Feedback File Definition

June 15, 2012

Contents

List of Tables

Changes to this document

Date	Version	Change
2012-01-26		Implement variables for radar operator, VN_VLS: line of sight wind.
2012-01-04		Add comments, update f90 interface documentation.
2011-12-05	1.01	Change variable 'phase' from BYTE to SHORT, use for GPSRO PCD as well.
		Write variable 'plev' for AIREPs as well.
		New body entry 'accuracy': accuracy from data provider.
		Use optional variable 'plevel' for PILOTs as well.
		Write data base time 'time_dbase' to NetCDF feedback file.
		New optional parameters 'ierr', 'fill') for subroutines get_var_real_2, read_fdbk_veri: error return parameter, value to replace with fillvalue.
		Modifications for writing COSMO feedobs files,
		Change 'level_sig' to SHORT to hold WMO instrument numbers (instead of RTTOV).
		New routines 'write_veri', 'write_fdbk_var', add error handling.
		New flag values:
		codetype OC_WP_JP = 134: Japanese wind profilers,
		'varno' parameter VN_FLEV: nominal flight level for Aircrafts,
		status flag ST_OBS_ONLY: observations only, no model equivalent available,
		VE_ENS_MEAN_OBS: ensemble mean taken in observation space,
		FL_NO_BIASCOR: no bias correction available,
		FL_NO_OBS: no observations in report,
		FL_OPERATOR: observation operator not applicable,
		VE_MEMBER: generic value for complete set of ensemble members.
2011-02-08		define additional parameters tp write COSMO feedobs files.
		new flags FL_NO_BIASCOR, FL_NO_OBS.
		make variable 'sun_zenith' mandatory for all observation types.
		use optional variable 'tracking' also for AIREP observations.
		new Fortran 90 interface module mo_fdbk_io.
2010-07-28		add overview section, update f90-interface description.
2010-06-08		meaning of veri_initial_date and veri_forecast_time clarified.
2010-01-12	1.00	changes for RADAR data:
		new header entries range_bin_size, varno_fallback, make varno optional.
		make level, level_typ, level_sig optional.
		define radar specific table, entries:
		radar_azimuth, radar_elevation, radar_range.
		new dimension and attribute: d_radar, n_radar.
2009-11-12	0.99	radiances: use phase for Field Of View index.
		new body entry sat_zenit : satellite zenith angle.
		new body entry sun_zenit : sun zenith angle.
		make w_vq an entry in veri_data (Id: WQC_WEIGHT).
2009-05-04	0.99	type of variable `instype` changed from `byte` to `short`
2009-04-23	0.99	new variables: mdlsfc, qual, plevel.
		removed variables: mdlsf, pcc, snr.

1 Overview

Feedback files hold information on observations and their usage in the data assimilation system:

- Type of observation
- Coordinates of observation
- Specification of the measurement instrument or station
- Observed values
- Background values used in the data assimilation
- Analysed value
- Usage in the assimilation system (i.e. used, rejected, etc.)
- Bias correction used (if applicable)
- Weight in variational quality control (if applicable)
- Optionally: Values of forecasts with different lead times
- ...

Feedback files are currently used to gather information for verification and innovation statistics in the global data assimilation system (GME) and shall be used in the regional data assimilation (COSMO) as well. They are also used in the COSMO ensemble data assimilation system (KENDA) to pass information on observations and first guess (taken at the appropriate time) from the COSMO model do the LETKF.

This document describes the format and content of the NetCDF feedback files (Sections 2 and 3), provides hints for the migration from the COSMO VOF files to this format (Section 4), a description of the Fortran 90 interface routines to read and write these files (Section 5), and an example ncdump (Section 6).

2 Feedback File Definition

2.1 Global Attributes

Global attributes following the NetCDF Climate and Forecast (CF) Metadata Conventions:

title (character string, len=28)

A succinct description of what is in the dataset. Model name and string 'Verification Data':

`'COSMO Verification Data'`

institution (character string, len=24)

Specifies where the original data was produced. In general:

`'German Weather Service'`

source (character string, len=16)

The method of production of the original data. If it was model-generated, source should name the model and its version, as specifically as could be useful. For instance:

`'COSMO Version Y.Z'`

history (character string, len=muliple of 80)

Provides an audit trail for modifications to the original data. Well-behaved generic netCDF filters will automatically append their name and the parameters with which they were invoked to the global history attribute of an input netCDF file. We recommend that each line begin with a timestamp indicating the date and time of day that the program was executed:

```
'2007-11-23 01:23 COSMO 2007112300 deterministic_fc '
'2007-11-23 01:23 COSMO 2007112300 deterministic_fc '
'2007-11-23 02:24 COSMO 2007112300 Ensemble-member_001'
'2007-11-23 03:45 COSMO 2007112300 Ensemble-member_002'
'2007-11-23 02:24 LETKF 2007112300 '
```

Each entry uses 80 characters, seperated by newline characters.

The following CF recommendations are not used:

references

Published or web-based references that describe the data or methods used to produce it.

comment

Miscellaneous information about the data or methods used to produce it.

In addition to the CF recommendations the following global attributes are set:

file_version_number (character string, len=5)

'01.00'

Feedback file specific attributes:

n_hdr (integer)

Number of report entries actually used. May be smaller than the allocated dimension d_hdr.

n_body (integer)

Number of body entries actually used. May be smaller than the allocated dimension d_body.

n_radar (integer)

Number of RADAR observation specific entries actually used. May be smaller than the allocated dimension d_radar.

verification_ref_time (integer, hhmm)

verification_ref_date (integer, yyyymmdd)

Verification reference time and date.

In case of a forecast reference time is the start of the forecast period.

In case of the nudging scheme reference time is the end of the assimilation period.

In case of 3D-Var or LETKF reference time is the analysis time.

verification_start (integer, minutes)

verification_end (integer, minutes)
Start and end of the verification period with respect to `verification_reftime`.

resolution (float(2))
Model resolution in degree, seperately for the x- and y-dimension.

domain_size (integer(3))
COSMO: nx, ny, nz
GME: ni, ni, nz

For the COSMO model:

pole_lat_lon (float(2))

lower_left_lat_lon (float(2))

upper_right_lat_lon (float(2))
Coordinates of the rotated pole and the edges of the model domain (degree).

2.2 Dimensions

name	description
d_hdr	number of reports in the file.
d_body	number of observations.
d_radar	size of RADAR observation specific table. The allocated size of `d_hdr`, `d_body` and `d_radar` may be larger than the number of data items actually stored. The latter is given in the global attributes `n_hdr`, `n_body` and `n_radar`.
d_veri	number of verification runs. This dimension is the unlimited dimension, so that further verification runs can be added.
d_2,d_3	Arrays of length 2 or 3
char10,char12,char64	length of character strings

2.3 Data

Data is either of dimension (d_hdr) for the report header or of dimension (d_body) for data related to each observation, or of dimension (d_veri, d_body) for verification data. There are observation operator specific tables (RADAR operator so far) of size (d_radar).

2.3.1 Report Headers

Data is of dimension (d_hdr) and provides information related to each report (single level report, multi level report, sattelite field of view.

Data may be optional (cf. column `opt.` in the table below). Optional data may be restricted to a model (COSMO, GME), to the assimilation scheme (3DVAR, LETKF) or to certain observation operators (TEMP, PILOT, . . .).

name	type	opt.	units	description
`i_body`	`int`			Index of the data in the corresponding report body. For example: 1: for the first report n+1: for the second report if the first oneconsists of n observations.
`l_body`	`short`			Number of observations in the corresponding report body. n for the first report in the example above.
`n_level`	`short`			Number of different levels in the report body.
`i_spec`	`int`	RADAR		Index of the data in the corresponding report type specific table. For example: 1: for the first report of a given obstype n+1: for the second report of this obstype if the first oneconsists of n observations.
`l_spec`	`short`	RADAR		Number of observations in the corresponding report specific table n for the first report in the example above.
`data_category`	`short`		WMO	BUFR 4 data category.
`sub_category`	`short`		WMO	BUFR 4 data sub-category.
`center`	`short`		WMO	Station processing center.
`sub_center`	`short`		WMO	Station processing sub-center.
`obstype`	`byte`		Table 1	Observation (report) type.
`codetype`	`short`		Table 2	Observation code type.
`ident`	`int`			Station ID as integer. Not applicable for aircraft reports or ship TEMP reports.
			WMO C-2	Satellite ID for satellite observations.
`statid`	`char`			Station id in character form. (10 char)
`lat`	`float`		degree	Latitude of observation.
`lon`	`float`		degree	Longitude of observation.
`time`	`short`		min	Observation time minus reference time.
`time_nomi`	`short`		min	Nominal (synoptic) observation time minus reference time
`time_dbase`	`short`		min	Data base time minus reference time. At DWD decoding time. Used to monitor/simulate data available within a certain cutoff time.
`z_station`	`short`		m	Station height.
`z_modsurf`	`short`		m	Model surface height.
`sun_zenit`	`float`		degree	Sun zenith angle.
`r_state`	`byte`		Table 3	Status of the report.
`r_flags`	`int`		Table 4	Quality check bit flag.
`r_check`	`byte`		Table 4	Check which caused rejection of the report. One value out of `r_flags`.
`sta_corr`	`byte`			Station correction indicator (1: yes; 0: no).
`index_x`	`short`			model index x of grid point to which the report is assigned.
`index_y`	`short`			model index y of grid point to which the report is assigned.
`mdlsfc`	`byte`		Table 17	Model surface characteristics.
`instype`	`short`		Table 9	TEMP: Radiosond type
			WMO C-8	GPSRO, RAD: Classification of Sattelite instruments (AMSUA, AMSUB).
			WMO 002149	BUOY: type of data buoy.
			WMO 002001	SYNOP, AIREP: type of station.

name	type	opt.	units	description
retrtype	short	SATOB	Table 11	Station retrieval type.
tracking	byte	TEMP PILOT AIREP	Table 10	Tracking technique.
meas_type	byte	TEMP PILOT	Table 12	Type of measuring equipment used.
rad_corr	byte	TEMP	Table 13	Solar and infrared radiation correction.
phase	byte	AIREP	Table 18	Aircraft phase: Bit flags from BUFR report.
		RAD SCATT GPSRO		Radiances, Scatterometer: Field of View number
		RAD		Radiances: Field of View index.
flg_1dvar	byte	RAD	Table 16	Bit pattern for 1DVAR processing flags.
flg_cld	byte	RAD	Table 15	Bit pattern for 1DVAR cloud flag.
surftype	byte	RAD	Table 14	Bit pattern for surface type.
sat_zenit	float	RAD	degree	Satellite zenith angle.
varno_back	short	RADAR	Table 5	Type of the observed quantity. Used in case that all observations of the report are of the same type. In that case body entry varno may be not present.
range_bin_size	float	RADAR	m	Spacing of distance from radar location.
obs_id	int	3DVAR		Unique observation Id in the assimilation program. Remaines unchanged if a different number of processors is used.
source	byte	3DVAR		Input file number.
record	int	3DVAR		Record number of the report in the original input (BUFR) file.
subset	short	3DVAR		Subset number of the report in the original input (BUFR) file.
dbkz	short	3DVAR		DWD data base id.
index_d	byte	GME		model diamond index to which the report is assigned.

2.3.2 Report Bodies

Data fields with dimension (d_body) for data related to individual observations:

name	type	opt.	units	description
`obs`	`float`			Bias corrected observation.
`bcor`	`float`			Bias correction (corrected - observed value).
`e_o`	`float`			Observational error.
`state`	`byte`		Table 3	Status of the observation.
`flags`	`int`		Table 4	Bit flag table.
`check`	`byte`		Table 4	Check which caused rejection of the observation. One value out of `flags`.
`qual`	`short`			Observation confidence from data provider:
			%	AMV: 0=very bad, 100=very good.
			Table 19	Aircraft: roll angle.
			db	Wind profiler: signal to noise ratio.
`accuracy`	`float`	PILOT GPSRO	as obs	Accuracy of observation from data provider, used for some wind profilers.
`plevel`	`float`	GPSRO RAD PILOT AIREP	Pa	Nominal height for observations which do not have a vertical pressure coordinate.
`varno`	`short`	not RADAR	Table 5	Type of the observed quantity.
`level`	`float`	not RADAR		Value of observation level. (In general pressure in Pa).
`level_typ`	`short`	not RADAR	Table 5	Type of level information.
`level_sig`	`short`	TEMP/PILOT	Table 8	Level significance.
		SYNOP	WMO 008002	Level significance for individual and general cloud type.
		RAD	WMO C-8	Instrument type for radiances.
`spec_index`	`int`	RADAR		Index of observation for the case of sparse coverage.

2.3.3 RADAR observation type specific table

A specific table is defined for volume radar data in order to allow a sparse storage of the observations and to efficiently describe the underlying coordinate system.

One RADAR report holds all data for a nominal elevation angle. Thus the observations are specified as a function of two coordinates only: azimut and distance. Entries in the observation body are only stored for valid data and the relation to the complete data set is given by variable spec_index. The storage sequence convention for the complete data set corresponds to arrys defined as (max(range), number of azimuth entries) in Fortran notation.

The entries of the RADAR observation type specific table are related to the header table entries by the variables i_spec and l_spec. The size of the table corresponds to the number of azimuth entries. Each entry holds additional information on the coordinates:

name	type	opt.	units	description
`radar_azimuth`	`float`	RADAR	degree	azimuth of ray.
`radar_elevation`	`float`	RADAR	degree	elevation of ray (may differ from nominal value).
`radar_range`	`short`	RADAR		range of ray (may be less than the maximum value).

2.3.4 Verification Data

Verification data is stored in the variable `veri_data` with dimensions `(d_body, d_veri)`. The content of `veri_data` (modelled quantities or estimates of the errors of the modelled quantities) and the type of the verification run (analysis, forecast, . . .) is specified in the accompanying variables of dimension `(d_veri)`.

name	type	units	description
`veri_data`	`float`		Modeled quantity or estimates of the error of the modeled quantity.
`veri_model`	`char(10)`		Model used for verification, i.e. 'COSMO', 'GME'
`veri_run_type`	`byte`	Table 20	Type of the verification run.
`veri_run_class`	`byte`	Table 21	Class of the verification run.
`veri_initial_date`	`char(12)`	`yyyymmddhhmm`	Start time of the 'model' run. specifically: forecast: start time of the forecast; 3D-Var and LETKF analysis: analysis time; nudging analysis: start time of the nudging run; first guess: start time of the forecast run used as the first guess at verificalion_ref_time.
`veri_forecast_time`	`int`	`hhhmm`	Forecast time at verification_ref_time . Zero for all kinds of analyses.
`veri_resolution`	`float(2)`	degree	Model resolution, seperately for the x- and y-dimension.
`veri_domain_size`	`integer(3)`		Domain size of the model. COSMO: nx, ny, nz GME: ni, ni, nz
`veri_description`	`char(64)`		Possibly more detailed description than `run_type`: 'nudging run', 'etkf analysis', ...
`veri_ens_member`	`int`	Table 22	Ensemble member number with special meaning for non-positive numbers.
`veri_exp_id`	`int`		Experiment id of the verification run.

3 Definition of Data Items

3.1 Observation and Code Types

Specification of the contents of the header variables `obstype` and `codetype`.

value	name	description	
1	SYNOP	SYNOP report,	(ECMWF convention)
2	AIREP	AIREP report,	(ECMWF convention)
3	SATOB	SATOB report (AMV),	(ECMWF convention)
4	DRIBU	DRIBU report,	(ECMWF convention)
5	TEMP	TEMP report,	(ECMWF convention)
6	PILOT	PILOT report,	(ECMWF convention)
7	SATEM	SATEM report,	(ECMWF convention)
8	PAOB	PAOB report.	(ECMWF convention)
9	SCATT	Scatterometer report,	(ECMWF convention)
10	RAD	Radiances	(ECMWF convention)
11	GPSRO	GPS Radio occultations,	(DWD convention)
12	GPSGB	GPS ground based observations	(DWD convention)
13	RADAR	RADAR (volume data)	(DWD convention)

Table 1: Observation Types

value	name	description
11	SRSCD	synop surface report
14	ATSCD	automatic synop surface report
21	AHSCD	ship synop report
24	ATSHS	automatic ship synop report
140	METAR	METAR
110	GPS	GPS
141	AIRCD	airep report
41	CODAR	codar report
144	AMDAR	amdar report
87	CLPRD	cloud (height) product
88	STBCD	satob report
90	AMV	AMV
165	DRBCD	dribu report
64	TESAC	scatterometer
35	LDTCD	land temp report
36	SHTCD	temp ship report
135	TDROP	temp-drop report
37	TMPMB	temp mobile
32	LDPCD	land pilot report
33	SHPCD	ship pilot report
38	PLTMB	pilot mobile
210	ATOVS	ATOVS satellite data (1dvar)
132	WP_EU	European wind profiler
133	RA_EU	European sodar/rass report
134	WP_JP	Japanese wind profiler
136	PR_US	wind/profiler/rass report (USA)
137	RAVAD	radar VAD wind profile report
218	SEVIR	SEVIRI
123	ASCAT	ASCAT scatterometer
122	QSCAT	QSCAT scatterometer
216	AIRS	AIRS
217	IASI	IASI

Table 2: Observation Code Types

3.2 Report and observation status

value	name	description
0	ACCEPTED	active and VQC accepted (used in 3D-Var only)
1	ACTIVE	used in the assimilation
3	MERGED	not used, merged into multilevel report
5	PASSIVE	not used, only monitored
7	REJECTED	not used due to suspicious quality
9	PAS_REJ	passive and rejected
11	OBS_ONLY	observation only, no model equivalent available
13	DISMISS	dismiss observation, should not appear in file

Table 3: Observation or report status values

Observations used in the assimilation are denoted as `ACTIVE` or `ACCEPTED`. The `ACCEPTED` flag indicates that the observation obtained a weight larger than 0.5 in the Variational Quality Control.

Observations not used in the assimilation are denoted as `REJECTED` if they are dismissed due to insufficient quality (did not pass all of the quality control checks). They are denoted as `PASSIVE` if they are not assimilated but processed by the assimilation system just for monitoring purposes. The `PAS_REJ` flag indicates that passivly monitored observations did not pass the quality control checks. The status `MERGED` refers to reports which were merged into others (Airep multilevel reports, TEMP parts A,B,C,D) and are thus redundant. The status `OBS_ONLY` indicates that no model equivalent to the observation is present, either because the operator could not applied (arguments were in an unphysical range, cf. flag `OPERATOR`) or because the information is only included for verification purposes.

3.3 Quality check flags

bit#	name	description	
2	SUSP_LOCT	suspicious location or date/time	1
3	TIME	time not in valid range	2
4	AREA	location not in valid area	3
8	PRACTICE	bad reporting practice/insuff. data	4
9	DATASET	dataset qality flags	5
1	BLACKLIST	blacklist (or not whitelist)	6
5	HEIGHT	location not in valid height range	7
6	SURF	incorrect surface (land,ice,etc)	8
7	CLOUD	cloud check	9
16	GROSS	gross error flag	10
0	OBSTYPE	passive report type (at obs.location)	11
10	REDUNDANT	redundant report	12
11	FLIGHTTRACK	flight track error flag	13
12	MERGE	merged reports (e.g. TEMP ABCD)	14
13	THIN	thinning	15
14	RULE	complex rule	16
17	NO_BIASCOR	no bias correction available	17
15	OBS_ERR	observation error too large	18
19	NO_OBS	no observations in report	19
18	FG	observation - first guess check	20
21	FG_LB	obs- lateral boundary condition check	21
20	OPERATOR	observation operator not applicable	22
32	NONE	no flag set	

Table 4: Report or Observation Flags

If the report or observation flag value is `OBSTYPE` or `THIN`, the status flag will be set to `PASSIVE`. For a value of `MERGE` the status becomes `MERGED`. Other report or observation flag values will lead to a status of `REJECTED`.

The entries of table 4 are used as bit numbers (using Fortran convention, 0 being the least significant bit). If the observation has not passed a check the respective bit is raised in variables `flags` or `r_flags`. Variables `check` and `r_check` hold the number of the check that first caused the rejection of the data. In different assimilation systems the checks may be applied in different orders. The sequence number (right column in table 4 corresponds to the order used within COSMO. It may be used to derive a reasonable value of variable `check` from `flags` if not given otherwise.

The value `NONE` is only used in variables `check` and `r_check` to indicate that no bit was set in `flags` or `r_flags`.

3.4 Observation variable and level types

value	name	units	description	
0	NUM		ordinal (channel) number	L
3	U	m/s	u-component of wind	
4	V	m/s	v-component of wind	
8	W	m/s	vertical velocity	
1	Z	(m/s)**2	geopotential	(also L)
57	DZ	(m/s)**2	thickness	
9	PWC	kg/m**2	precipitable water content	
28	TRH	0..1	transformed relative humidity	
29	RH	0..1	relative humidity	
58	RH2M	0..1	2 metre relative humidity	
2	T	K	upper air temperature	
59	TD	K	upper air dew point	
39	T2M	K	2 metre temperature	
40	TD2M	K	2 metre dew point	
11	TS	K	surface temperature	
30	PTEND	Pa/3h	pressure tendency	
60	W1	WMO 020004	past weather	
61	WW	WMO 020003	present weather	
62	VV	m	visibility	
63	CH	WMO 020012	type of high clouds	
64	CM	WMO 020012	type of middle clouds	
65	CL	WMO 020012	type of low clouds	
66	NH	m	cloud base height	
67	N_L	WMO 020011	low cloud amount	
93	N_M	WMO 020011	medium cloud amount	
94	N_H	WMO 020011	high cloud amount	
69	C	WMO 500	additional cloud group type	L
70	NS	WMO 2700	additional cloud group amount	
71	SDEPTH	m	snow depth	
72	E	WMO 020062	state of ground	
79	TRTR	h	time period of information	L
80	RR	kg/m**2	precipitation amount	
81	JJ	K	maximum temperature	
87	GCLG	Table 6	general cloud group	
91	N	WMO 020011	total cloud amount	
92	SFALL	m	6h snow fall	
110	PS	Pa	surface (station) pressure	
111	DD	degree	wind direction	
112	FF	m/s	wind force	
118	REFL	0..1	reflectivity	
119	RAWBT	K	brightness temperature	
120	RADIANCE	W/sr/m**3	radiance	
41	U10M	m/s	10m u-component of wind	
42	V10M	m/s	10m v-component of wind	
7	Q	kg/kg	specific humidity	
56	VT	K	virtual temperature	
155	VN_CTH	m	cloud top height	
156	HEIGHT	m	height	L
157	FLEV	m	nominal flight level	L
192	RREFL	Db	radar reflectivity	
193	RADVEL	m/s	radial velocity	
128	PDELAY	m	atmospheric path delay	
162	BENDANG	rad	bending angle	
252	IMPPAR	m	impact parameter	L
248	REFR		refractivity	
245	ZPD		zenith path delay	
246	ZWD		zenith wet delay	
247	SPD		slant path delay	
242	GUST	m/s	wind gust	
251	P	Pa	pressure	L
243	TMIN	K	minimum temperature	

3.5 General and individual cloud group

In addition to the entries in the table below, level significance is encoded in the variable `level_sig` (WMO table 8002, 6 bits)

value	name	units	description
0	CLBP	WMO 020011	bit position for cloud amount
4	LCBP	WMO 020012	bit position for low cloud type
10	MCBP	WMO 020012	bit position for middle cloud type
16	HCBP	WMO 020012	bit position for high type
4	CLOC	WMO 020011	no.bits used for cloud amount
6	LCOC	WMO 020012	no.bits used for low cloud type
6	MCOC	WMO 020012	no.bits used for middle cloud type
6	HCOC	WMO 020012	no.bits used for high type

Table 6: general cloud group bit positions

value	name	units	description
0	CLBP	WMO 020011	bit position for cloud amount
4	CTBP	WMO 020012	bit position for cloud type
10	BSBP	m	bit position for cloud base height
4	CLOC	WMO 020011	no.bits used for cloud amount
6	CTOC	WMO 020012	no.bits used for cloud type
14	BSOC	m	no.bits used for cloud base height

Table 7: individual cloud group bit positions

3.6 Level significance

bit#	name	description
0	SURFACE	surface
1	STANDARD	standard level
2	TROPO	tropopause level
3	MAX	maximum wind level
4	SIGN	significant level

Table 8: level significance for TEMP/PILOT.

For general and individual cloud group level significance is encoded according to WMO table 8002.

3.7 Specification of Instruments and computational methods

value	name	description
17	GRAW	Graw (D)
26	BASORA	Basora (CH)
27	RS_RU_A_MRZ	AVK-MRZ (Russia)
28	RS_RU_MET1	Meteorit Marz2-1 (Russia)
37	RS_80	Vaisala RS 80
49	RS_VIZ_M2	VIZ MARK II (USA)
57	RS_DC_MODEM	M2K2-DC Modem (France)
58	RS_RU_A_BAR	AVK-BAR (Russia)
71	RS_90_DIG12	Vaisala RS 90 Digicora I,II or Marvin
75	RS_RU_ARMA	AVK-MRZ-ARMA (Russia)
79	RS_92_DIG12	Vaisala RS 92 Digicora I,II or Marvin
80	RS_92_DIG3	Vaisala RS 92 Digicora III
81	RS_92_AUTO	Vaisala RS 92 Autosonde
88	RS_RU_V_MRZ	MARL-A or Vektor-M-MRZ (Russia)
89	RS_RU_V_BAR	MARL-A or Vektor-M-BAR (Russia)
99	RS_SA_DAT4G	BAT-4G (South Africa)
255	MISS	missing value

Table 9: radiosonde type (NRARA), for other values see WMO common code table C2

value	name	description
0	NOWIND	no windfinding
2	AUX_OPTIC	automatic with aux. optical direction finding
3	AUX_RANGE	automatic with auxiliary ranging
6	LORANC	automatic cross chain Loran-C
8	SATNAV	automatic satellite navigation
19	NOTSPEC	tracking technique not specified
70	NORMAL	all systems in normal operation
127	MISS	missing value
18	AIR_PHASE	aircraft obs: flight phase from data assimil.

Table 10: tracking technique (NSASA), for other values see WMO common code table C7

value	name	description
1	IR	infrared channel
2	VIS	visible channel
3	WV	water vapour channel
101	IR1	8.7 um Meteosat 8-9, 10.7 um GOES 10-12
201	IR2	9.7 um Meteosat 8-9, 3.9 um GOES 10-12
301	IR3	10.8 um Meteosat 8-9
102	VIS1	0.6 um Meteosat 8-9, 0.65um GOES 10-12
202	VIS3	0.8 um Meteosat 8-9
302	VIS2-HR	0.75um Meteosat 8-9
103	WV1	6.2 um Meteosat 8-9,
203	WV2	7.3 um Meteosat 8-9,
303	WV3	6.8/6.5 um GOES 10,11/12

Table 11: satellite derived wind computation method

value	name	description
0	PRESS	pressure instrument associated with wind measuring equipment
1	OPTTHEO	optical theodolit
2	RADTHEO	radio theodolite
3	RADAR	radar
4	VLFOMEGA	VLF-Omega
6	WINDPROF	wind profiler
5	LORANC	Loran-C
7	SATNAV	satellite navigation
8	RASS	radio acoustic sounding system (RASS)
9	SODAR	SODAR
15	MISS	missing

Table 12: type of measuring equipment used (NA4, 002003)

value	name	description
0	NO	no correction
1	CS_CI	CIMO solar + CIMO infrared corrected
2	CS_IN	CIMO solar + infrared corrected
3	CS	CIMO solar corrected only
4	SO_IN_AUTO	solar + infrared corr., automatic. by rsond. system
5	SO_AUTO	solar corrected automatically by radiosonde system
6	SO_IN_CNTRY	solar + infrared corr. as specified by country
7	SO_CNTRY	solar corrected by country
15	MISS	missing

Table 13: solar and infrared radiation correction (NSR, 002013)

3.8 1DVAR Surface type, cloud flags and processing flags

bit#	name	description
0	SEA	
1	ICE	
2	LAND	
3	HIGHLAND	
4	MISMATCH	

Table 14: surface types consistent with 1d-Var

bit#	name	description
0	CLEAR	
1	IR_CLOUDY	
2	MW_CLEAR	
3	MW_CLOUDY	

Table 15: cloud flag

bit#	name	description
0	DATA	
1	MIN	1dvar minimisation failed
2	SUR	wrong surface type
3	CLD	cloud flag

Table 16: 1dvar processing flag

3.9 Model surface characteristics

bit#	name	description
0	LAND	set if some fraction is covered by land
1	SEA	set if some fraction is covered by sea
2	ICE	set if some fraction is covered by sea-ice
3	NO_ICE	set if some fraction is not covered by sea-ice
4	SNOW	set if some fraction is covered by snow
5	NO_SNOW	set if some fraction is not covered by snow

Table 17: model surface characteristics

Values are missing if corresponding bits (eg. LAND and SEA or ICE and NO_ICE) are set to zero.

3.10 Observation quality information

value	name	description
2	UNS	Unsteady
3	LVR	Level flight, routine observation
4	LVW	Level flight, highest wind encountere
5	ASC	Ascending
6	DES	Descending
7	MIS	Missing

Table 18: aircraft phase, radiances fov, gpsro pcd

value	name	description
0	GOOD	Good
1	BAD	Bad
3	MIS	Missing value

Table 19: aircraft roll angle

3.11 Specification of verification runs

value	name	description
0	FORECAST	forecast
1	FIRSTGUESS	first guess
2	PREL_ANA	preliminary analysis in observation space
3	ANALYSIS	analysis
4	INIT_ANA	initialised analysis
5	LIN_ANA	linear operator on analysis (Y_a)

Table 20: type of verification run

value	name	description
0	HAUPT	main forecast cycle
1	VOR	pre-run
2	ASS	assimilation cycle
3	TEST	test (offline)

Table 21: class of verification run

value	name	description
0	ENS_MEAN	ensemble mean
-1	DETERM	derterministic model run
-2	ENS_SPREAD	ensemble spread
-3	BG_ERROR	3dvar background error
-4	TALAGRAND	Talagrand index
-5	VQC_WEIGHT	variational quality control weight
-6	MEMBER	generic value for ensemble member
-7	ENS_MEAN_OBS	ensemble mean in observation space

Table 22: specification of the verification data

The value `MEMBER` is not actually written to the file. It is merely used to specyfy all ensemble members generically in certain subroutine calls.

`ENS_MEAN_OBS` indicates the ensemble mean taken in observation space, in contrast to `ENS_MEAN` which is derived by application of the observation operator to the ensemble mean in model space.

4 Migration of COSMO VOF format to NetCDF format

4.1 File Header

```
The information in the VOF file header is represented in the NetCDF
file as follows:


1) Verification period:

   'initial date and hour' renamed to 'verification_reference_time'
   (yyyymmddhhmm), now in general 'end of assimilation period'

   'start', 'end' renamed to 'verification_start', 'verification_end'
   (minutes)


2) LM-grid: pole, corners:            'lat_pole', 'lon_pole',
                                      'lat_lower_left','lon_lower_left'
                                      'lat_upper_right','lon_upper_right'

         resolution, domain size:  to be generalised for GME GRID


3) initial time and date :   not used
   QC time step          :   not used
   QC thresholds         :   not used


4) Number of model runs to compare with observations: -> 'n_veri'

   This quantity becomes a dimension (not a global attribute)



5) Further information on the verification runs:

   The following quantities are stored as data (variables of dimension
   'n_veri'), not as global attributes.

   Domain used for verification  :  not used
   Types assigned for model runs :  -> 'run_type' to be redefined:
                                    0 forecast
                                    1 first_guess
                                    2 preliminary_analysis
                                    3 analysis
                                    4 initialised_analysis

   'run_type'
   'initial_date'  (yyyymmddhhmm)
   'forecast_time' (hhhmm)        at verification_reference_time

   'mesh_width'    (float)        degree
   'n_levels'
   'description'                possibly more detailed than 'run_type'
                                'nudging run', etkf analysis', ...
 additionally:

   'ens_member'                 ensemble member number, special
                                meaning for non-positive values:
                                 0 : ensemble mean
                                -1 : deterministic run
                                -2 : ensemble spread
                                -3 : bg_error (3DVAR)

   'exp_id'                     experiment id
```

4.2 Report Header

```
 The information in the VOF file is represented in the NetCDF
 file as follows:

 1) basic report type : not used

 2) station identity  : 'statid'    character (len_010)

 3) longitude         : 'lon'       float     (degree)

 4) latitude          : 'lat'       float     (degree)

 5) observation time  : 'time'      int       observation - verification_reference_time in minut

 6) station altitude  : 'z_station' short     (m)

 7) model orography   : 'z_model'   short     (m)

 8) observation type  : 'obstype'   short      1 SYNOP
                                               2 AIREP
                                               3 SATOB (AMV)
                                               4 DRIBU
                                               5 TEMP
                                               6 PILOT
                                               7 SATEM (ATOVS)
                                               8 PAOB
                                               9 SCATT
                                              10 GPS RO
                                              11 GPS ground based  (was =8 in VOF)

 9) observ. code type : 'codetype'  short     cf. Figure 5.10, revised (see Table 5)

10) station
    characteristics   : split up in different variables,

    bit positions
    in VOF:

    0,1                 'r_state'     byte     merged, passive, rejected,
                                               active

    2-6,9,20,21         'r_flags'     int      bit pattern of flags: station location,
                                               height distance, blacklist, code type
                                               excluded, redundant,
                                               suspicion indicator (--> DATASET)

                        'r_check'     byte     check which caused 'r_state'
                                               (one out of 'r_flags')

    22-29               'phase'       byte     aircraft phase and roll angle
                                               (bit-Feld, mit INFO in BUFR
                                                abgleichen)

    13-19               'instype'     int      station type or satellite instrument

    10                  cancelled              (important station indicator)

                         7) sea grid point
                         8) station correction indicator: adopt

11) report flags on lat/lon/date/time/altitude  bit pattern, align with INFO in BUFR

                        'r_flags'     int      bit pattern of flags: station location,

12) status 0/1/2: active/single_level_aircraft_set_passive/passive

                        'r_state'     byte     merged, passive, rejected,
                                               active

13) threshold quality control (QC) for extrapolated surface pressure

                        'flags'       int      in body:
                                               bit pattern of flags: fg-check etc.

14) model index x       'index_x'     int

15) model index y       'index_y'     int

    model index diamond 'index_d'     int
```

5 Fortran 90 interface

The Fortran 90 modules **mo_t_table**, **mo_fdbk_tables**, **mo_t_netcdf_file**, **mo_fdbk**, **mo_fdbk_io**, and **test_feedback** are intended for common use in the COSMO model, the DWD 3D-Var and the LETKF.

The modules **mo_t_table** and **mo_t_netcdf_file** hold general data type definitions and operations for NetCDF file handling, not specific to the feedback-file layout.

Modules **mo_fdbk_tables** and **mo_fdbk** hold variable definitions (table entries) and routines specific the the feedback file. These modules may be used for low level I/O routines which access the feedback file content directly (variable by variable).

Module **mo_fdbk_io** provides higher level routines to read/write the feedback file and derived types to hold the complete file content in memory. Thus application of these routines puts higher demands on memory usage but is easier to use may be more appropriate if no direct storage into model or assimilation system specific data structures is intended. Module **mo_fdbk_rad** builds up on **mo_fdbk_io** and further faciliates writing of feedback files for radiance data.

The module **mo_fdbk_3dvar** holds routines to write the feedback file. This module is specific for the DWD 3D-Var and is provided only as an example. This example is much more complex than **mo_fdbk_io** or **mo_fdbk_rad** as it involves MPI parallelisation and storage in the 3D-Var specific data structures.

The program **test_feedback** writes an (empty) feedback file and prepares the LaTEX sources (tab....tex) with the table entries used in this document. It may serve as an example program how to write feedback files.

The program **test_feedback_read** reads a feedback file either via the low level routines from **mo_fdbk** or via the higher level routines from **mo_fdbk_io** and does some calculation and printout. It may serve as an example program how to read feedback files.

The program **test_feedback_rad** is an example routine which writes a feedback file for radiance data using modules **mo_fdbk_io** and **mo_fdbk_rad**.

5.1 Tables

5.1.1 Data type definitions (mo_t_table)

Derived data types and operators to maintain tables with the description of the data content (as defined in Section 3) are defined within this module.

Derived type definitions:

t_entry

Derived type to hold a table entry relating the value of a datum with a name (mnemonic), a description, and units (if applicable):

```
integer, parameter :: NLEN = 16  ! length of entry name
integer, parameter :: ULEN = 12  ! length of units field
integer, parameter :: DLEN = 64  ! length of entry description

type t_entry
  integer            :: value       ! numerical value or bit number
  character(len=NLEN) :: name        ! associated name (mnemonic)
  character(len=ULEN) :: units       ! units of the numerical value
  character(len=DLEN) :: description  ! some text
end type t_entry
```

t_table

Derived type to hold a table:

```
type t_table
  type (t_entry) ,pointer :: e(:)    ! list of table entries
  character(len=16)       :: name    ! name of table
  character(len=128)      :: caption ! caption (for LaTEX doc)
  integer                 :: n       ! number of entries
  integer                 :: first   ! smallest value or bit number in table
  integer                 :: last    ! largest  value or bit number in table
  logical                 :: bit     ! used as bit-table
end type t_table
```

The flag `bit` indicates if the values are bit numbers used in a flag table.

Subroutines and functions:

init_table

```
subroutine init_table (table, entries, name, caption, bit, latex)
type (t_table)               ,pointer  :: table      ! table
type (t_entry)   ,intent(in) ,target   :: entries(:) ! table entries
character(len=*) ,intent(in)           :: name       ! name of table
character(len=*) ,intent(in)           :: caption    ! caption (for LaTEX)
logical          ,intent(in)           :: bit        ! used as bit flag
logical          ,intent(in) ,optional :: latex      ! write LaTeX file
```

Subroutine to set the values of a table:

1. allocate the pointer `table`.
2. link the pointer component `e` table with the table entries.
3. derive the components `n` (number of entries), `first`, and `last` (smallest and largest value in the table).
4. set `name`, `caption` and `bit` components with the values of the actual arguments.
5. if `latex` is passed with the value `.true.` write a latex file `tab.`*name*`.tex`. This option was used to prepare the tables within Section 3. In the LaTEX sources the following sequences are replaced:

```
'_'  by  '\_'
'%'  by  '\%'
'#'  by  '\#'
'~'  by  '\hfill '
```

name_value

```
elemental function name_value (table, value) result (name)
type (t_table)      ,intent(in) :: table
integer             ,intent(in) :: value
character(len=NLEN)             :: name
```

determines the name of a table entry from its value or bit number.

value_name

```
elemental function value_name (table, name) result (value)
type (t_table)   ,intent(in) :: table
character(len=*) ,intent(in) :: name
integer                      :: value
```

determines the value or bit number of a table entry from its name. If a table entry cannot be found a value of `INVALID_VALUE = -999` is returned.

position

```
  elemental function position (table, name)
  type (t_table)   ,intent(in) :: table
  character(len=*) ,intent(in) :: name
  integer                      :: position
or
  elemental function position (table, value)
  type (t_table) ,intent(in) :: table
  integer        ,intent(in) :: value
  integer                    :: position
```

determines the position of the entry in the table from its name or value.

5.1.2 Feedback file tables (mo_fdbk_tables)

In this module the tables (variables of derived type `t_table`) are defined. The module has the following public entities:

The tables used in Section 3 of this document:

`type (t_table) ::  status, flags, obstype, codetype, varno, runtype, runclass, satsens, rsondtype, trackteqn, meas_equip, radiation_corr, surftype, flg_1dvar, flg_cld, level_sig, phase, rollangle, retrtype, ensmem`

Constant definitions for the table entry values:

`integer, parameter ::  ST_ACCEPTED, ST_ACTIVE, ..` for the entries of table status.
`integer, parameter ::  FL_OBSTYPE, FL_BLACKLIST, ..` for the entries of table flags.
`...`

init_fdbk_tables (subroutine)

```
  subroutine init_fdbk_tables (latex)
  logical, intent(in), optional :: latex
```

Subroutine to initialise the tables (variables of derived type `t_table`) defined in this module. join table entries and table meta data (name and caption), optionally (if 'latex' is given and true) write LaTeX file with tables for inclusion in the documentation.

Actually subroutine `init_table` defined in module `mo_t_table` is called for each table with suitable parameters.

5.2 NetCDF interface

5.2.1 Data type definitions (mo_t_netcdf_file)

This module defines the derived type **t_netcdf_file** which mirrors some of the meta data of a NetCDF-file. The subroutines **add_dim** and **add_var** add a NetCDF-dimension or a NetCDF-variable to a variable of this type. Subroutine **create_netcdf_file** creates (writes) an empty (dimension and variable definitions) NetCDF file based on the information in the variable. Subroutines open_netcdf_file_read and open_netcdf_file_write open a netcdf file for read or write access, respectively. Subroutines to actually write the data (contents of the variables) are not yet implemented.) Subroutine **close_netcdf_file** closes the NetCDF file. Subroutine **destruct_netcdf_file** finally deallocates pointer components of a variable of type **t_netcdf_file** when it is not used any more.

Derived type definitions:

t_netcdf_file

Derived type to hold some of the meta-data of a NetCDF-file:

```
integer, parameter :: NLEN =  32     ! len  of variables name
integer, parameter :: LLEN =  64     ! len  of longname
integer, parameter :: ULEN =  16     ! len  of units attribute
integer, parameter :: OLEN =   8     ! len  of optionals string
integer, parameter :: ODIM =   4     ! size of optionals array
integer, parameter :: PLEN = 128     ! len  of file path/name
integer, parameter :: MDIM =   4     ! max. number of dimensions

type t_netcdf_file
  character(len=PLEN)          :: path     =  ''        ! file path/name
  integer                      :: status   =  UNDEFINED ! file status
  integer                      :: error    =  NF_NOERR  ! error return value
  integer                      :: ndim     =  0         ! number of dimensions
  integer                      :: nvar     =  0         ! number of variables
  integer                      :: ncid     =  0         ! NetCDF file id
  type (t_netcdf_dim) ,pointer :: dims (:) => NULL()    ! dimensions
  type (t_netcdf_var) ,pointer :: vars (:) => NULL()    ! variables
end type t_netcdf_file
```

The pointer component `dims` holds information on the NetCDF-dimensions:

```
type t_netcdf_dim
  character(len=NLEN)    :: name      = ''      ! name of dimension
  integer                :: len       = -1      ! length of dimension
  logical                :: unlimited = .false. ! unlimited dimension ?
  integer                :: dimid     = -1      ! NetCDF dimension id
  integer                :: pos       =  0      ! index of this entry in array
end type t_netcdf_dim
```

The pointer component `vars` holds information on the NetCDF-variables:

```
type t_netcdf_var
  character(len=NLEN)    :: name       =  ''       ! name of the variable
  character(len=LLEN)    :: longname   =  ''       ! CF convention
  character(len=ULEN)    :: units      =  ''       ! CF convention
  type(t_table) ,pointer :: table      => NULL()   ! table of valid values
  integer                :: invalid    = -huge(1)  ! invalid value for ..
  real                   :: rinvalid   = -huge(1.) ! .. this variable
  integer                :: nvdims     =  0        ! number of dimensions
  type(p_netcdf_dim)     :: p (MDIM)               ! pointer to dimensions
  character(len=OLEN)    :: optionals(ODIM) =  ''  ! mark optional variables
  logical                :: opt_used = .false.     ! indicate used opt.variable
  integer                :: xtype      = -1        ! NetCDF data type
  integer                :: varid      = -1        ! NetCDF variable id
end type t_netcdf_var
```

The array component `p` holds pointers to the NetCDF dimensions of the variable:

```
type p_netcdf_dim
  type(t_netcdf_dim) ,pointer :: dim => NULL()
end type p_netcdf_dim
```

Subroutines:

add_dim

```
subroutine add_dim (file, name, len, pos, unlimited)
type (t_netcdf_file),intent(inout)        :: file      ! NetCDF meta data
character(len=*)    ,intent(in)           :: name      ! name of dimension
integer             ,intent(in)           :: len       ! length of dimension
integer             ,intent(out)          :: pos       ! index in file%dims
logical             ,intent(in) ,optional :: unlimited ! unlimited dimension
```

Adds a dimension to the NetCDF file meta data.

add_var

```
subroutine add_var (file, name, xtype, pos, longname, units, table, opt, invalid, rinvalid)
type (t_netcdf_file),intent(inout)        :: file     ! NetCDF meta data
character(len=*)    ,intent(in)           :: name     ! name of the variable
integer             ,intent(in)           :: xtype    ! NetCDF data type
integer             ,intent(in)           :: pos (:)  ! pointer to dimensions
character(len=*)    ,intent(in)           :: longname ! CF convention
character(len=*)    ,intent(in) ,optional :: units    ! CF convention
type(t_table)       ,pointer    ,optional :: table    ! table to valid values
character(len=*)    ,intent(in) ,optional :: opt      ! optional flags
integer             ,intent(in) ,optional :: invalid  ! fillvalue for ints
integer             ,intent(in) ,optional :: rinvalid ! fillvalue for reals
```

Add a variable to the NetCDF file meta data. The optional parameter `opt` may be passed with a list of mnemonics separated by blanks (for instance `TEMP SYNOP PILOT`) in order to indicate that the respective variables are optional. They will be actually written to the NetCDF file only if one of the mnemonics passed later as the actual argument `opt` of subroutine `create_netcdf_file` matches this string. The optional arguments `invalid` or `rinvalid` may specify a non-default fillvalue to be used for this variable.

create_netcdf_file

```
subroutine create_netcdf_file (file, cmode, opt)
type (t_netcdf_file)       ,intent(inout) :: file  ! NetCDF meta data
integer          ,optional ,intent(in)    :: cmode ! NetCDF creation mode
character(len=*) ,optional ,intent(in)    :: opt   ! optional parameter flag
```

Create a NetCDF file from its meta data provided in variable `file`:

This routine creates a NetCDF file and writes its meta data. It only defines NetCDF-dimensions, NetCDF-variables and some variable attributes (CF conventions: longname, units). Global attributes and variable contents must be written separately. Variables are only defined if the optional parameter flags match the respective flags set by subroutine `add_var`.

open_netcdf_file_read

```
subroutine open_netcdf_file_read (file)
type (t_netcdf_file) ,intent(inout) :: file  ! NetCDF meta data
```

Open the netCDF file for read access.

open_netcdf_file_write

```
subroutine open_netcdf_file_write (file)
type (t_netcdf_file) ,intent(inout) :: file  ! NetCDF meta data
```

Open the netCDF file for read/write access.

close_netcdf_file

```
subroutine close_netcdf_file (file)
type (t_netcdf_file) ,intent(inout) :: file
```

Close a NetCDF file. This routine should be called after the NetCDF file was created by subroutine `create_netcdf_file` and global attributes and variable contents has been written.

destruct_netcdf_file

```
subroutine destruct_netcdf_file (file)
type (t_netcdf_file) ,intent(inout) :: file
```

Clean up the NetCDF file meta data derived type variable: deallocate components. This routine should be called if the content of the variable `file` is not used any more.

5.2.2 Feedback file interface (mo_fdbk)

This module defines the derived type **t_fdbk**. In addition to the information on dimensions and variables (within component **nc** of derived type **t_netcdf_file**) the type holds feedback file specific information (number of data items actually written, global attributes) and meta data on the verification runs (within component **veri**. Subroutine **setup_fdbk** sets up the information within this data type, **create_fdbk** writes a NetCDF file based on the meta-data stored in a variable of this type.

Subroutines **open_fdbk_read** and **open_fdbk_write** open a feedback file for read or write access, respectively.

Subroutine **add_history** adds a history entry to the global attributes. **write_global_attributes** actually writes the global attributes to the file.

Subroutine **add_verification** adds verification meta data and writes it to the file.

Subroutine **read_meta** reads the attributes and verification meta data and **print_fdbk** produces a printout of this meta data.

Subroutine **get_varid** gets the NetCDF varid of a variable. **get_fillvalue** returns the fillvalue used for a specific variable. **write_fdbk_var** actually writes a variable to the file.

get_veri_index gets the index or indices of verification data items. **get_veri** reads and **write_veri** writes verification data.

Subroutine **close_fdbk** closes the NetCDF-file and **cleanup_fdbk** deallocates pointer components of **t_fdbk**.

All the routines are based on those defined in `mo_t_netcdf_file` but extend them by the feedback file specific parts.

Derived type definitions:

t_fdbk

```
type t_fdbk                                 ! specific feedback file attributes
  !--------------------------------
  ! global feedback file attributes
  !--------------------------------
  integer             :: n_hdr         = 0 ! number of header records    used
  integer             :: n_body        = 0 ! number of body   records    used
  integer             :: n_veri        = 0 ! number of verification runs used
  character(len=TLEN) :: title         ='' ! (CF) title
  character(len=5)    :: version       ='' ! (CF) version.subversion
  character(len=ILEN) :: institution   ='' ! (CF) institution
  character(len=HLEN) &                    ! (CF) history
             ,pointer :: history(:) =>NULL()
  character(len=SLEN) :: source        ='' ! (CF) source
  integer             :: refdate       = 0 ! reference date 'yyyymmdd'
  integer             :: reftime       = 0 ! reference time 'hhmm'
  integer             :: start         = 0 ! verification start (minutes)
  integer             :: end           = 0 ! verification end   (minutes)
  real                :: resolution (2)= 0 ! resolution : lat,lon  (degree)
  integer             :: domain     (3)= 0 ! domain size: x,y,z
  real                :: pole       (2)= 0 ! pole coordinates : lat,lon
  real                :: lower_left (2)= 0 ! lower left  corner of domain
  real                :: upper_right(2)= 0 ! upper right corner of domain
  !------------------------------
  ! generic NetCDF data structure
  !------------------------------
  type (t_netcdf_file) :: nc               ! generic NetCDF data structure
  !---------------------------
  ! verification run meta data
  !---------------------------
  type(t_fdbk_meta) ,pointer :: veri(:) =>NULL() ! verification run meta data
end type t_fdbk
```

Stores NetCDF meta data (global attributes) of a feedback file.

t_fdbk_meta

```
type t_fdbk_meta
  character(len=MLEN) :: model          = ' '! model used for verification
  integer             :: run_type       = -1 ! type of model run
  integer             :: run_class      = -1 ! class of model run
  character(len=IDLEN):: initial_date   = ' '! start of verification period
  integer             :: forecast_time  =  0 ! forecast time at verification_ref_time
  real                :: resolution (2) =  0.! model resolution (x,y)
  integer             :: domain_size(3) =  0 ! domain size      (x,y,z)
  character(len=DLEN) :: description    = ' '! detailed description
  integer             :: ens_member     =  0 ! ensemble member number / special values
  integer             :: exp_id         = -1 ! experiment Id
end type t_fdbk_meta
```

Stores meta data concerning the verification data.

Subroutines:

setup_fdbk

```
subroutine setup_fdbk (nc, latex)
type (t_netcdf_file) ,intent(out)          :: nc    ! NetCDF data type
logical              ,intent(in) ,optional :: latex ! write LaTEX tables
```

This subroutine sets up the information on dimensions and variables stored in variable `nc` of derived type `t_netcdf_file`. These information apply to feedback files in general but are not specific to a certain model (COSMO, GME) or report type (TEMP, SYNOP).

create_fdbk

```
subroutine create_fdbk (fb, path, model, version, institution, n_hdr,    &
                        n_body, refdate, reftime, start, end, resolution,&
                        domain, comment, time, runtime,                  &
                        pole, lower_left, upper_right, opt, create       )
!------------------------------------------------------------
! create new feedback file
!   task 1: set up derived type according to actual parameters
!   task 2: create file and write global attributes
!------------------------------------------------------------
type(t_fdbk)                 ,intent(inout) :: fb         ! feedback file data type
character(len=*)             ,intent(in) :: path          ! pathname
character(len=*)             ,intent(in) :: model         ! model string
character(len=*)             ,intent(in) :: version       ! model version
character(len=*)             ,intent(in) :: institution   ! institution string
integer                      ,intent(in) :: n_hdr         ! allocated size of header
integer                      ,intent(in) :: n_body        ! allocated size of body
integer                      ,intent(in) :: refdate       ! reference time yyyymmdd
integer                      ,intent(in) :: reftime       ! reference time hhmm
integer                      ,intent(in) :: start         ! verification start (minutes)
integer                      ,intent(in) :: end           ! verification stop  (minutes)
real                         ,intent(in) :: resolution (2)! model resolution    (degree)
integer                      ,intent(in) :: domain     (3)! domain size          (x,y,z)
character(len=*)             ,intent(in) :: comment       ! comment  (for history)
character(len=*)             ,intent(in) :: time          ! time     (for history)
character(len=*) ,optional ,intent(in) :: runtime       ! run time (for history)
real             ,optional ,intent(in) :: pole       (2)! location of pole    (degree)
real             ,optional ,intent(in) :: lower_left (2)! lower left  (lat,lon degree)
real             ,optional ,intent(in) :: upper_right(2)! upper right (lat,lon degree)
character(len=*) ,optional ,intent(in) :: opt           ! flag optional variables
logical          ,optional ,intent(in) :: create        ! if .false. postpone task 2
```

This subroutine stores information in variable `fb` of derived type `t_fdbk`. Information specific to the feedback file file (global attributes, dimension sizes to allocate) are passed by actual arguments. Finally the file is actually created (umless parameter `create` is set to `.false.`.

open_fdbk_read

```
subroutine open_fdbk_read (fb, path)
type(t_fdbk)     ,intent(inout) :: fb    ! feedback file data type
character(len=*) ,intent(in)    :: path  ! pathname
```

Opens a feedback file for read access.

open_fdbk_write

```
subroutine open_fdbk_write (fb, path)
type(t_fdbk)     ,intent(inout) :: fb    ! feedback file data type
character(len=*) ,intent(in)    :: path  ! pathname
```

Opens a feedback file for write access.

add_history

```
subroutine add_history (fdbk, model, starttime, runtype, runtime, write)
!---------------------------------------
! add history entry to global attributes
!  1) store in derived type
!  2) actually write to file
!---------------------------------------
type (t_fdbk)     ,intent(inout)        :: fdbk
character(len=*)  ,intent(in)           :: model
character(len=*)  ,intent(in)           :: starttime
character(len=*)  ,intent(in)           :: runtype
character(len=*)  ,intent(in) ,optional :: runtime   ! (yyyymmddhhmm)
logical           ,intent(in) ,optional :: write     ! if .false. postpone 2)
```

Adds history entry to the global attributes (hold in the derived type) and writes them to the file (unless parameter `write` is set to `.false.`.

write_global_attributes

```
subroutine write_global_attributes (fb)
type (t_fdbk), intent(inout) :: fb
```

Write the global attributes hold in the derived type to the file.

add_verification

```
subroutine add_verification (fdbk, model, run_type, run_class, initial_date,&
                             fc_time, resolution, domain_size, description, &
                             ens_member, exp_id, id_veri, replace, ierr)
```

Add an verification entry to the feedback file. All meta data entries (variables `veri_*`) are written. Only the actual data (`veri_data`) has to be written in a subsequent step. If `replace` is present and > 0, the entry with the respective entry is replaced. Otherwise the entry is appended. The index actually used is returned in `replace`. By passing `ierr` a program abort due to an error condition may be captured. In this case `ierr` returns a value $/ =$ zero.

read_meta

```
subroutine read_meta (fb)
type (t_fdbk), intent(inout) :: fb
```

Reads global attributes and verification meta data.

print_fdbk

```
subroutine print_fdbk (fdbk)
type (t_fdbk)     ,intent(in) :: fdbk
```

Prints global attributes and verification meta data.

get_varid

```
function get_varid (fb, name) result (varid)
```

Gets NetCDF varid of a variable in the feedback file. Returns -999 if the variable is not present.

get_fillvalue

```
function get_fillvalue (fb, name) result (fillvalue)
type(t_fdbk)     ,intent(in) :: fb        ! feedback file data type
character(len=*) ,intent(in) :: name      ! variable name
real                         :: fillvalue ! fillvalue
```

Return the fillvalue used for a variable specified by its name.

write_fdbk_var

```
type(t_fdbk)      ,intent(in) :: fb     ! feedback file meta data
integer           ,intent(in) :: iv     ! variable index
....              ,intent(in) :: var(:) ! variable to write
integer ,optional ,intent(in) :: fill   ! fillvalue to replace
```

Write a variable to the feedback file. `var` may be of type `integer`, `real(wp)` or `character(len=*)`. If `fill` is passed any element with this value will be replaced by the fillvatue used for the respective variable.

get_veri_index

```
subroutine get_veri_index (idx, nidx, fb, model, run_type, run_class,      &
                           initial_date, forecast_time, ens_member, exp_id)
integer          ,intent(out)           :: idx (:)  ! indices returned
integer          ,intent(out)           :: nidx     ! number of indices
type (t_fdbk)    ,intent(in)            :: fb
character(len=*) ,intent(in)  ,optional :: model
integer          ,intent(in)  ,optional :: run_type
integer          ,intent(in)  ,optional :: run_class
character(len=*) ,intent(in)  ,optional :: initial_date
integer          ,intent(in)  ,optional :: forecast_time
integer          ,intent(in)  ,optional :: ens_member
integer          ,intent(in)  ,optional :: exp_id
```

Returns the indices of verification entries specified by the (optional) arguments.

get_veri

```
function get_veri (fb, index) result (veri)
type (t_fdbk)    ,intent(in) :: fb
integer          ,intent(in) :: index
real                         :: veri(fb% n_body)
```

Actually reads a verification entry.

write_veri

```
type (t_fdbk)    ,intent(in) :: fb
integer          ,intent(in) :: index
real(sp or dp)               :: veri(:)
```

Actually writes a verification entry.

close_fdbk

```
subroutine close_fdbk (fb)
type (t_fdbk) ,intent(inout) :: fb
```

Close a feedback file.

cleanup_fdbk

```
subroutine cleanup_fdbk (fb)
type (t_fdbk) ,intent(inout) :: fb
```

Deallocate components of a variable of derived type `t_fdbk` when not used any more.

Examples how to use this interface are provided in the subsequent Section 5.3.

5.2.3 Feedback file I/O handling (mo_fdbk_io)

This module holds the derived type definition **t_fdbk_data** to hold the content of a feedback file and routines to read/write its content from/to a file. The subroutine **read_fdbk_data** reads the entries of a feedback file and stores them in a variable of type **t_fdbk_data**. Entries may be reorganised by deleting entries in the header and body components of the derived type variable by means of subroutine **pack_fdbk** or **associate_hb**. Subroutines **read_fdbk_veri**, **read_fdbk_var_head**, and **read_fdbk_var_body** may be used to read verification data and optional variables. Subroutine **write_fdbk_file** writes the content of the derived type variable. The content of a variable of type **t_fdbk_data** is deleted by subroutine **destruct**. The example program **test_feedback_read** (cf. section 5.3.2 shows how to use the routines from this module.

Derived type definitions:

t_fdbk_data

Container for feedback file data: header and body entries and verification meta data.

```
type t_fdbk_data
  type(t_fdbk)               :: f                         ! feedback file meta data
  type(t_fdbk_head) ,pointer :: h (:) => NULL()           ! feedback file header
  type(t_fdbk_body) ,pointer :: b (:) => NULL()           ! feedback file body data
  real(wp)          ,pointer :: veri_data(:,:) => NULL() ! verification data
  type(t_fdbk_meta) ,pointer :: veri_meta  (:) => NULL() ! verific.meta data
end type t_fdbk_data
```

t_fdbk_head

Feedback file header content

```
  type t_fdbk_head
    !------------------
    ! mandatory entries
    !------------------
    integer           :: i_body        = ifill
    integer           :: l_body        = ifill
    integer           :: n_level       = ifill ! number of levels in body
    integer           :: obstype       = ifill
    integer           :: codetype      = ifill
    integer           :: center        = ifill ! station processing center
    integer           :: sub_center    = ifill ! processing subcenter
    integer           :: data_category = ifill
    integer           :: sub_category  = ifill
    integer           :: ident         = ifill
    character(len=10) :: statid        = ''
    real(wp)          :: lat           = rfill
    real(wp)          :: lon           = rfill
    integer           :: time          = ifill
    integer           :: time_dbase    = ifill ! data base time
    integer           :: time_nomi     = ifill ! nominal (synoptic) time
    real(wp)          :: sun_zenit     = rfill ! solar zenith angle
    integer           :: mdlsfc        = ifill
    integer           :: z_station     = ifill ! station height
    integer           :: z_modsurf     = ifill ! model surface height
    integer           :: sta_corr      = ifill ! correction indicator
    integer           :: instype       = ifill ! instrument type
    integer           :: r_state       = ifill
    integer           :: r_flags       = ifill
    integer           :: r_check       = ifill
    integer           :: index_x       = ifill
    integer           :: index_y       = ifill
    !----------------------
    ! some optional entries
    !----------------------
    integer          :: dbkz         = -1
    integer          :: record       = -1
    integer          :: subset       = -1
    integer          :: source       = -1
    integer          :: phase        = -1
    integer          :: obs_id       = -1
    integer          :: flg_1dvar    = -1
    integer          :: flg_cld      = -1
    integer          :: index_d      = -1
    integer          :: surftype     = -1
    real(wp)         :: sat_zenit    = -999._wp
    !-------------------------------------
    ! pointer to original position in file
    !-------------------------------------
    integer          :: pos          = 0        ! position in file
    !------------------------
    ! pointer to body entries
    !------------------------
    integer          :: ib           = 0        ! start index in body
    integer          :: nb           = 0        ! number of body entries
  end type t_fdbk_head
```

The default initialisation values correspond with the values the components will take for the case that an entry is not present in the file or that it takes its NetCDF fillvalue. They will be replaced by the NetCDF fillvalues when the file is written.

t_fdbk_body

Feedback file body content.

```
  type t_fdbk_body
    !-------------------------
    ! content of feedback file
    !-------------------------
    real(wp)         :: obs       = rfill
    real(wp)         :: bcor      = rfill
    real(wp)         :: e_o       = rfill
    real(wp)         :: level     = rfill
    real(wp)         :: plevel    = -1._wp ! optional
    integer          :: level_typ = ifill
    integer          :: level_sig = ifill
    integer          :: varno     = ifill
    integer          :: state     = ifill
    integer          :: flags     = ifill
    integer          :: check     = ifill
    integer          :: qual      = ifill
    !----------------
    ! pointer to file
    !----------------
    integer          :: pos       = 0 ! position in file
    !------------------------
    ! pointer to header entry
    !------------------------
    integer          :: ih        = 0 ! index of header entry
  end type t_fdbk_body
```

The default initialisation values correspond with the values the components will take for the case that an entry is not present in the file or that it takes its NetCDF fillvalue. They will be replaced by the NetCDF fillvalues when the file is written.

Subroutines:

read_fdbk_data

Reads entries of a feedback file and stores them in variable `data` of derived type `t_fdbk_data`.

```
subroutine read_fdbk_data (data, file)
type (t_fdbk_data) ,intent(inout) :: data
character(len=*)   ,intent(in)    :: file
```

pack_fdbk

```
subroutine pack_fdbk (fb, mask_h, mask_b)

type(t_fdbk_data)  ,intent(inout) :: fb         ! feedback file data type
logical, optional  ,intent(in)    :: mask_h (:) ! mask for header entries
logical, optional  ,intent(in)    :: mask_b (:) ! mask for body   entries
```

This routine applies some filtering on header and body entries stored in derived type variable `fb`. The header and body entries to keep are specifiel by the optional parameters `mask_h` and `mask_b`, respectively. The disgarded entries are removed from `fb`. The references from header to body entries and vice versa are updated. Subsequent calls to `read_fdbk_var_head`, `read_fdbk_var_body`, and `read_fdbk_veri` only provide the reduced set of data.

associate_hb

```
  subroutine associate_hb (data)
  type (t_fdbk_data) ,intent(inout) :: data
```

Updates references from header to body entries and vice versa.

read_fdbk_var_head

```
subroutine read\_fdbk\_var\_head (fb, name, var, ierr, fill)
type(t_fdbk_data)  ,intent(in)  :: fb      ! feedback file data type
character(len=*)   ,intent(in)  :: name    ! variable name
return-type        ,intent(out) :: var (:) ! variable returned
integer  ,optional ,intent(out) :: ierr    ! error return parameter
integer  ,optional ,intent(in)  :: fill    ! fillvalue to use
```

Read a real or integer variable with name `name` from the header of the feedback file. The file must be read previously and stored in variable `fb`. The error return parameter is zero if the variable was read successfully. Fillvalues in the NetCDF file may be replaced by a different value provided by the optional parameter `fill`. `var` is either of type `integer` or `real(wp)`.

read_fdbk_var_body

```
subroutine read_fdbk_var_body (fb, name, var, ierr, fill)
type(t_fdbk_data)  ,intent(in)  :: fb      ! feedback file data type
character(len=*)   ,intent(in)  :: name    ! variable name
return-type        ,intent(out) :: var (:) ! variable returned
integer  ,optional ,intent(out) :: ierr    ! error return parameter
integer  ,optional ,intent(in)  :: fill    ! fillvalue to use
```

Same as `read_fdbk_var_h`, but reads a variable from the body of the feedback file.

read_fdbk_veri

Read the verification data and store it into `data%veri_data`. Verification data to be read is restricted by the optional parameters or by an explicit provision of the verification indices `ix`. The meta-data corresponding to the verification data read is provided in the component `data%meta_data`. It is a subset of the entries in `data%f%veri` selected according to the optional parameters. Observations to be read are be restricted by a previous thinning and reorganisation of the derived type variable `data` (cf. subroutine `associate_hb`).

```
subroutine read_fdbk_veri (data, fill, ix, model, run_type, run_class,    &
                           initial_date, forecast_time, ens_member, exp_id)

type (t_fdbk_data) ,intent(inout)        :: data
real(wp)           ,intent(in) ,optional :: fill          ! new fillvalue
integer            ,intent(in) ,optional :: ix (:)
character(len=*)   ,intent(in) ,optional :: model
integer            ,intent(in) ,optional :: run_type
integer            ,intent(in) ,optional :: run_class
character(len=*)   ,intent(in) ,optional :: initial_date
integer            ,intent(in) ,optional :: forecast_time
integer            ,intent(in) ,optional :: ens_member
integer            ,intent(in) ,optional :: exp_id
```

write_fdbk_file

Write the content of the derifed type variable `fb` to the file.

```
type(t_fdbk_data) ,intent(inout)        :: fb      ! feedback file data type
character(len=*)  ,intent(in) ,optional :: file    ! file name
character(len=*)  ,intent(in) ,optional :: comment ! history extension
character(len=*)  ,intent(in) ,optional :: opt     ! options: model, obstypes
logical           ,intent(in) ,optional :: close   ! if .false. dont close
```

destruct

deallocate components of derived type `t_fdbk_data`.

```
subroutine destruct_fdbk_data (data)
type (t_fdbk_data) ,intent(inout) :: data
```

5.2.4 Feedback file output for radiances (mo_fdbk_rad)

Simple interface to write radiance feedback files based on functionality in module `mo_fdbk_io` (section 5.2.3). An example for its usage is given by program `test_fdbk_rad` (5.3.3).

Typical usage:

1) set up derived type meta data
   call setup_fdbk ! from module mo_fdbk
   call create_fdbk ! from module mo_fdbk
   call **setup_fdbk_rad** ! from this module

2) fill in remaining header and body content
   fdbk_data% h% .. =
   fdbk_data% b% .. =

3) write header and body content
   call **write_fdbk_rad** ! from this module

4) optionally write verification datat
   call add_verification ! from module mo_fdbk
   call write_veri ! from module mo_fdbk

5) close the file
   call close_fdbk ! from module mo_fdbk
   call cleanup_fdbk ! from module mo_fdbk

setup_fdbk_rad

Allocates the derived type components of `d` for header and body and pre-sets some data from the subroutine parameters.

```
subroutine setup_fdbk_rad (d, n_fov, n_chan, refdate, reftime, &
                           ident, varno, chan, instr, e_o        )
type(t_fdbk_data) ,intent(inout) :: d
integer                          :: n_fov          ! number of FOV
integer                          :: n_chan         ! number of channels / fov
integer                          :: refdate        ! yyyymmdd
integer                          :: reftime        ! hhmm
integer                          :: ident          ! WMO satellite id
integer                          :: varno (n_chan) ! variable number
integer                          :: chan  (n_chan) ! channel number
integer                          :: instr (n_chan) ! WMO instrument number
real                             :: e_o   (n_chan) ! observation error
```

write_fdbk_rad

Writes the content of the derived type variable `d` to a file.

```
subroutine write_fdbk_rad (d)
type(t_fdbk_data) ,intent(inout) :: d
```

5.3 Test facilities and 3D-Var interface

5.3.1 Test program (test_feedback)

test_feedback

This program calls the subroutines of module `mo_fdbk` to create an (currently empty) feedback file and to write the LaTEX tables used in this document. It may serve as an example program how to write feedback files.

5.3.2 Test program (test_feedback_read)

test_feedback_read

This program reads a feedback file and does some calculation and printout. It may serve as an example program how to read feedback files.

Two different interfaces are used, either the high level interface provided by module `mo_fdbk_io`, or the low level interface from module `mo_fdbk`. The name of the file to read and the interface to be used must passed by stdin.

5.3.3 Test program (test_fdbk_rad)

test_fdbk_rad

Example program to write a feedback file for radiances, based on modules `mo_fdbk_io` (section 5.2.3) and `mo_fdbk_rad` (section 5.2.4).

5.3.4 3d-Var interface (mo_fdbk_3dvar)

This module holds the feedback file interface under development to be used in the 3dvar/LETKF. This module is very 3dvar specific and only included as a reference. A similar interface has to be implemented for the COSMO model.

6 Feedback File Example (ncdump)

File created by test_feedback.f90 :

```
netcdf test_feedback {
dimensions:
d_hdr = 10 ;
d_body = 50 ;
d_radar = 1 ;
d_veri = UNLIMITED ; // (3 currently)
d_2 = 2 ;
d_3 = 3 ;
char10 = 10 ;
char12 = 12 ;
char64 = 64 ;
variables:
int i_body(d_hdr) ;
i_body:longname = "index of 1st entry in report body" ;
i_body:_FillValue = -2147483647 ;
short l_body(d_hdr) ;
l_body:longname = "number of entries in report body" ;
l_body:_FillValue = -32767s ;
short n_level(d_hdr) ;
n_level:longname = "number of levels in report" ;
n_level:_FillValue = -32767s ;
short data_category(d_hdr) ;
data_category:longname = "BUFR4 data category" ;
data_category:_FillValue = -32767s ;
short sub_category(d_hdr) ;
sub_category:longname = "BUFR4 data sub-category" ;
sub_category:_FillValue = -32767s ;
short center(d_hdr) ;
center:longname = "station processing center" ;
center:_FillValue = -32767s ;
short sub_center(d_hdr) ;
sub_center:longname = "station processing sub-center" ;
sub_center:_FillValue = -32767s ;
byte obstype(d_hdr) ;
obstype:longname = "observation type" ;
obstype:_FillValue = -127b ;
short codetype(d_hdr) ;
codetype:longname = "code type" ;
codetype:_FillValue = -32767s ;
int ident(d_hdr) ;
ident:longname = "station or satellite id as integer" ;
ident:_FillValue = -2147483647 ;
char statid(d_hdr, char10) ;
statid:longname = "station id as character string" ;
float lat(d_hdr) ;
lat:units = "degree" ;
lat:longname = "latitude" ;
lat:_FillValue = 9.96921e+36f ;
float lon(d_hdr) ;
lon:units = "degree" ;
lon:longname = "longitude" ;
lon:_FillValue = 9.96921e+36f ;
short time(d_hdr) ;
time:units = "min" ;
```

```
time:longname = "observation minus reference time" ;
time:_FillValue = -32767s ;
short time_nomi(d_hdr) ;
time_nomi:units = "min" ;
time_nomi:longname = "nominal (synoptic) minus reference time" ;
time_nomi:_FillValue = -32767s ;
short time_dbase(d_hdr) ;
time_dbase:units = "min" ;
time_dbase:longname = "data base minus reference time" ;
time_dbase:_FillValue = -32767s ;
short z_station(d_hdr) ;
z_station:units = "m" ;
z_station:longname = "station height" ;
z_station:_FillValue = -32767s ;
short z_modsurf(d_hdr) ;
z_modsurf:units = "m" ;
z_modsurf:longname = "model surface height" ;
z_modsurf:_FillValue = -32767s ;
byte r_state(d_hdr) ;
r_state:longname = "status of the report" ;
r_state:_FillValue = -127b ;
int r_flags(d_hdr) ;
r_flags:longname = "report quality check flags" ;
r_flags:_FillValue = -2147483647 ;
byte r_check(d_hdr) ;
r_check:longname = "check which raised the report status flag value" ;
r_check:_FillValue = -127b ;
byte sta_corr(d_hdr) ;
sta_corr:longname = "station correction indicator" ;
sta_corr:_FillValue = -127b ;
short index_x(d_hdr) ;
index_x:longname = "index x of model grid point assigned to report" ;
index_x:_FillValue = -32767s ;
short index_y(d_hdr) ;
index_y:longname = "index y of model grid point assigned to report" ;
index_y:_FillValue = -32767s ;
byte mdlsfc(d_hdr) ;
mdlsfc:longname = "model surface characteristics" ;
mdlsfc:_FillValue = -127b ;
short instype(d_hdr) ;
instype:longname = "station type or satellite instrument type" ;
instype:_FillValue = -32767s ;
float sun_zenit(d_hdr) ;
sun_zenit:longname = "sun zenith angle" ;
sun_zenit:_FillValue = 9.96921e+36f ;
int obs_id(d_hdr) ;
obs_id:longname = "unique observation id" ;
obs_id:_FillValue = -2147483647 ;
byte source(d_hdr) ;
source:longname = "input file number" ;
source:_FillValue = -127b ;
int record(d_hdr) ;
record:longname = "record number in the input file" ;
record:_FillValue = -2147483647 ;
short subset(d_hdr) ;
subset:longname = "subset number in the record" ;
subset:_FillValue = -32767s ;
short dbkz(d_hdr) ;
dbkz:longname = "DWD data base id" ;
```

```
dbkz:_FillValue = -32767s ;
byte index_d(d_hdr) ;
index_d:longname = "model grid diamond index assigned to report" ;
index_d:_FillValue = -127b ;
short varno(d_body) ;
varno:longname = "type of the observed quantity" ;
varno:_FillValue = -32767s ;
float obs(d_body) ;
obs:longname = "bias corrected observation" ;
obs:_FillValue = 9.96921e+36f ;
float bcor(d_body) ;
bcor:longname = "bias correction, corrected minus observed" ;
bcor:_FillValue = 9.96921e+36f ;
float level(d_body) ;
level:longname = "level of observation" ;
level:_FillValue = 9.96921e+36f ;
short level_typ(d_body) ;
level_typ:longname = "type of level information" ;
level_typ:_FillValue = -32767s ;
short level_sig(d_body) ;
level_sig:longname = "level significance" ;
level_sig:_FillValue = -32767s ;
byte state(d_body) ;
state:longname = "status of the observation" ;
state:_FillValue = -127b ;
int flags(d_body) ;
flags:longname = "observation quality check flags" ;
flags:_FillValue = -2147483647 ;
byte check(d_body) ;
check:longname = "check which raised the observation status flag value" ;
check:_FillValue = -127b ;
float e_o(d_body) ;
e_o:longname = "observational error" ;
e_o:_FillValue = 9.96921e+36f ;
short qual(d_body) ;
qual:longname = "observation confidence from data provider" ;
qual:_FillValue = -32767s ;
float veri_data(d_veri, d_body) ;
veri_data:longname = "observation minus model or model error estimate" ;
veri_data:_FillValue = 9.96921e+36f ;
char veri_model(d_veri, char10) ;
veri_model:longname = "model used for verification, i.e. COSMO, GME ..." ;
byte veri_run_type(d_veri) ;
veri_run_type:longname = "type of model run" ;
veri_run_type:_FillValue = -127b ;
byte veri_run_class(d_veri) ;
veri_run_class:longname = "class of model run" ;
veri_run_class:_FillValue = -127b ;
char veri_initial_date(d_veri, char12) ;
veri_initial_date:units = "yyyymmddhhmm" ;
veri_initial_date:longname = "start of verification period" ;
int veri_forecast_time(d_veri) ;
veri_forecast_time:units = "hhmm" ;
veri_forecast_time:longname = "forecast time at verification_ref_time" ;
veri_forecast_time:_FillValue = -2147483647 ;
float veri_resolution(d_veri, d_2) ;
veri_resolution:units = "degree" ;
veri_resolution:longname = "model resolution, seperately for the x- and y-dimension" ;
veri_resolution:_FillValue = 9.96921e+36f ;
```

```
int veri_domain_size(d_veri, d_3) ;
veri_domain_size:longname = "domain size: nx,ny,nz for COSMO; ni,ni,nz for GME" ;
veri_domain_size:_FillValue = -2147483647 ;
char veri_description(d_veri, char64) ;
veri_description:longname = "more detailed description than veri_run_type" ;
int veri_ens_member(d_veri) ;
veri_ens_member:longname = "ensemble member number, special meaning for non-positive values" ;
veri_ens_member:_FillValue = -2147483647 ;
int veri_exp_id(d_veri) ;
veri_exp_id:longname = "experiment Id of the verification run" ;
veri_exp_id:_FillValue = -2147483647 ;

// global attributes:
:history = "201206151323    3DVAR 200701010000 analysis                                    " ;
:title = "3DVAR      Verification Data" ;
:institution = "German Weather Service" ;
:source = "3DVAR      01.00" ;
:file_version_number = " 1.01" ;
:n_hdr = 0 ;
:n_body = 0 ;
:n_radar = 0 ;
:verification_ref_date = 20070101 ;
:verification_ref_time = 0 ;
:verification_start = 0 ;
:verification_end = 0 ;
:resolution = 1.f, 1.f ;
:domain_size = 192, 192, 40 ;
}
```