ODB API Training

https://software.ecmwf.int/wiki/display/ODBAPI/

Piotr Kuchta

Peter.Kuchta@ecmwf.int

ECMWF

Overview

- What is ODB API?
- Data structure / file format
- Overview of Python, C and Fortran API
- Overview of ODB API tools
- ODB API SQL specifics
- ODB Governance
- ODB MARS Archive
- Advanced functionality

What is ODB API?

- Software
  - **O**bservational **D**ata**B**ase **A**pplication **P**rogramming **I**nterface
  - C++ library & tools for encoding and processing of observational feedback
  - SQL engine for efficient processing of data encoded in the format
  - C, Fortran and Python bindings/wrappers to the above: https://software.ecmwf.int/wiki/display/ODBAPI/ODB+API+Home
- File format
  - file format suitable for archiving and streaming, used for archiving on MARS
- Highlights
  - Efficient processing of large amounts of data (long time series) using small amount of RAM
  - Used at ECMWF, MetOffice, BoM,…
- What it is not
  - ODB: incore, parallel (MPI) database used by IFS on the ECMWF HPC

Observational feedback

- Observational Feedback is a diagnostic information related to a given observation produced by the screening or assimilation system
- Observations themselves are input to IFS, the "feedback" is IFS output
- Examples:
  - **fg_depar** is the difference between the model prediction of the temperature and the observed temperature, "first guess departure" in data assimilation terminology
  - **datum_status@body** was the observation rejected in the screening?
  - **datum_event1@body** why was this observation rejected?
  - **an_depar@body** the difference between the observed value and the analysis estimate
  - **biascorr@body** the bias correction applied to the observation (the bias is estimated during the 4d-Var process)
  - **t2m@modsurf** what was the model 2m temperature at the lat,lon,time of the observation (the observation might be for another variable)

ODB API data structure / file format

- Tabular data structure, like a table in a relational database
- Line / row oriented
- Each file consists of one or more blocks, each block containing:
  - Header: column names, types, codec info, values of constant columns
  - Data section: packed sequence of rows (customizable, by default 10,000)
  - When reading files only one block needs to be unpacked into RAM at a time
- The above properties allow for
  - Processing files of arbitrary length in constant, small amount of memory
  - Data can be appended to existing files / files can be concatenated

Data format: blocks

- ODB API file consists of one or more blocks
- Each block consists of a header (metadata) and data section

ODB Governance
Report type:
16001 - Automatic Land SYNOP
varno:
110 - surface pressure,
58 - 2m relative humidity

andate :integer	antime :integer	reportype :integer	date :integer	time :integer	varno :integer	obsvalue :real	….
20170108	120000	16001	20170108	150000	110	101220.00	
20170108	120000	16001	20170108	150000	58	0.992837	
20170108	120000	16001	20170108	150000	110	100110.00	
20170108	120000	16001	20170108	150000	110	100870.00	
20170108	120000	16001	20170108	150000	110	100980.00	
andate :integer	antime :integer	reportype :integer	date :integer	time :integer	varno :integer	obsvalue :real	….
20170108	120000	16001	20170108	150000	110	101230.00	
20170108	120000	16001	20170108	150000	58	0.992852	
20170108	120000	16001	20170108	150000	110	100110.00	
20170108	120000	16001	20170108	150000	110	100860.00	
20170108	120000	16001	20170108	150000	110	100550.00	

Header

Data

Header

Data

Files with heterogeneous metadata

- Over time new columns are added to ODB. When retrieving from MARS data spanning multiple cycles it may happen that part of the retrieved data has more columns than the rest.

andate :integer	antime :integer	reportype :integer	date :integer	time :integer	varno :integer	obsvalue :real
20170108	120000	16001	20170108	150000	110	101230.00
20170108	120000	16001	20170108	150000	58	0.992852
20170108	120000	16001	20170108	150000	110	100110.00
20170108	120000	16001	20170108	150000	110	100860.00
20170108	120000	16001	20170108	150000	110	100550.00

andate :integer	antime :integer	reportype :integer	date :integer	time :integer	varno :integer	obsvalue :real	new_flag :bitfield
20170109	120000	16001	20170109	150000	110	101230.00	1
20170109	120000	16001	20170109	150000	58	0.992852	1
20170108	120000	16001	20170108	150000	110	100110.00	0
20170108	120000	16001	20170108	150000	110	100860.00	0
20170108	120000	16001	20170108	150000	110	100550.00	1

Overview of C, Fortran and Python APIs

- Python API
  - `import odb`
  - PEP 249 -- Python Database API Specification v2.0
  - Built on top of the new C API
- New C/C++ API
  - Pure C, can be used from C++ and other languages
  - Modelled after sqlite3 API
  - C/C++ examples: https://software.ecmwf.int/wiki/pages/viewpage.action?pageId=61117603
- Fortran
  - ISO Bindings / wrappers (e.g. convert Fortran strings to C) to the new C API
  - Uses Fortran 2003 features
  - https://software.ecmwf.int/wiki/display/ODBAPI/Fortran+examples

Example Python script

```
import odb
conn = odb.connect('')
c = conn.cursor()
c.execute('''CREATE TABLE  foo  AS
                ( x INTEGER, y DOUBLE, v STRING)
                ON 'new_api_example.odb'; ''')
c.executemany('INSERT INTO foo (x,y,v) VALUES (?,?,?);',
              [[1, 0.1, '  one   '],
               [2, 0.2, '  two   '],
               [3, 0.3, '  three '']])
conn.commit()
```

Define metadata with CREATE TABLE
Associate SQL table with file.

Prepared statement INSERT to insert multiple rows

```
c.execute('SELECT * FROM foo')
for row in c.fetchall():
    print row
```

SELECT data from file associated with SQL table foo.

ODB API Command line tools

- At ECMWF:
  - $ module load odb_api/new
- Most tools available via single binary odb
  - $ odb *command options parameters*
- Running odb without any arguments prints out list of commands
- Command help prints detailed help for a given command:
  - $ odb help sql # print detailed information on sql command's options
- Online documentation: https://software.ecmwf.int/wiki/display/ODBAPI
- Reporting issues: https://software.ecmwf.int/wiki/display/ODBAPI/Reporting+an+issue

Execute SQL on command line

- SQL can be passed to the tool in a file or directly on command line:
  - $ odb sql query.sql # query.sql is a text file with a SELECT
  - $ odb sql select * -i conv.odb # no FROM clause, input provided with -i
  - $ odb sql ‘select *’ -i conv.odb # as above, but SELECT statement quoted
- By default result set is printed in a text format to stdout
  - -o *file* sends output to *file*
  - -f odb will produce output in ODB API format
- Few more options available, see
  - $ odb help sql

ODB API SQL specifics

- Types
  - REAL, single precision (32bits) floating point number,
  - STRING, for text columns. Currently limited to 8 characters.
  - INTEGER, 32 bits signed integer.
  - BITFIELD, for efficient encoding of flags.
  - DOUBLE, double precision (64 bits) floating point numbers.
- Scalar functions:
https://software.ecmwf.int/wiki/display/ODBAPI/Scalar+functions
- Aggregate functions:
https://software.ecmwf.int/wiki/display/ODBAPI/Aggregate+functions

ODB API SQL specifics: bitfields

- Output of ‘odb header’ contains definitions of bitfields:

```
20. name: report_status@hdr, type: BITFIELD [active:1;passive:1;rejected:1;blacklisted:1;use_emis
21. name: report_event1@hdr, type: BITFIELD [no_data:1;all_rejected:1;bad_practice:1;rdb_reject
22. name: report_rdbflag@hdr, type: BITFIELD [lat_humon:1;lat_qcsub:1;lat_override:1;lat_flag:2;la
```

- Particular fields of a bitfield can be accessed with dot (‘.’)
  - select * from “conv.odb” where report_status.active = 1
  - select report_status.* from “conv.odb” # star expands to a list of all fields
- Defining tables with bitfields:

```
CREATE TYPE my_bitfield_type AS (active bit1, passive bit2, );
CREATE TABLE foo AS (x INTEGER, status my_bitfield_type) ON ‘file.odb';
```

- https://software.ecmwf.int/wiki/display/ODBAPI/Bitfields

ODB MARS Archive and ODB Governance

- ECMWF operational suite, experiments and Copernicus reanalysis archive observational feedback in MARS
  - RETRIEVE, TYPE=OFB/MFB, OBSGROUP=CONV, TARGET=conv.odb
- Access via:
  - command line MARS client
  - MARS Catalogue: http://*apps.**ecmwf**.int/**mars-catalogue**/*
  - ECMWF Web API
  - Metview (embedded MARS client)
  - ODB API since 0.16.0 (experimental)
- ODB Governance: *http://apps.ecmwf.int/odbgov/reporttype/*
  - description of codes, column names, units

MARS language

- MARS language can describe data and, optionally, post processing like interpolation (gridded data) or filtering:
  - *verb, atr1=v1, atr2=v2, atr3=v3/v4/v5*
  - For example: RETRIEVE,TYPE=MFB,OBSGROUP=CONV,TARGET=conv.odb
  - https://software.ecmwf.int/wiki/display/UDOC/MARS+user+documentation
- Keywords and their values relevant for observational feedback data:
  - TYPE = OFB / MFB # Observational Feedback, Monitoring Feedback (or MONDB feedback)
  - REPORTTYPE = ...
    http://apps.ecmwf.int/odbgov/reporttype/
    http://apps.ecmwf.int/odbgov/unionall/ (all details)
  - FILTER = “SELECT … WHERE …”
  - OBSGROUP = HIRS/AMSUA/AMSUB/ (groups of report types)
    http://apps.ecmwf.int/odbgov/group/

OFB vs MFB

- OFB (Observational Feedback)
  - all observations screened by IFS
- MFB (Monitoring or MONDB Feedback)
  - datum_status.active = 1
    Observations that made it through screening and were active in 4DVar, ~10% of all
  - Added FSO (Forecast Sensitivity towards Observations)
    - an_sens_obs@body how the observation helped in the analysis
    - fc_sens_obs@body how the observation helped in the forecast

MARS command line example

Compute standard deviation of forecast departure for all assimilated observations measured by the AMSU-A instrument on the METOP-A satellite for the year 2015, separately for all frequency channels.

```
$ mars <<END
retrieve, type = ofb, reportype = 1007, date = 20150101/to/20160101,
target  = stdev_by_channel.odb,
filter = "select vertco_reference_1, stdev(fg_depar) where datum_status.active=1"
END
```

MARS request with embedded SQL (streaming processing)

```
$ odb sql select \* order by 2 -i stdev_by_channel.odb
```

SQL on command line

vertco_reference_1@body	stdev(fg_depar)
6	0.150481
9	0.189032
10	0.228713

MARS retrieval: server indexing

- MARS keywords used for selecting ODB data to be retrieved correspond to ODB columns used by server index

MARS Keyword	ODB Column	
DATE	andate	analysis date
TIME	antime	analysis time
REPORTYPE	reportype	Report type name or id (ODBGOV)
OBSGROUP	groupid	Group name or id (ODBGOV)
TYPE	type	OFB, MFB
CLASS	class	
STREAM	stream	
EXPVER	expver	Experiment version

MARS retrieval: filtering with SQL

- Keyword FILTER can be used for fine grained filtering with SQL SELECT
- Filtering is done on the client, directly on the data coming from the server (no temporary files)
- If you plan to execute many SQL queries on data retrieved from MARS, it is better to retrieve the whole dataset (no FILTER) to your local disk and then execute the queries using ‘odb sql’

Advanced, new and experimental features

- Splitting functionality integrated into SQL
- MARS client functionality in ODB API
- ECML, embedded MARS language parser and interpreter
- ODB Server functionality for local files embedded in ODB API
- Server side processing

Splitting functionality integrated into SQL

- File name in the INTO clause becomes a file name template if it contains column names in curly brackets

```
$ cat split_by_rt.sql

SELECT *
INTO "conv/{andate}/{reportype}.odb"
FROM "conv.odb";

$ odb sql split_by_rt.sql
000 2017-03-09 17:44:04 (I) 0: creating 'conv/20170308/16001.odb'
000 2017-03-09 17:44:04 (I) 1: creating 'conv/20170308/16002.odb'
000 2017-03-09 17:44:04 (I) 2: creating 'conv/20170308/16004.odb'
000 2017-03-09 17:44:04 (I) 3: creating 'conv/20170308/16005.odb‘
…
```

MARS client functionality

```
SELECT *
FROM "mars://RETRIEVE,CLASS=OD,TYPE=MFB,STREAM=OPER,
    EXPVER=0001,DATE=20170308,TIME=1200,REPORTYPE=16001,
    DATABASE=marsod";
```

- ODB API can retrieve data directly from MARS or ODB Server
  - Parse request in the MARS language
  - Retrieve data via network (using the metkit library)
- No defaults, expansions & checking that is done by the traditional mars client

ECML (1)

- ODB API has a parser of the MARS language
- ECML – interpreter of an extended MARS language
  - It can parse and execute classic MARS requests like RETRIEVE
  - New verbs can be added in C++ or using verb FUNCTION
  - Requests can be composed

```
let, input = "conv.odb"
sql, filter = "select varno,sum(varno_count) as varno_count",
     source = (for, c = (chunk, source = (input)),
                    do = (sql,
                          source = (c),
                          filter = "select varno,count(*) as varno_count",
                          target = (temporary_file))),
     target = out.odb
```

  - Verb **for** is parallel (OpenMP)

```
$ OMP_NUM_THREADS=8 odb ecml chunk_example.ecml
```

ECML (2)

- Some of the currently built in verbs:
  - **retrieve**, **archive**
  - **let** introduces one or more new variables. Its value is a dictionary (request).
  - **sql** executes SQL statement
  - **compare** two ODB files
  - **test** used for testing
  - **split** for splitting ODB files (as the command line)
  - **for** execute a request passed in **do** for all elements of a list, in parallel (if OpenMP available in compile time and OMP_NUM_THREADS set)
- ECML is designed to be an embedded data processing strategy engine
- ECML is currently used for testing ODB API (see files with extension .ecml)

ODB Server functionality for local files

- Data descriptor “local://”
  - Similar to “mars://” but for local files
  - Maps data described in request to a list of files according to pathname schema

```
SELECT *
FROM "local://retrieve,
date=20150218,
time=1200,
reportype=16058,
odbpathnameschema = '{date}/{time}/{reportype}.odb',
odbserverroots = "~/data/root",
database=localhost"
```

Server side processing

- Needs to be enabled during MARS / ODB Server compilation
- Keyword SERVER_SIDE supported by ODB API, not by the standard client
- Allows to process data on the server and send results to the client
  - Often transferring data is more expensive than its processing

Server side processing: example

```
retrieve, class = OD, date = 20150218, time = 1200, type = OFB, obsgroup = conv,
          reportype = 16058, stream = oper, expver = qu12,
          server_side = (
            # Value passed to server_side should be a closure.
            # This closure will be executed by the server to process data
            # described by the request before it is send to the client.
            function, of = source,    # “source” is the name of the fun. parameter
                      _ = (sql,       # verb sql will find “source” in its dynamic scope
                               filter = "select varno, count(*) order by varno",
                               target = (temporary_file)))
                                       # processed data will be saved to a temporary file
                                       # Verb sql returns file name passed as target;
                                       # Value of the last expression in function body is
                                       # the return value of the function.
                                       # Value returned from the server_side closure
                                       # should be a list files (one element in the ex.).
                                       # Server will send content of those files to client
```

Exercise

- From yesterday's 12 UTC operational run find out which variables are contained in report type "Automatic Land SYNOP" of type Monitoring feedback (MFB).
- What is the range of values of those variables?
- Tips:
  - Retrieve data with mars
  - Numeric code of variable is available in ODB column `varno`
    See http://apps.ecmwf.int/odbgov/varno/ for codes & descriptions
  - Actual value of the variable (measurement) is in column `obsvalue`
  - `module load odb_api/new`

Solution

```
$ mars <<END
    retrieve,
    class = OD,
    type = MFB,
    stream = OPER,
    expver = 1,
    reportype = Automatic Land SYNOP,
    date = -1,
    time = 1200,
    target  = als.odb
END
$ module load odb_api/new

# Show list of variables:
$ odb sql ‘select distinct varno’ -i  als.odb

# List of variables and their ranges:
$ odb sql ‘select varno,min(obsvalue),max(obsvalue)’ -i als.odb
```

Instead of
“Automatic Land SYNOP”
We could put its numeric
code: 16001