MDV FORMAT
Interface Control Document (ICD)

Mike Dixon
Research Applications Laboratory (RAL)
National Center for Atmospheric Research (NCAR)
Boulder, Colorado, USA
November 2006

Overview

MDV history

The Meteorological Data Volume (MDV) format for gridded data was developed by the Research Applications Laboratory at NCAR in the early 1990s. At the time a number of gridded data formats were in use at RAL. To simplify the data systems, it was decided to standardize on a single gridded data type for internal use.

No public data standard available at the time was considered suitable in terms of data encapsulation and internal compression. MDV evolved as a data format unique to RAL and NCAR. It is an effective format for gridded data, with good meta-data support and an efficient internal compression capability which allows for selected decompression of a single plane from a single data field.

Data imported from, and exported to, organizations outside RAL is frequently in formats such as NetCDF, GRIB and HDF5. To the extent possible, converters have been developed to allow transformation between MDV and these other data formats.

An XML/binary version of MDV is under development. The header information will be stored as ASCII in an XML file and the field data will be stored as binary information in a secondary file.

Mdv concepts

MDV is a general purpose data file format for storing two- and three-dimensional gridded data.

MDV is a single-time format - each MDV data set contains data for a single time. Time searching and retrieval is handled by a time-based file naming convention. This is described in detail later in this document.

MDV provides capabilities for managing multiple data fields in a single file. For example, one MDV file might contain radar data with fields of reflectivity and radial velocity, or model data with temperature, humidity and wind speed as separate fields.

In the (x,y) dimension, MDV supports a number of projection types, including Lambert Conformal Conic, Stereographic, the simple Latitude-Longitude grid (also known as Simple Cylindrical) and polar coordinates for radar data.

In the vertical dimension, MDV supports a number of vertical coordinate types, including height in km or ft, pressure levels, flight levels for aviation, sigma levels for numerical models and elevation angles for radar data.

Data fields may be represented as 4-byte floating point, 2- and 1-byte scaled integers and 4-byte RGBA image pixels.

Internal compression is supported, with each plane of each field individually compressed, for speed of access.

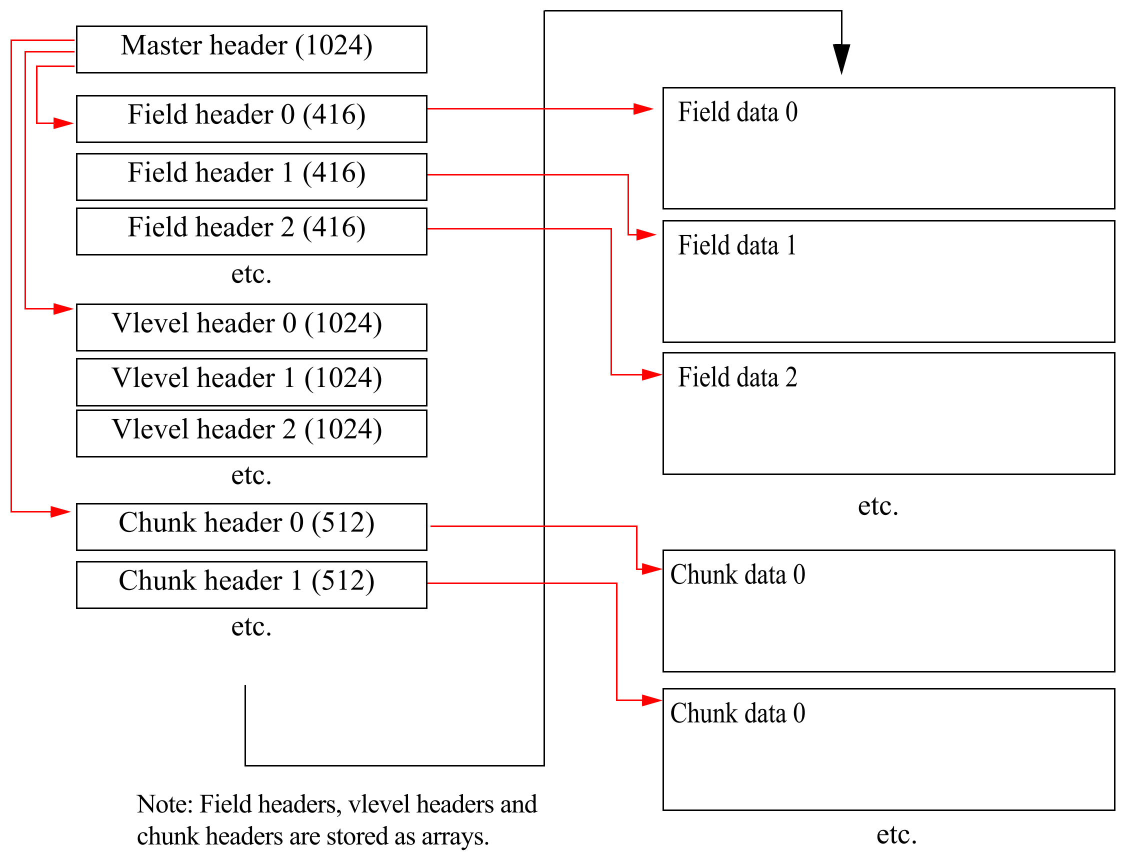
The MDV format is extensible in that it provides space and access capabilities for optional generic “chunk” data defined by the MDV user. Chunk data allows MDV users to attach to the data set additional information that is not suitable for storage in the MDV headers or data fields. As examples, the user may wish to store the elevation angles from which a Cartesian radar grid was derived, or the navigation header for satellite data.

Other features of the MDV file format include:

- a FORTRAN-compatible I/O structure (for non-compressed data);
- date/time stamping for creation, expiration, and forecasting;
- byte swapping to handle cross-platform data representation schemes.

Overview of the MDV file format

The MDV data set structure is organized as follows:



All MDV header information appears at the beginning of the file followed by the field data and any chunk data. In the figure, header lengths are given in bytes.

The master header contains file offsets to the field header array, vlevel header array and (optional) chunk header array.

The field headers contain file offsets to the field data.

Vlevel headers are used to store the details of the third dimension data, e.g. Cartesian plane heights, radar elevation angles, etc.

The (optional) chunk headers contain file offsets to the chunk data.

Binary data types

The header structures in MDV use the following data types:

- 1-byte ASCII characters

- signed 4-byte integers
- IEEE 4-byte floating point values

The data fields in MDV use the following data types:

- unsigned 1-byte integers
- unsigned 2-byte integers
- unsigned 4-byte integers
- IEEE 4-byte floating point values

Byte ordering

For data types with a length of more than 1 byte, the ordering of the data bytes is important in interpreting the data.

All MDV binary data is stored in so-called **big-endian** or **network-byte-order**. This is the native byte ordering for platforms such as SUN, and the opposite of that used by INTEL and AMD processors common to LINUX and Windows systems.

When reading MDV data on a little-endian platform, the multi-byte values will require byte-swapping.

MDV file naming convention

For meteorological data, one of the most important attributes is **time**. Each MDV file contains data for a single time.

MDV files are named according to the time of the data stored in the file. As a general rule, UTC times are used in all MDV data.

A number of times may be applicable:

- **valid time** - the time at which an observation was made. This is also referred to as ‘observation time’;
- **generate time** - the time at which a model was run or a forecast was generated;
- **forecast time** - the time at which a forecast is valid;
- **lead time** - the time difference between the forecast time and generate time for a forecast.

MDV files are named in two ways, as follows.

- By valid time:
  ```
  data_dir/yyyymmdd/hhmmss.mdv
  ```
- By generate time and lead time:
  ```
  data_dir/yyyymmdd/g_hhmmss/f_llllllll.mdv
  ```

Most data sets are stored using **valid time**. The files for a single day are stored in a subdirectory (`yyyymmdd`), named after the year, month and day. The files within the directory are named according to the hour, minute and second (`hhmmss.mdv`).

For forecast-style data, you can choose whether to use the **valid-time** naming convention above, or whether to use the more complicated **generate-time** and **lead-time** convention. The former is simpler, but can lead to over-writing of data sets if forecast results from different generate times have the same valid time. For example, for a model run at 0 UTC and 6 UTC, the 9-hour forecast from the 0 UTC run will be over-written by the 3-hour forecast from the 6 UTC run because they will both be valid at 09 UTC.

To avoid over-writing, use the second naming convention. There are 2 levels of sub-directory. The name of the upper one is based on the year, month and day of the **generate** time (`yyyymmdd`). The lower one is named `g_hhmmss,` based on the hour, minute and second of the **generate** time.The files themselves are named using an 8-digit lead time in seconds (`f_llllllll.mdv`). For example, the 6-hour (21600 second) forecast for the 9 UTC model run on 1 July 2005 will be named:

```
        20050701/g_090000/f_00021600.mdv
```

MDV Headers

C-style syntax

MDV was originally developed for reading and writing in the C and C++ languages. The most comprehensive library support is in C++. Therefore, this document is written with a C++ programmer in mind, and C-style syntax is used because this provides an exact and well-understood way to describe the data format.

However, there is nothing in the MDV format specific to the C language. Readers and writers for MDV have been developed in FORTRAN, Java and IDL.

Portable data types

The following portable C-style data types are defined for use in MDV:

```
typedef unsigned char ui08;    // 1 byte unsigned int
typedef unsigned short ui16;   // 2 byte unsigned int
typedef signed int si32;       // 4 byte signed int
typedef unsigned int ui32;     // 4 byte unsigned int
typedef float fl32;            // 4 byte IEEE float
```

These data types are used in the remainder of the document.

Time representation

Times are represented as 4-byte signed integers.

Time values are computed as the number of seconds since 0 UTC on 1 January 1970. This is so-called UNIX time. (This representation will wrap in the year 2038. Hence the need an XML format.)

Constants

The following constants are defined as part of the MDV format:

```
#define     MDV_CHUNK_INFO_LEN      480
#define     MDV_INFO_LEN            512
#define     MDV_LONG_FIELD_LEN       64
#define     MDV_MAX_PROJ_PARAMS       8
#define     MDV_MAX_VLEVELS         122
#define     MDV_NAME_LEN            128
#define     MDV_SHORT_FIELD_LEN      16
#define     MDV_TRANSFORM_LEN        16
#define     MDV_UNITS_LEN            16
#define     MDV_N_COORD_LABELS        3
#define     MDV_COORD_UNITS_LEN      32
```

Master header

A single master header is placed at the start of the file.

The master header has a length of 1024 bytes.

The master header is defined by the following C-style structure:

```
typedef struct {
  si32 record_len1;
  si32 struct_id;
  si32 revision_number;
  si32 time_gen;
  si32 user_time;
  si32 time_begin;
  si32 time_end;
  si32 time_centroid;
  si32 time_expire;
  si32 num_data_times;
  si32 index_number;
  si32 data_dimension;
  si32 data_collection_type;
  si32 user_data;
  si32 native_vlevel_type;
  si32 vlevel_type;
  si32 vlevel_included;
  si32 grid_orientation;
  si32 data_ordering;
  si32 n_fields;
  si32 max_nx;
  si32 max_ny;
  si32 max_nz;
  si32 n_chunks;
  si32 field_hdr_offset;
  si32 vlevel_hdr_offset;
  si32 chunk_hdr_offset;
  si32 field_grids_differ;
  si32 user_data_si32[8];
  si32 time_written;
  si32 unused_si32[5];
  fl32 user_data_fl32[6];
  fl32 sensor_lon;
  fl32 sensor_lat;
  fl32 sensor_alt;
  fl32 unused_fl32[12];
  char data_set_info[MDV_INFO_LEN];
  char data_set_name[MDV_NAME_LEN];
  char data_set_source[MDV_NAME_LEN];
  si32 record_len2;
} master_header_t;
```

Details of the entries in the master header are as follows:

si32 record_len1

Used by FORTRAN applications, value = 1016.

This is the size of the structure in bytes, not including the record_len integers. (1024 - 2 * 4).

si32 struct_id

Magic cookie. Value = 14142.

si32 revision_number

Revision number of the format, value = 1.

si32 time_gen

Generate time for forecast data sets.

Not used for non-forecast data, in which case value = 0.

si32 user_time

User-application-specified data. Not used by MDV. Normally 0.

si32 time_begin

Start time of data set, if applicable. Often set equal to time_centroid.

si32 time_end

End time of data set, if applicable. Often set equal to time_centroid.

si32 time_centroid

Valid time of the data set. This is the **principal time** used for the data set, to name the files, retrieve data, etc. For observation data, this is normally set to the observation time. For model data, this is set to `time_gen` plus `forecast_delta` (see field header).

si32 time_expire

Not used. Included for backward compatibility.

si32 num_data_times

Not used. Value = 1. Included for backward compatibility.

si32 index_number

Not used. Value = 0. Included for backward compatibility.

si32 data_dimension

2 if all fields contain 2D data, 3 if any fields contain 3D data.

si32 data_collection_type

Used as information on how the data was obtained or generated. This is for information only, it is not used by MDV.

DATA_MEASURED = 0, measured from some instrument.
DATA_EXTRAPOLATED = 1, extrapolated by an algorithm.
DATA_FORECAST = 2, forecast by an algorithm or model
DATA_SYNTHESIS = 3, combination of measured and modelled data
DATA_MIXED = 4, different types of data in data set
DATA_IMAGE = 5, photograph, RGB Image
DATA_GRAPHIC = 6, synthetically Rendered RGB Image

si32 user_data

User-application-specified data. Not used by MDV. Normally 0.

si32 native_vlevel_type

The native vertical data types of the data, before translation into MDV. See vlevel_type.

Used for information only, not used by MDV.

si32 vlevel_type

This is the vertical level type of the data in the fields. VERT_TYPE_VARIABLE applies if the vertical level types differ between fields.

- VERT_TYPE_SURFACE = 1, earth surface field, 2D
- VERT_TYPE_SIGMA_P = 2, sigma pressure levels
- VERT_TYPE_PRESSURE = 3, pressure levels, units = mb
- VERT_TYPE_Z = 4, constant altitude, units = km MSL
- VERT_TYPE_SIGMA_Z = 5, model sigma Z levels
- VERT_TYPE_ETA = 6, model eta levels
- VERT_TYPE_THETA = 7, isentropic surface, units = Kelvin
- VERT_TYPE_MIXED = 8, any hybrid vertical grid, not used much
- VERT_TYPE_ELEV = 9, elevation angles - radar
- VERT_TYPE_COMPOSITE = 10, composite, i.e., max value at any height
- VERT_TYPE_CROSS_SEC = 11, cross sectional view of a set of planes
- VERT_SATELLITE_IMAGE = 12, satellite data
- VERT_FLIGHT_LEVEL = 15, ICAO flight level (100's of ft)
- VERT_EARTH_IMAGE = 16, image, conformal to the surface of the earth
- VERT_TYPE_AZ = 17, azimuth angles - radar RHI
- VERT_TYPE_TOPS = 18, Echo or cloud tops
- VERT_TYPE_ZAGL_FT = 19, constant altitude above ground, units = ft
- VERT_TYPE_VARIABLE = 99, if variable types in fields

si32 vlevel_included

Always 1.

Included for backward compatibility.

si32 grid_orientation

Always 1. The data in the grids is always stored South-to-North and West-to-East.

Included for backward compatibility.

si32 data_ordering

Always 0. The data ordering is always XYZ, meaning that X varies the fastest and Z the slowest in the data arrays.

Included for backward compatibility.

si32 n_fields

Number of data fields.

si32 max_nx

Maximum nx in all fields.

si32 max_ny

Maximum ny in all fields.

si32 max_nz

Maximum nz in all fields.

si32 n_chunks

Number of chunks.

si32 field_hdr_offset

Offset of field header array, from start of file, in bytes.

si32 vlevel_hdr_offset

Offset of vlevel header array, from start of file, in bytes.

si32 chunk_hdr_offset

Offset of chunk header array, from start of file, in bytes.

si32 field_grids_differ

0 if all fields in the file have the same geometry, 1 if not.

si32 user_data_si32[8]

User-application-specified data. Not used by MDV. Normally 0.

si32 time_written

Time at which the file was written.

si32 unused_si32[5]

Unused - for future expansion. All 0.

fl32 user_data_fl32[6]

User-application-specified data. Not used by MDV. Normally 0.

fl32 sensor_lon

Longitude of the sensor, in degrees, if applicable. Otherwise 0.

An example would be the location of a radar.

fl32 sensor_lat

Latitude of the sensor, in degrees, if applicable. Otherwise 0.

fl32 sensor_alt

Altitude of the sensor, in km, if applicable. Otherwise 0.

fl32 unused_fl32[12]

Unused - for future expansion. All 0.

char data_set_info[512]

Data set information in ASCII.

char data_set_name[128]

Data set name, in ASCII.

char data_set_source[128]

Data set source, in ASCII.

si32 record_len2

Used by FORTRAN applications, value = 1016.

Field header

The field headers are stored as an array following the master header.

There are `n_fields` field headers - see master header.

The offset of the start of the array from the start of the file, in bytes, is given by the `field_hdr_offset` in the master header.

Each field header is 416 bytes in length.

The field header is defined by the following C-style structure:

```
typedef struct {
```

```
si32 record_len1;
si32 struct_id;
si32 field_code;
si32 user_time1;
si32 forecast_delta;
si32 user_time2;
si32 user_time3;
si32 forecast_time;
si32 user_time4;
si32 nx;
si32 ny;
si32 nz;
si32 proj_type;
si32 encoding_type;
si32 data_element_nbytes;
si32 field_data_offset;
si32 volume_size;
si32 user_data_si32[10];
si32 compression_type;
si32 transform_type;
si32 scaling_type;
si32 native_vlevel_type;
si32 vlevel_type;
si32 dz_constant;
si32 data_dimension;
si32 zoom_clipped;
si32 zoom_no_overlap;
si32 unused_si32[4];
fl32 proj_origin_lat;
fl32 proj_origin_lon;
fl32 proj_param[MDV_MAX_PROJ_PARAMS];
fl32 vert_reference;
fl32 grid_dx;
fl32 grid_dy;
fl32 grid_dz;
fl32 grid_minx;
fl32 grid_miny;
fl32 grid_minz;
fl32 scale;
fl32 bias;
fl32 bad_data_value;
fl32 missing_data_value;
fl32 proj_rotation;
fl32 user_data_fl32[4];
fl32 min_value;
fl32 max_value;
fl32 min_value_orig_vol;
fl32 max_value_orig_vol;
fl32 unused_fl32;
char field_name_long[MDV_LONG_FIELD_LEN];
char field_name[MDV_SHORT_FIELD_LEN];
char units[MDV_UNITS_LEN];
char transform[MDV_TRANSFORM_LEN];
```

```
      char unused_char[MDV_UNITS_LEN];
      si32 record_len2;
    } field_header_t;
```

Details of the entries in the field header are as follows:

si32 record_len1

Used by FORTRAN applications, value = 408.

This is the size of the structure in bytes, not including the record_len integers. (416 - 2 * 4).

si32 struct_id

Magic cookie. Value = 14143.

si32 field_code

Grib-table field code, if set. Otherwise 0.

Not used by MDV, for information only.

si32 user_time1

User-application-specified data. Not used by MDV. Normally 0.

si32 forecast_delta

Forecast lead time, in seconds. The `forecast_time` in the field header (`time_centroid` in the master header) is the `gen_time` plus the `forecast_delta`.

si32 user_time2

User-application-specified data. Not used by MDV. Normally 0.

si32 user_time3

User-application-specified data. Not used by MDV. Normally 0.

si32 forecast_time

Time at which the forecast is valid. Set equal to the `time_centroid` in the master_header.

si32 user_time4

User-application-specified data. Not used by MDV. Normally 0.

si32 nx

Number of grid cells in X, or W-E dimension.

si32 ny

Number of grid cells in the Y, or S-N dimension.

si32 nz

Number of vertical levels.

si32 proj_type

Projection type. Supported types are:

- PROJ_LATLON = 0, (x,y) in degrees. Simple latitude-longitude grid.
  Also known as the Simple Cylindrical or Platte Carree projection.
- PROJ_LAMBERT_CONF = 3, (x,y) in km. Lambert Conformal Conic projection.
- PROJ_POLAR_STEREO = 5, (x,y) in km. Polar Stereographic projection.
- PROJ_FLAT = 8, Cartesian, (x,y) in km. This is a simple line-of-sight projection used for single radar sites. The formal name is Oblique Lambert Azimuthal projection.
- PROJ_POLAR_RADAR = 9, radar data in native Plan Position Indicator (PPI) coordinates of range, azimuth angle and elevation angle. x is radial range (km), y is azimuth angle (deg), z is elev angle (deg).
- PROJ_OBLIQUE_STEREO = 12, (x,y) in km. Oblique Stereographic projection.
- PROJ_RHI_RADAR = 13, radar data in native Range Height Indicator (RHI) coordinates.
  x is radial range (km), y is elev angle (deg), z is az angle (deg).

si32 encoding_type

This is the encoding type of the field data. MDV supports scaled 1-byte and 2-byte integers, 4-byte floating point and 4-byte RGBA image data.

- ENCODING_INT8 = 1, unsigned 8 bit integer
- ENCODING_INT16 = 2, unsigned 16 bit integer
- ENCODING_FLOAT32 = 5, 32 bit IEEE floating point
- ENCODING_RGBA32 = 7, RGBA image (4 x 8 bits) - same as TIFF RGBA

See scale and bias below for how to convert scaled integers into floats.

si32 data_element_nbytes

1, 2 or 4, depending on the encoding_type above.

si32 field_data_offset

Offset to the start of the data for this field, from the start of the file, in bytes.

si32 volume_size

Length of the field, in bytes.

For non-compressed data, this is nx * ny * nz * data_element_nbytes.

For compressed data, this is the size of the compressed data buffer. See ‘Field data’ section.

si32 user_data_si32[10]

User-application-specified data. Not used by MDV. Normally 0.

si32 compression_type

Compression type for the field data.

Supported types are:

- COMPRESSION_NONE = 0, no compression
- COMPRESSION_ZLIB = 3, Lempel-Ziv
- COMPRESSION_BZIP = 4, bzip2
- COMPRESSION_GZIP = 5, Lempel-Ziv in gzip format

GZIP is the most common compression scheme used now. Other compression options are for backward compatibility.

See the ‘Field data’ section for details on the compression scheme.

si32 transform_type

- DATA_TRANSFORM_NONE = 0
- DATA_TRANSFORM_LOG = 1, natural log

If data is to be transformed and then stored as scaled integers, the floating point data is first transformed using the natural logarithm function, and the scaling is then applied to convert the floating point numbers into scaled integers.

si32 scaling_type

Scaling_type only applies to ui08 and ui16 data. For floating point data, the scaling information is irrelevant.

- SCALING_ROUNDED = 1, scaling factors were computed from the dynamic range of the data and then rounded to reasonable values.
- SCALING_INTEGRAL = 2, scaling factors are integers.
- SCALING_DYNAMIC = 3, scaling factors were computes from the dynamic range of the data.
- SCALING_SPECIFIED = 4, scaling factors were specified by the writer of the file.

The scaling type is included for information only - it is not relevant to readers of the data, since the supplied scaling factors are supplied. (See scale and bias below).

si32 native_vlevel_type

This is the vlevel type of the data from which the data in the file was derived.

For example, the vlevel_type in the file could be VERT_TYPE_COMPOSITE, but the original data may have been a radar volume, in which case the native_vlevel_type could be VERT_TYPE_Z.

See vlevel_type for details on the various types.

si32 vlevel_type

This is the vertical level type of the data in the field, as follows:

- VERT_TYPE_SURFACE = 1, earth surface field, 2D
- VERT_TYPE_SIGMA_P = 2, sigma pressure levels, 3D
- VERT_TYPE_PRESSURE = 3, pressure levels, units = mb, 3D
- VERT_TYPE_Z = 4, constant altitude, units = km MSL, 3D
- VERT_TYPE_SIGMA_Z = 5, model sigma Z levels, 3D
- VERT_TYPE_ETA = 6, model eta levels, 3D
- VERT_TYPE_THETA = 7, isentropic surface, units = Kelvin, 3D
- VERT_TYPE_MIXED = 8, any hybrid vertical grid, not used much, 3D
- VERT_TYPE_ELEV = 9, elevation angles - radar, 3D
- VERT_TYPE_COMPOSITE = 10, composite, i.e., max value at any height, 2D
- VERT_TYPE_CROSS_SEC = 11, cross sectional view of a set of planes, 2D
- VERT_SATELLITE_IMAGE = 12, satellite data, 2D
- VERT_FLIGHT_LEVEL = 15, ICAO flight level (100's of ft), 3D
- VERT_EARTH_IMAGE = 16, image, conformal to the surface of the earth, 2D
- VERT_TYPE_AZ = 17, azimuth angles - radar RHI, 2D
- VERT_TYPE_TOPS = 18, Echo or cloud tops, 2D
- VERT_TYPE_ZAGL_FT = 19, constant altitude above ground, units = ft , 2D

si32 dz_constant

1 if the difference between successive vlevels is constant, 0 if not.

si32 data_dimension

2 for 2D data, 3 for 3D data.

si32 zoom_clipped

Not applicable to data in a file.

Set to 1, on read, if the data returned is clipped, i.e. does not cover the full domain in the file. Otherwise 0.

si32 zoom_no_overlap

Not applicable to data in a file.

Set to 1, on read, if there is no spatial overlap between the data in the file and the area requested. Otherwise 0.

si32 unused_si32[4]

Unused, for future expansion. All 0.

fl32 proj_origin_lat

Latitude of the projection origin, in degrees.

In other words, this is the latitude of the (x = 0, y = 0) point in the projection.

Applicable to all projections except LATLON.

fl32 proj_origin_lon

Longitude of the projection origin, in degrees.

In other words, this is the longitude of the (x = 0, y = 0) point in the projection.

Applicable to all projections except LATLON.

fl32 proj_param[8]

Projection parameters for the various projections.

- PROJ_FLAT: not used.
- PROJ_LATLON: not used
- PROJ_LAMBERT_CONF: proj_param[0] = lat1, proj_param[1] = lat2.
- PROJ_POLAR_STEREO: proj_param[0] = tangent_lon, proj_param[1] = pole_type, 0 for North pole, 1 for South pole.
- PROJ_OBLIQUE_STEREO: proj_param[0] = tangent_lat, proj_param[1] = tangent_lon.

fl32 vert_reference

Not used, value = 0. Included for backward compatibility.

fl32 grid_dx

Resolution of the grid, in the X or W-E direction.

The units are km for all projections except LATLON, for which the units are degrees.

fl32 grid_dy

Resolution of the grid, in the Y or S-N direction.

The units are km for all projections except LATLON, for which the units are degrees.

fl32 grid_dz

Resolution of the grid in the vertical dimension.

This only makes sense if dz_constant is 1. Included for backward compatibility. Use vlevels instead.

fl32 grid_minx

The X, or W-E, coordinate of the **center** of the grid cell at the SW corner of the grid.

For the LATLON projection, this is the longitude, in degrees, of the **center** of the grid cell at the SW corner of the grid.

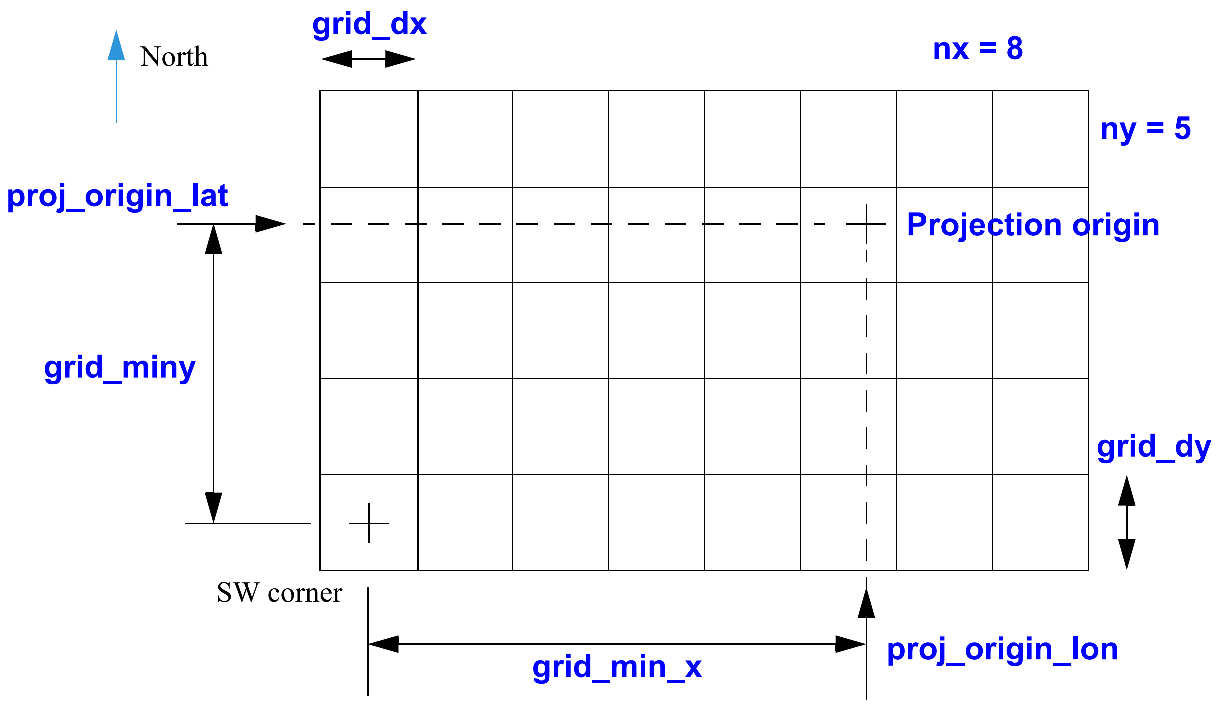
For all other projections, this is the X coordinate, in km, of the **center** of the grid cell at the SW corner of the grid, relative to the projection origin.

fl32 grid_miny

The Y, or S-N, coordinate of the **center** of the grid cell at the SW corner of the grid.

For the LATLON projection, this is the latitude, in degrees, of the **center** of the grid cell at the SW corner of the grid.

For all other projections, it is the Y coordinate, in km, of the **center** of the grid cell at the SW corner of the grid, relative to the projection origin.



Grid and projection details

fl32 grid_minz

The value of the lowest vlevel.

Included for backward compatibility - use vlevels instead.

fl32 scale

The scale value for computing floating point values from scaled integers. See bias.

fl32 bias

The bias value for computing floating point values from scaled integers.

To compute floating point values from scaled integers:

```
float = (scaled_integer * scale) + bias
```

fl32 bad_data_value

Value to identify a bad data value.

For fl32 data, the comparison can be made directly:

```
fl32 value = x;
if (value == bad_data_value) {
  // we have bad data
}
```

For scaled integers, the comparison is made **BEFORE** scaling is applied. We need to cast the value to a floating point value and then perform the comparison:

```
ui16 value = x;
if ((fl32) value == bad_data_value) {
  // we have bad data
}
```

fl32 missing_data_value

Value to flag missing data.

For fl32 data, the comparison can be made directly:

```
fl32 value = x;
if (value == missing_data_value) {
  // we have missing data
}
```

For scaled integers, the comparison is made **BEFORE** scaling is applied. We need to cast the value to a floating point value and then perform the comparison:

```
ui16 value = x;
if ((fl32) value == missing_data_value) {
  // we have missing data
}
```

fl32 proj_rotation

Applies to PROJ_FLAT projection only. This is the rotation of the grid relative to True North, in degrees.

Can be used to align FLAT grids with Magnetic North.

fl32 user_data_fl32[4]

User-application-specified data. Not used by MDV. Normally 0.

fl32 min_value

Minimum value in the field.

fl32 max_value

Maximum value in the field.

fl32 min_value_orig_vol

Not applicable to data in the file. Applies to read operations.

Minimum value in the data field before reading from the file. If a subset of the data is read and returned to the client, then `min_value` will apply to the data returned, and `min_value_orig_vol` will apply to the data in the file.

fl32 max_value_orig_vol

Not applicable to data in the file. Applies to read operations.

Maximum value in the data field before reading from the file. If a subset of the data is read and returned to the client, then `max_value` will apply to the data returned, and `max_value_orig_vol` will apply to all of the data in the file.

fl32 unused_fl32

Unused, for future expansion. All 0.

char field_name_long[64]

Long field name, in ASCII.

char field_name[16]

Short field name, in ASCII.

char units[16]

Field units, in ASCII.

char transform[16]

Field transform, in ASCII. For example, “dB” for radar reflectivity. Normally “none”.

char unused_char[16]

Unused, for future expansion. All 0.

si32 record_len2

Used by FORTRAN applications, value = 408.

Vlevel header

The vertical level headers are stored as an array following the field headers.

There is one vlevel header per field, so the number of vlevel headers is equal to the number of fields - see `n_fields` in the master header.

The offset of the start of the array from the start of the file, in bytes, is given by the `vlevel_hdr_offset` in the master header.

Each vlevel header is 1024 bytes in length.

The vlevel header is defined by the following C-style structure:

```
typedef struct {
  si32 record_len1;
  si32 struct_id;
  si32 type[MDV_MAX_VLEVELS];
  si32 unused_si32[4];
  fl32 level[MDV_MAX_VLEVELS];
  fl32 unused_fl32[5];
  si32 record_len2;
} vlevel_header_t;
```

Details of the entries in the vlevel header are as follows:

si32 record_len1

Used by FORTRAN applications, value = 1016.

This is the size of the structure in bytes, not including the record_len integers. (1024 - 2 * 4)

si32 struct_id

Magic cookie. Value = 14144.

si32 type[122]

Array of vertical level types. Refer to `vlevel_type` in the field header.

Only the first nz values will be set, the rest can be ignored.

For most data sets, the type will be constant and set equal to the vlevel_type in the field header.

Occasionally, a data set will be of the VERT_TYPE_MIXED type and the vertical level types will vary with height. This is very rare.

si32 unused_si32[4]

Unused, for future expansion. All 0.

fl32 level[122]

Array of vertical level values.

Only the first nz values will be set, the rest can be ignored.

fl32 unused_fl32[5]

Unused, for future expansion. All 0.

si32 record_len2

Used by FORTRAN applications, value = 1016.

Chunk header

The chunk headers, if they exist, are stored as an array following the vlevel headers.

The number of chunks is specified by n_chunks in the master header.

The offset of the start of the array from the start of the file is given by the chunk_hdr_offset in the master header.

Chunks contain data the format of which is unknown to MDV. The reader and writer must have agreement on how for encode/decode the chunk data.

Each chunk header is 512 bytes in length.

The chunk header is defined by the following C-style structure:

```
typedef struct {
  si32 record_len1;
  si32 struct_id;
  si32 chunk_id;
  si32 chunk_data_offset;
  si32 size;
  si32 unused_si32[2];
  char info[MDV_CHUNK_INFO_LEN];
  si32 record_len2;
} chunk_header_t;
```

Details of the entries in the chunk header are as follows:

si32 record_len1

Used by FORTRAN applications, value = 504.

This is the size of the structure in bytes, not including the record_len integers. (512 - 2 * 4)

si32 struct_id

Magic cookie. Value = 14145.

si32 chunk_id

This is a user-application-specified ID for the chunk. The MDV format does not specify the ID convention. The reader and writer must agree on a convention for the IDs.

si32 chunk_data_offset

Offset of the chunk data, from the start of the file, in bytes.

si32 size

Length of the chunk data, in bytes.

si32 unused_si32[2]

Unused, for future expansion. All 0.

char info[480]

Chunk info, in ASCII.

si32 record_len2

Used by FORTRAN applications, value = 504.

MDV Field Data

Field data location and size

The field data location is found using the `field_data_offset` entry in the field header. This specifies the offset of the field data, in bytes, from the start of the file.

Also in the field header is the `volume_size` entry, which specifies the length of the field data in bytes.

Field data types

MDV supports 4 data types for field data:

- ui08 - unsigned 1-byte scaled integers
- ui16 - unsigned 2-byte scaled integers
- fl32 - IEEE 4-byte floating point
- ui32 - unsigned 4-byte integers for RGBA image data.

See the field header section on how to convert the scaled integers to floating point values.

Note that the fl32 floating point values and ui32 RGBA values are not scaled - they can be used as is.

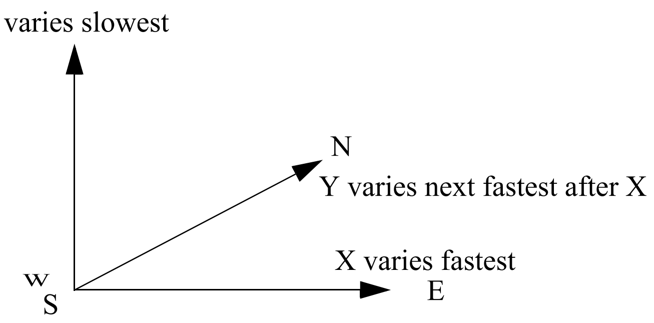
Byte order

The 2-byte and 4-byte data is stored in big-endian byte ordering.

On a small-endian machine, the bytes will need to be swapped.

Storage order

The field data is stored in an array, packed in X-Y-Z ordering, meaning that X varies the fastest, Y next and Z slowest.



Data packing order

The first byte in the array is at the SW corner of the grid and at the lowest vertical level.

To start unpacking the data, begin at the SW corner of the grid and at the lowest vertical level.

Move first in the X dimension from West to East in the first row along the South edge of the grid, for the lowest vertical level, incrementing X as you go.

Once one row in X is completed, increment Y and go back to the West edge.

Then, repeat the process from West to East, this time in the second row.

Continue until the first vertical level is unpacked.

Then, the process repeats for each vlevel in turn, from the bottom to the top.

Data compression

MDV data may be stored internally in uncompressed or compressed form.

For exporting the data, non-compressed data is obviously the easiest to deal with.

If compression is used for file size or bandwidth considerations, GZIP compression is recommended.

Uncompressed data

For uncompressed data, the field is stored in a contiguous array.

The length of the array - `volume_size` in the field header - is equal to

```
nx * ny * nz * data_element_nbytes.
```

Unpack the array as specified in the ‘Storage Order’ section above.

Compressed data buffer containing multiple vlevels

Compressed data is stored in a buffer, the length of which is given by `volume_size` in the field header.

The data for each vertical level is compressed individually, to allow for efficient retrieval of a single level.

The compressed data for the vlevels is concatenated into a single buffer.

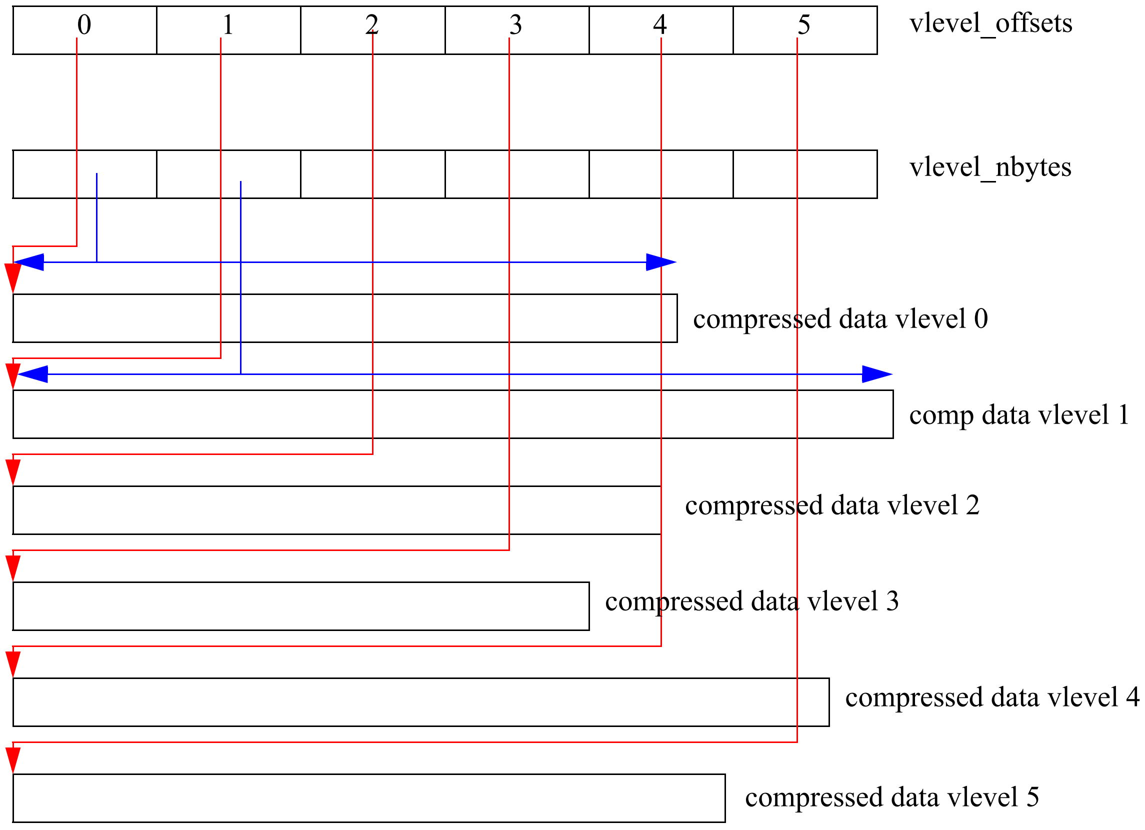
At the start of the compressed buffer are two arrays:

```
ui32 vlevel_offsets[nz]
ui32 vlevel_nbytes[nz]
```

These arrays allow you to find the compressed data for the vlevel you need.

The `vlevel_offsets` array contains the offsets, in bytes, of the compressed data for each vlevel, from the start of the field buffer (*not the file!*).

The `vlevel_nbytes` array contains the size of the compressed data for each individual vlevel.



Compressed buffer layout

Compressed vlevel buffer

Each compressed vlevel buffer has a 24-byte header at the start, and then the compressed data.

The header is as follows:

```
typedef struct {
  ui32 magic_cookie;
  ui32 nbytes_uncompressed;
  ui32 nbytes_compressed; /* including this header */
  ui32 nbytes_coded; /* nbytes_compressed - sizeof(hdr) */
  ui32 spare[2];
} compress_buf_hdr_t;
```

ui32 magic_cookie

The `magic_cookie` identifies the scheme used for compression.

An important point here is that sometimes a compression scheme may fail - it may produce a buffer which is larger than the uncompressed data. In that case, the magic cookie will be set accordingly and the vlevel buffer will contain the original data in uncompressed form.

The magic cookies are as follows. They are specified in hexa-decimal.

TA_NOT_COMPRESSED = 0x2f2f2f2f

The data is not compressed. Use as is it.

GZIP_COMPRESSED = 0xf7f7f7f7

The data was compressed using GZIP.

Uncompressed appropriately.

GZIP_NOT_COMPRESSED = 0xf8f8f8f8

GZIP compression was tried, but it failed.

The data is not compressed. Use as it is.

BZIP_COMPRESSED = 0xf3f3f3f3

The data was compressed with BZIP2 version 0.9.0c.

Uncompress appropriately.

BZIP_NOT_COMPRESSED = 0xf4f4f4f4

BZIP2 compression was tried, but it failed.

The data is not compressed. Use it as it is.

ZLIB_COMPRESSED = 0xf5f5f5f5

The data was compressed with ZLIB compression.

This is the same as GZIP but without the GZIP header structure.

Uncompress appropriately.

ZLIB_NOT_COMPRESSED = 0xf6f6f6f6

ZLIB compression was tried, but it failed.

Data is not compressed, us it as it is.

ui32 nbytes_uncompressed

Number of bytes uncompressed.

ui32 nbytes_compressed

Number of bytes compressed, including the 24-byte header.

ui32 nbytes_coded

Number of bytes compressed, not including the 24-byte header.

ui32 spare[2]

For later use.