 153
MDBOMN, 153
MDBON, 151
MDBORNO, 158
MDBOTP, 151
MDBPER, 154
MDBPILID, 154
MDBPNAS, 158
MDBPP2A, 159
MDBPP, 153
MDBPRL, 153
MDBRBLE, 152
MDBRB, 153
MDBRBP0, 153
MDBRBC, 153
MDBDRDF, 153
MDBRER, 154
MDBREV1, 151
MDBREV2, 152
MDBRF, 152
MDBRO3C1, 155
MDBRO3CP, 155
MDBRO3F0, 155
MDBRO3I, 155
MDBRO3L1, 155
MDBRO3LNL, 156
MDBRO3PS, 155
MDBRO3L, 156
MDBRO3R, 156
MDBRST, 151
MDBSABA, 159
MDBSAZA, 159
MDBSBCMM, 156
MDBSBPT, 156
MDBSBIUP, 156
MDBSBI1, 156
MDBSCBAA, 157
MDBSCBIA, 157
MDBSCCNO, 157
MDBSCDIS, 157
MDBSCIRF, 157
MDBSCIR, 157
MDBSCKP, 157
MDBSCLA, 159, 162
MDBSCCP, 157
MDBSCS, 157
MDBSID, 151
MDBSKT, 159
MDBSLNO, 158
MDBSOBA, 159
INDEX

MDBSOZA, 159
MDBSSANP, 159
MDBSSCBW, 159
MDBSSCCF, 159
MDBSSIA, 151
MDBSSRB, 153
MDBSTD, 151
MDBSTHA, 159
MDBSYPC, 154
MDBTELID, 154
MDBTLA, 151
MDBTO, 151
MDBTORB, 153
MDBTORBA, 153
MDBTOS, 158
MDBVAR, 153
MDBVCO, 153
MDBVNM, 153
MDBVS3..4A, 159
MDBVSG, 158
MDBZAN, 158
MDIDB, 122, 123, 126
Megabyte, 263
MERGEODB, 194, 263
METEOSAT, 203, 204
METVIEW, 263
MF, M-F, 263
MFlops, 263
MGEOSATN_GMS, 204
MGEOSATN_GOES, 204
MGEOSATN_INDSAT, 204
MGEOSATN_METEOSAT, 204
MGEOSATN_MODIS, 204
MGEOSATN_MSG, 204
min(x,y), 45
Minimization, 263
MIPAS, 263
MIPS, 263
MKCMARPL, 263
MLNK_ATOV5ATOVS_PRED, 160, 162
MLNK_DESC2DDRS, 148
MLNK_DESC2POOLMASK, 148
MLNK_DESC2TIMESLOT_INDEX, 148
MLNK_INDEX2HDR, 150
MLNK_SAT2ATOVS, 155, 162
MLNK_SAT2REO3, 155, 162
MLNK_SAT2STATEM, 155
MLNK_SAT2SATOBS, 155, 162
MLNK_SAT2SCATT, 155
MLNK_SSMI2SSMI_BODY, 158
MLNK_TIMESLOT_INDEX2INDEX, 150
MLNKH2B, 115, 168
MMETHOD_COM_SPEC, 204
MMETHOD_IR, 204
MMETHOD_VIS, 204
MMETHOD_WVCL, 204
MMETHOD_WVMIX, 204
MMETHOD_WVMW, 204
mnemonic, 208–207
MOBHDR, 115, 168
MOBODY, 115
MOBSU, 115
mod, 45
mode, 59
Model variables, 263
MODIS, 204, 263
module/ecsort.F90, 247
MPI, 48, 51, 56, 63, 65, 67, 72, 73, 263
MPI I/O, 263
MPI-task, 18
mpirun, 48
MPL, 21, 63, 264
MSSENSOR_AIRS, 203
MSSENSOR_AMSUA, 203
MSSENSOR_AMSUB, 203
MSSENSOR_GOESIMG, 203
MSSENSOR_HIRS, 203
MSSENSOR_METEOSAT, 203
MSSENSOR_MSG, 203
MSSENSOR_MSU, 203
MSSENSOR_SSMI, 203
MSSENSOR_SSU, 203
MSSENSOR_VTPR1, 203
MSSENSOR_VTPR2, 203
INDEX

MSERIES_DMSP, 204
MSERIES_GMS, 204
MSERIES_GOES, 204
MSERIES_METEOSAT, 204
MSERIES_NOAA, 204
MSERIES_NOAA, 204
MSL, 264
MSU, 203, 264
multi-value, 103
mycompS, 111
MYDB, 176–178
MYDB.dtl, 176
MySQL, 101
myview.sql, 185
NAMELIST, 147
names, 78
NASA, 204
natural join, 95
NCAR, 264
ncols, 57, 60, 69
NESDIS, 264
NEW, 64, 118
newodb, 176, 177
newvalue, 74
next output, 93
nint(x), 45
NOAA, 204, 264
NOT, 37
NOT, AND, OR, 38
NOT, , 37
NOT IN, 37
NOT LIKE, NOT RLIKE, 37
nproc, 190
nrows, 58, 60, 72
ntid, 63
NVNUMB, 205–207
o_(), 117
o_(it), 114
Observation operator, 264
Observed variables, 264
OBSPROC, 264
hdr, 59, 199–203
ODB, 51–53, 56, 59, 63, 64, 67, 76, 117, 121,
125, 132, 247, 264
odb.F90, 50, 63
odb/scripts, 242
odb/scripts/make.xxxx, 241
ODB/SQ L, 3, 5, 13, 16, 21–25, 27, 32, 35–37,
40, 42–44, 87, 94, 95, 97, 102–104,
137, 138, 165, 264
ODB2BUFR, 165, 239
ODB2ECMA, 264
odb2netcdf, 187
odb98.flags, 32
odb98.x, 24
ODB_addpools, 50
ODB_addview, 50, 52, 59, 90, 103
ODB_cancel, 50, 54, 61, 90, 103
ODB_CATCH_SIGNALS, 189, 191
ODB_CCMA_CREATE_DIRECT, 189
ODB_CCMA_IGNORE_TSLots, 189
ODB_CCMA_OBS_RANDOMIZE, 189
ODB_close, 50, 55, 68, 82, 86, 90, 103, 106,
190
ODB_COMPILER_FLAGS, 26, 32
ODB_CONSIDER_TABLES, 22, 190, 192
ODB_CTX_DEBUG, 190, 191
odb_CY28R2.tar.gz, 241
ODB_DATAPATH, 85, 111, 190, 191, 194
ODB_DATAPATH_Sdbname, 111, 190, 191
ODB_EDITOR, 185
ODB_end, 50, 56, 82
ODB_ERRTRA, 189–191
ODB_FLPCHECK, 191
ODB_get, 49, 50, 56, 68, 69, 90, 103, 115,
195, 245, 246
ODB_getnames, 50, 58, 90, 94, 103
ODB_getsize, 50, 60, 86
ODB_getval, 50, 62, 75, 77, 86, 97
ODB_iget, 115
ODB_IGNORE_POOLMASKING, 191, 194
ODB_init, 50, 63, 82
INDEX

ODB_INTCHECK, 191
ODB_IO_FILESIZE, 31, 191, 192
ODB_IO_GRPFSIZE, 31, 192
ODB_IO_METHOD, 70, 76, 130–132, 190, 191
ODB_put, 115
ODB_load, 132
ODB_open, 22, 50–52, 54, 55, 57, 59, 61, 62, 64, 66, 69–72, 75–78, 82, 86, 89, 103, 111, 190–192, 194
ODB_PACKING, 192, 193
ODB_PACKING_MAP_$n, 193
ODB_PERMANENT_POOLMASK, 191–193
ODB_print, 50, 66, 94
ODB_put, 49, 50, 57, 68, 86, 103, 104, 106, 115, 245, 246
ODB_release, 50, 68, 70, 79, 90, 106
ODB_remove, 50, 71, 107
ODB_REPRODUCIBLE_SEQNO, 194
ODB_select, 49, 50, 54, 56, 61, 72, 90, 97, 98, 101, 103, 195
ODB_setval, 43, 50, 62, 74, 77, 97, 98
ODB_SRCPATH, 85, 111, 190, 194
ODB_SRCPATH_dbname, 111, 194
ODB_SRCPATH_ddbname, 31
ODB_STATIC_LINKING, 52, 82, 194
ODB_store, 68
ODB_swapout, 50, 68, 70, 75, 90, 106
ODB_TEST_INDEX_RANGE, 195
ODB_TRACE_FILE, 195
ODB_TRACE_FLUSH_FREQ, 195
ODB_TRACE_PROC, 189–191, 195
ODB_varexist, 50, 77
ODB_varindex, 50, 78, 103, 107, 116
ODB_WRITE_EMPTY_FILE, 192
ODB_WRITE EMPTY FILES, 196
odbcomp, 24, 25
odbcompsch, 26
odbf90, 47
odbf90mp, 48
odbprint.F90, 50
ODBSHUFFLE, 264
odbutil.F90, 50
odbviewer, 4, 35
OI, 264
OLD, 64, 118
old_npools, 51
oldvalue, 101
OML, 63, 130, 264
ONELOOPER, 38
OPENDB, 117, 129
open.F90, 118
OpenMP, 5, 115, 190, 265
openmp_objs, 114
OPTRON, 266
OR, 37, 96
ORDERBY, 38, 42, 53, 58, 247
ORDERBY, ORDER BY, 38
ORDERBY, ORDER BY, 42
OSE, 265
outnames, 59
output, 92, 106
output listing, 89, 109
output listing, 85
packable column, 193
PAOB, 265
paral, 46
parallel queries, 21
parameterized query, 100
parameterized variable, 86
paramquery.inp, 101
paramquery.sql, 100
paramquery1.F90, 97, 100
paramquery1.out, 100
paramquery2.F90, 99, 101
paramquery2.out, 101
parcma.F90, 156
parent, 14, 83
participating pools, 52
Passive data, 265
permanent, 25, 194
permanent database, 82
pgf90, 19
PILOT, 265
PINFO, 122, 124
pk1int, 30
pk9real, 30
Platform, 242
pool_excess, 51
poolmask, 149, 161
poolno, 52, 70, 71
populate.F90, 87
populate.out, 89
populate.usage, 88
pow(x,y) or, 45
Preconditioning, 265
PREODB, 6, 166, 175, 178, 239, 265
PREODB.dd, 178
PREODB.ddl, 166
PREODB.sch, 178
PREPIFS, 265
previous program, 97
PRIMARY KEY, 15
PUTATDB, 128, 129
putatdb.F90, 129, 165, 174
PUTDB, 118, 120, 123, 127, 129, 172
putdb.F90, 124, 165
QC, 265
QTAR, 192
qtar, 192
queries, 94
query.rpt, 17
query.sql, 23, 24
querydata.F90, 90, 94, 97
querydata.usage, 91
querydata1.sql, 91
querydata12.inp, 92
querydata12.out, 92
querydata12onego.out, 93
querydata2.sql, 91
QuikScat, 265

rad, 46
rad2deg, 46
radians, 46
rc, 59
RDB, 265
read, 192

READONLY, 35, 38, 55, 65, 68, 70, 76, 89, 104, 118
REAL, 62, 75
REAL(8), 50, 56, 68, 245
recognized, 25, 194
recognized database, 82, 84
recognized packing method, 193
REFERENCES, 15
Rejected data, 265
REO3, 265
reo3, 155, 161
replicate_PE, 172
RESET ALIGN, 38
RESET ONELOOPER, 38
RISC, 61, 265
ROBDDR, 115
ROBHDR, 115, 116, 121, 128, 168
ROBODY, 115, 116
ROBSU, 115
round-robin fashion, 189
rsort32, 248
RT-codes, 266
RTCALC, 203
RTIASI, 266
RTOVS, 265
RTTOV, 266
sat, 137, 155, 161
SATBODY, 115
SATHDR, 115
sat, 137
SATOB, 266
satob, 156, 161
SATPRED, 115
save, 106
SBUV, 266
scatt, 157
scatt_body, 157
Schema, 266
schema, 83
SCIAMACHY, 266
Screening, 266
second output, 110
SELECT, 23, 36, 38, 41, 42, 53, 95
SELECT ALL, 38
SELECT DISTINCT, 38, 57
SELECT UNIQUE, 38
seqno, 83, 94–96
SET, 22, 38, 42, 43, 53, 62, 75, 77
SETACTIVEDB, 118
setactivedb.F90, 118
SET ... = ..., 37
sgimips, 242
SHUFFLEDDB, 132, 133
shuffledb.F90, 117, 132, 133
signal handler, 191
Silicon Graphics, 194
similar output, 101
simulobs2odb, 87, 89
sin, 45
single value, 102
sinh(x), 45
skeleton program, 49
skeleton.F90, 49
skintemp, 160
SORTBY, 42, 53, 58
SORTBY, SORT BY, 38
SORT BY, 42
SPACE, 44
sql, 167
SQL-commands, 86
SQL-file, 23, 101
SQL-request, 107
sqrt(x), 46
SSM/I, SSMI, 266
SSMI, 203
ssmi, 158
ssmi_body, 158
SSMIS, 266
SST, 266
SSU, 203, 266
standard C I/O-library, 192
static.c, 25
static.h, 25
statid, 94, 95
Status, 242
stderr, 190
store.F90, 117
STOREDB, 131, 132
storedb.F90, 131
string, 50, 246
Structured Query Language, 35
subqueries, 94
hdr, 197, 198, 200–203
svnmb.F90, 205
swapout.F90, 117
SWAPOUTDB, 130
swapoutdb.F90, 130
SYNOP, 266
SYNSATRAD, 266
system I/O-routines, 192
TAB, 44
TABLE, 56, 73
table join, 94, 95
tables, 174
tan, 45
tanh, 45
tar, 192
TB, 266
tcsh, 241
TCWV, 264, 266
tdiff, 45
TEMP, 266
Terabyte, 266
thefirst.x, 17
Thinning, 266
thread, 190
timeslot_index, 150, 161
TL, 266
touch, 45
TOVS, 266
traceback, 191
Trajectory, 267
TRMM, 267
TRUE, 55, 57, 67, 69, 73, 76, 77, 106, 247, 248
trunc, 45
twindow, 45
INDEX

- **type aliasing**, 40
- **TYPEDEF**, 39
- **u-wind component**, 153
- **uint**, 45
- **UNIQUEBY**, 38, 53
- **Units**, 205–207
- **UNKNOWN**, 103
- **UNMAPDB**, 130
- **unmapdb.F90**, 130
- **UPDATE**, 102, 103
- **update**, 154, 161
- **update.inp**, 102
- **update.out**, 106
- **update.sql**, 103
- **update_1**, 154
- **update_2**, 154
- **UPDATED**, 38
- **updatedb.F90**, 104
- **use**, 82
- **use odb**, 185
- **USE-statement**, 48
- **use_odb**, 81
- **use_odb_sh**, 81
- **USE odb_module**, 49
- **UTC**, 267
- **utilities/rsort32.c**, 247
- **UV**, 267
- **v-wind component**, 153
- **value**, 62
- **Variable label**, 203, 204
- **varname**, 62, 74, 77
- **varno**, 96
- **body**, 205–207
- **varno.h**, 205
- **m body**, 205
- **VarQC**, 267
- **VIEW**, 53
- **viewname**, 62, 74, 75, 77
- **VIS**, 204
- **VTPR1**, 203
- **VTPR2**, 203
- **weight[:]**, 156
- **WHERE**, 23, 38, 42–44, 53, 72, 95, 97, 102, 108, 122, 123, 126, 185
- **wind direction**, 153
- **wind speed**, 153
- **WMO**, 267
- **WMO code**, 205–207
- **writeoba.F90**, 189
- **WVCL**, 204
- **WVMIX**, 204
- **WVMW**, 204
- **x**\(^\ast\ast\)\(y\)**, 45
- **x**\(^\%\)\(y\)**, 45
- **XCHANGEDATADB**, 133
- **xchagedatadb.F90**, 132, 133, 165
- **xyz**, 163
- **body**, 163
- **hdr**, 168
- **x**\(^\wedge\)\(y\)**, 45
- **yomdb**, 117
- **yomdb.F90**, 113, 118, 164
- **yomdb_defs.h**, 114, 165–167
- **yomdb-vars.h**, 114, 165–167
- **yomtvradF90**, 203
- **YYYYMMDD**, 148, 151, 165