The INTREPID DDF format (R08)

Top

This appendix contains information about the structure of the DDF (Data Description Files) used for importing ASCII columns data.

See "Importing ASCII Columns data" in Importing to INTREPID datasets (T05) for instructions for importing ASCII columns data.

DDF are free format ASCII (text) files with elements delimited by spaces. Examples of these files can be found in the directory ***install_path*/examples/DDF** (where ***install_path*** is the location of your INTREPID installation). The translation of this to a protobuf syntax , and therefore an open, published language syntax, occurs with V5.0. We document this change in this manaula as well, as it also includes previously undocumented features, to support the more exotic fields and formats available within INTREPID

Some sample DDF files:

DDF example 1

This DDF

- Creates a line dataset,
- Uses column numbers to delimit fields (only required when the fields are not properly delimited by spaces or some other delimiter),
- Has multiple line records in the input file (every record has the same number of lines),
- Specifies a field values lookup file for one of its fields,
- Has a constant value field.

```
TYPE(LINE)
SURVEY=557              INTEGER*2
#
LINE_NO     1-4         INTEGER*2
FID         6-12        INTEGER*4 MISSING(0)
RECOVERY    14-15       CHARACTER*2
EAST        16-24       REAL*8 MISSING(0) PROJ(TMAMG54,AGD66)IsX
NORTH       27-35       REAL*8 MISSING(0) PROJ(TMAMG54,AGD66)IsY
ZONE        36-36       INTEGER*2 LUT(Zones)
MAGNETIC    37-48       REAL*4
ALTIMETER   50-58       REAL*4
NEXTINPUTRECORD
#
TOTAL_CNT   1-9         REAL*4
POTASSIUM   11-19       REAL*4
URANIUM     21-29       REAL*4
THORIUM     31-39       REAL*4
RESIDUAL    41-49       REAL*4
#
GROUP BY LINE_NO IGNORE(0)
```

See "Fields associated with lookup tables" in INTREPID database, file and data structures (R05) for details of the **zones.leg** field values lookup files 'case study'.

DDF Example 2

This DDF

- Creates a line dataset,
- Uses commas to delimit fields,
- Defines two types of record, one to contain new values for 'group by' fields and one to contain the data belonging to this group,
- Specifies that all values for the field **PRESSURE** be multiplied by 1.00523

```
TYPE(LINE)
DELIMITER(",")
#
RECORD(HEADER)
LOOP  CHARACTER*4
LOOPID     REAL*8
GRAVIOMETERCHARACTER*4
UST   REAL*4
OBSERVER   CHARACTER*30
#
RECORD(DATA)
STATIONNUMBER    INTEGER*4
YEAR  INTEGER*2
MONTH      INTEGER*2
DAY   INTEGER*2
TIME  REAL*8
DIALREADINGREAL*8
DVM   REAL*8
DELTAH     REAL*8
PRESSURE   REAL*8 MULTIPLIER(1.00523)
TEMPERATURE      REAL*8
COMMENTS   CHARACTER*30
```

DDF Example 3

This abbreviated DDF

- Shows a **fiducial** and **raw_mag** field sampled ten times in a single import record using the **REPEAT()** notation, whereas the other fields are sampled only once.
- Shows an Exploranium GR820 data block specification,
- Uses column numbers to delimit fields.

```
TYPE(LINE)
line_number      9-13  INTEGER*4
flight_date      14-20 CHARACTER*6
time_hh_dddd     22-32 CHARACTER*10
fiducial   36-42 INTEGER*4
fiducial   135-141INTEGER*4 REPEAT(1)
fiducial   234-240INTEGER*4 REPEAT(2)
...
fiducial   927-933INTEGER*4 REPEAT(9)
raw_mag          43-52 REAL*4
raw_mag          142-151REAL*4    REPEAT(1)
raw_mag          241-250REAL*4    REPEAT(2)
...
raw_mag          934-943REAL*4    REPEAT(9)
baro_alt   1032-1038  REAL*4
temp_air_deg_c   1053-1059  REAL*4
total_count      1103-1111  REAL*4
gps_lat    1197-1205  REAL*8
gps_long         1207-1216  REAL*8
S          1259-1811  GR820
GROUP BY line_number
```

DDF Example 4

This DDF example enables INTREPID to import and calculate a date from three components in the data file. It stores the date data in the **Date_Type** compound data type. The coponents in the import file are Year, Julian day and Seconds since midnight. For example, the data **2006,230,32400** is 9:00 am, 18 August 2006.

```
TYPE(LINE)
Line INTEGER*2 ALIAS(LineNumber)
Flight INTEGER*2 ALIAS(FlightNumber)
date[Year] Date_Type
date[Jday] Date_Type
date[midnight_seconds] Date_Type
Latitude REAL*8 PROJ(GEODETIC,NAD83) UNITS("degrees")
Longitude REAL*8 PROJ(GEODETIC,NAD83) UNITS("degrees")
```

You can write a DDF structured like this for dates in the format *yyyy,mm,dd* (for example 1995,03,21).

```
date[Year] Date_Type
date[Month] Date_Type
date[Day] Date_Type
```

For more information about the **Date_Type** data type, see "Compound data types" in INTREPID database, file and data structures (R05).

DDF file format

DDFs contain statements that specify properties of the dataset or current record as a whole, and statements that specify dataset fields.

The DDF language supports comment lines. They must have # as their first character.

Case sensitivity

DDF keywords many be all in upper case or all in lower case, but not a mixture of upper and lower. There is one exception to this rule: **IsX** and **IsY** may have mixed case as shown.

Field names may have mixed case as desired.

Dataset TYPE( )

The first line of the DDF may contain a **TYPE** statement, specifying the dataset type for the data. You can import ASCII column data into point, line or polygon datasets.

The line consists of the word **TYPE** followed by the dataset type in parentheses (**LINE**, **POINT** or **POLYGON**).

For example, **TYPE(LINE)**

If you omit the **LINE** statement, INTREPID will assume that the dataset is **point**.

Field DELIMITER(" ")

The second line of the DDF specifies the field delimiter for whole input dataset. The line consists of the word **DELIMITER** followed by the character in **" "** then in parentheses. To specify the tab character, use **\t**.

Examples:

DELIMITER(",") specifies a comma delimiter,

DELIMITER("\t") specifies a tab delimiter.

If you omit the **DELIMITER** statement, INTREPID will assume that the delimiter is the space character (ASCII 32_{10}). See 'Column Numbers' in Section **Field definition lines** for further information about delimiting fields.

DDF example 1 and DDF Example 3 show space delimited DDFs with column number specifications. DDF Example 2 shows a DDF with comma delimiters.

DDFs specifying GR820 data blocks must be space delimited with column numbers specified for all fields.

RECORD(HEADER), RECORD(DATA)

If your dataset is grouped (e.g., a line dataset or a gravity points dataset, you can import data which has

- Header records containing new group information, each followed by
- A number of Data records containing the other data for the records in the group.

INTREPID can recognise this structure, and

- Create a new data point when it encounters a Data record,
- Create a new group when it encounters a Header record and
- Assign the 'group by' data from the most recent Header record to each new data point.

Define the Header record using the statement **RECORD(HEADER)** followed by the field specifications for the Header record.

Define the Data record using the statement **RECORD(DATA)** followed by the field specifications for the Data record.

Your Header record must contain at least one character field in a different position from any character field in the Data record. INTREPID recognises a the record type by matching the data types it finds with the field specifications.

If you omit the **RECORD()** statements, INTREPID will assume all data for each data point is in the corresponding import record.

DDF Example 2 shows the Header record / Data record DDF. The sample data shown below can be imported using this DDF.

```
LOOP,89.01,G20,+11,HReith
96999404,1996,03,29,10:43:54,3017.372,+0.000,219,946.47,21.0
96910804,1996,03,29,11:17:47,3015.922,+0.000,309,945.54,22.0,AU017
96910804,1996,03,29,12:50:56,3015.943,+0.000,308,944.10,25.5,sample_at_12:52
96999404,1996,03,29,13:06:41,3017.422,+0.000,216,944.48,21.0
96999404,1996,03,29,13:32:38,3017.421,+0.000,215,943.99,21.0
LOOP,89.02,G101,+11,HReith
96999404,1996,03,29,10:44:32,3043.608,+0.000,222,946.45,21.0
96999404,1996,03,29,13:35:27,3043.648,+0.000,220,943.93,21.0
LOOP,89.03,G132,+11,HReith
96910804,1996,03,29,11:25:06,3100.804,+0.000,303,945.43,22.5
96999404,1996,03,29,11:42:10,3102.225,+0.000,213,945.88,21.0
96910804,1996,03,29,11:57:04,3100.819,+0.000,305,945.10,24.0,sample_at_11:58
96999404,1996,03,29,13:10:12,3102.279,+0.000,213,944.41,21.0,sample_at_13:12
```

This keyword is associated with the use of **RECORD(HEADER)** and **RECORD(DATA)** in DDF files.

Please contact our technical support service for information about this feature.

Field definition lines

DDF files contain one definition line for each field. Field definition lines have several elements separated by one or more spaces.

Each field definition line must contain

- Field name
- Data type

In addition, each line may contain

- Column numbers
- Format for missing values to be assigned ***null***
- X or Y direction identifiers (The file must have one of each)
- Datum and projection names for the X and Y direction fields
- Assignment of a constant value
- Reference to a field values lookup file

Field Name Must be the first word in the line. Field names may not contain spaces. Join words using '_' .

Format tip For components of compound data types (see "Data Types in INTREPID datasets" in INTREPID database, file and data structures (R05)). This shows which component of the field you are importing in that position. Tips appear in square brackets immediately after the field name (not separatd by a space). Example:

```
obs[Strike] Structural    UNITS("degrees")
obs[Dip]    Structural    UNITS("degrees")
obs[Geo]    Structural    Comment(" unit where obs made")
```

Column Numbers refer to the position of the field in the input data record. The Column Number element consists of two whole numbers joined by '–' representing the first and last column of the field. See "Importing ASCII Columns data" in Importing to INTREPID datasets (T05) for a detailed explanation of this.

You only require Column Numbers if the field delimiter is space and

- Some input records have missing fields (i.e., spaces instead of data) or
- You wish to skip some non-space characters in the input file.

You may wish to use them in any case, to assist you with preventing errors and interpreting the data visually before import. See Field `DELIMITER(" ")` for further information.

DDFs specifying GR820 data blocks must be space delimited with column numbers specified for all fields.

Data Type indicates the type and precision or length of the field. Type and length or precision are separated by a '*****'. Data types include **`INTEGER, REAL, BYTE, CHARACTER`**. **`INTEGER`** fields can be 2 or 4 bytes in length, **`REAL`** fields can be 4 or 8 bytes, **`CHARACTER`** fields can be any length.

See "Data Types in INTREPID datasets" in INTREPID database, file and data structures (R05) for more detals about data types in INTREPID.

Available data types

Data type	Notation
Integer (2 byte)	**INTEGER*2**
Integer (4 byte)	**INTEGER*4**
Real (4 byte)	**REAL*4**
Real (8 byte)	**REAL*8**
Byte	**BYTE**
Logical	**LOGICAL**
Character	(e.g.) **CHARACTER*15**
Data block for Exploranium GR820 radiometrics instrument	**GR820**
Date	**Date_type**
Complex number	**Complex**
Vector	**Vector**
Component	**Component**
Gradient	**Gradient**
Tensor	**Tensor**
Structural geology observation	**Structural**
Observed data	**Observed**

Logical fields take the values:

- **False** on importing the number 0 or the string "**NO**" in upper, lower or mixed case.
- **True** on importing the number 1 or the string "**YES**" in upper, lower or mixed case.

Missing Value Format If you have missing values in your data that are represented by a consistent string or number such as '*********' or –999, you can instruct INTREPID to assign ***null*** to fields with the missing value.

To specify missing values, add the **MISSING** element to the field definition statement. This consists of the word **MISSING** followed by the missing value string or value in parentheses. There must be no space between **MISSING** and the parentheses.

Examples:

```
URANIUM   20-30   REAL*4   MISSING("*****")
EAST      15-23   REAL*8   MISSING(-999)
```

X and Y coordinate identifiers So that INTREPID can locate the data on the Earth's surface, every file must contain one field with coordinate data for the X direction and one for the Y direction. You can identify the X and Y fields using the elements **IsX** and **IsY**. The above examples show these notations.

Datum and projection for X and Y fields The X and Y field definition lines must contain the **PROJ** element identifying the datum and projection associated with the data. This element consists of the word **PROJ** followed in parentheses by the names of the projection and the datum. You must separate the projection and datum names using a comma. There must be no space between **PROJ** and the parentheses. Note that the projection name goes before the datum name.

Examples:

```
EAST    15-23  REAL*8  MISSING(0)  PROJ(TMAMG54,AGD66)  IsX
NORTH   27-35  REAL*8  MISSING(0)  PROJ(TMAMG54,AGD66)  IsY
```

The definition file of the same name for the datum and projection must be available to INTREPID in the normal projection / datum directory. See INTREPID's supported datums and projections (R09) for information about the datums and projections supplied with INTREPID and about creating your own definitions.

If the datum or projection information is wrong at the time of import, you can make a correction later. You can make the correction using the Import tool (See "Setting Geographic Registration for vector datasets" in Importing to INTREPID datasets (T05).

Constant value assignment You can assign a constant value to a field instead of obtaining its value from the input data. Use notation as shown in this example

```
SURVEY=557 INTEGER*2
```

There must not be any spaces between the **=** character and the value being assigned.

Multiplying imported data by a specified factor You can specify a factor for multiplying all data imported to a field. Use the notation (e.g.,) **MULTIPLIER(1.76324)** in the field definition line.

References to field values lookup files You can specify a field values lookup file for the field. If you do this, INTREPID will obtain results from the lookup file using the input data and use these results as the field value.

The field value lookup file reference must be the last item in the line. The lookup file must have the extension **.leg** and reside in **INTREPID/lut**.

See "Fields associated with lookup tables" in INTREPID database, file and data structures (R05) and"Field header (.vec) information" in INTREPID database, file and data structures (R05) for further information.

Scientific notation INTREPID automatically detects scientific notation (e.g.,) 9.636000E-2. You do not need to declare it any differently.

Multiband fields

To specify a multiband field, repeat the field name the appropriate number of times with the band number in square brackets following the field name.

For example

```
MAGNETIC[0]39-48 REAL*4
MAGNETIC[1]29-38 REAL*4
MAGNETIC[2]59-68 REAL*4
```

It is essential that all of the lines defining a multiband field are together in the DDF in ascending order of bands. If the data for the bands is in a different order in the input file, simply specify the column numbers correctly for locating the data (i.e., a column number range in any line of a DDF can specify any part of a line of the input file).

'Grouping' for different sample rates in an import record

You can import data with different sample rates within a single import record. For example, your survey may contain one reading for data such as temperature, GPS location or height for every ten fiducial values and magnetic data readings.

>> To specify grouping for different sample rates

- Include the (e.g.) ten data readings, each in its own position, in single import records,
- Specify the field name and corresponding position for each occurrence in the record
- Use the **REPEAT** notation (e.g.) **REPEAT(1)** to distinguish between and specify the order of the readings.

  For each reading, INTREPID will create a data point in the vector dataset (i.e., in our example of ten readings per import record, one import record will result in ten data points).

See DDF Example 3 for an illustration of this feature.

BREAK ON / GROUP BY line

If your dataset has groups, you can identify the 'group by' field(s) using **BREAK ON** or **GROUP BY** statements.

The statement occurs after the field definition statements and commences with the words **BREAK ON** or **GROUP BY** followed by a space and the name of the field.

Examples:

```
GROUP BY LINE_NO
```

```
GROUP BY DATE
```

You can specify a number of 'group by' fields. If any 'group by' field changes during import, INTREPID will create a new group at that point.

More generally, you can use **BREAK ON / GROUP BY** in any situation where you wish to create a secondary key in the dataset. **BREAK ON / GROUP BY** creates a 'one to many' relationship.

Skipping records using IGNORE( )

Grouped data may contain some unwanted records. You can instruct INTREPID to skip records with a certain 'group by' value by including the **IGNORE** element in the **BREAK ON / GROUP BY** line. It consists of the word **IGNORE** followed in parentheses by the 'group by' field value whose records are to be skipped. There must be no space between **IGNORE** and the parentheses. You can have a number of these statements referring to the same 'group by' field– one for each value to be ignored.

Examples:

```
GROUP BY LINE_NO IGNORE(0)
GROUP BY LINE_NO IGNORE(999)
```

You can also instruct INTREPID to ignore blank lines using

IGNORE(BLANKS).

Multiple line input records and NEXTINPUTLINE

The data for one data point may be split over a number of lines of the input file. In this case you need to specify in the DDF that the data point fields continue on the following line.

To specify continuation of a data point, include the statement

NEXTINPUTLINE

If your input file has fixed length fields rather than delimiters, the column numbers must start from 1 again after the **NEXTINPUTLINE** statement.

DDF example 1 has a two line data point specification using **NEXTINPUTLINE**.

Exploranium GR820 import

The INTREPID DDF language contains three features to facilitate import from ASCII files containing Exploranium GR820 binary data blocks:

- Grouping of data with higher sampling rate within single import records (See 'Grouping' for different sample rates in an import record).
- The **MULTIPLIER( )** notation for modifying data using a factor during import (See 'Multiplying imported data by a specified factor' in Field definition lines.
- The GR820 data block data type (notation **GR820**). You can use a range of columns in the import file for a block of GR820 encoded data, and define it as a field using any field name (INTREPID will ignore it) of with the specified column numbers and data type **GR820**. INTREPID will import this data automatically into the dataset. See DDF Example 3 for an example. DDFs containing GR820 binary data block specifications must specify column numbers for all fields.

For files containing only GR820 binary data blocks, use the GR_820 option from the InputFormat menu in the Import tool. See "Importing to vector datasets" in Importing to INTREPID datasets (T05).

Protobuf Langauge Definition

As at V5.0, the functionally equivalent, but altered syntax, required to push the above language into a form suitable for support using the GOOGLE protobuf parsers and general library support is transitioned. This is a suprisingly terse specifictaion, with

the added advantage, that what you see in the published syntax file, is exactly the same file that is used to build the parsers and then add support for this functionality within the production tools within INTREPID.

```
// Here is the specification of the intrepid geophysics
// batch data description format language
// using GOOGLE/Protobuf schema
//
// All message objects are preceded by a _DDF for now to clearly
// distinguish them from parameters.
//
//
// This is simply due to the way the code receives the tasks:
// currently it is a simple if/else if approach. Could also be
// event based with protobuf being used for the actual event messages.
// copyright  Intrepid Geophysics March 2011

// Sets the namespace in C++, package in Java and Python
import “commontaskmodel.proto”;
package ddf;

enum Input_Format_Type
{
IFT_free_format = 0;  // assume commas
IFT_fixed_format = 1;
IFT_constant = 2;  // introduced field, not in original file, but set as a constant value
IFT_from_file = 4;
IFT_from_gr820 = 5;  // radiometrics spectrum ascii format
};

enum FormatHint
{
FH_DEFAULT = 0;
FH_DMS=1;  // treat number as being in Degrees: Minutes:seconds notation
FH_Vx=2; FH_Vy=3; FH_Vz=4; FH_Vtype=5;  // vector components
FH_Txx=10; FH_Txy=11; FH_Tzx=12; FH_Tyy=13; FH_Tyz=14; FH_Tzz=16; // standard
tensor
FH_Tuv=17; FH_Txy_Repeat=18; FH_Tuv_Repeat=19;  // falcon
FH_Omag=20; FH_Obearing=21; FH_Ox=22; FH_Oy=23; FH_Oz=24; FH_Otype=25;  //
observed data type support
FH_Oxx=30; FH_Oxy=31; FH_Ozx=32; FH_Oyy=33; FH_Oyz=34; FH_Ozz=35;
FH_Ouv=36; // observed tensor
FH_Gdip=40; FH_Gstrike=41; FH_Geo=42; FH_Sx=43; FH_Sy=44; FH_Sz=45;  // observed
geology foliation
```

```
FH_R=50; FH_I=51;

FH_DATE=60; FH_TIME=61; FH_YEAR=62; FH_MONTH=63; FH_DAY=64; FH_HOUR=65;
FH_MINUTE=66; FH_SECOND=67;

FH_JDAY=68; FH_SEC_MIDNIGHT=69;

FH_QREAL=80; FH_QI=81; FH_QJ=82; FH_QK=83;  // parts of a quaternion/rotation
matrix

}


//

// for each field in a comma separated file etc, describe the attributes required when it
goes to the database

message FieldInfo_DDF {

required string name = 1;

optional string bandname = 2;

optional ctm.DfaFileType ftype = 3;

optional ctm.AccessType accessType = 4 [ default = AT_INDEXED];

optional int32 band = 5  [ default = 0];

optional int32 bands = 6 [ default=1];  // not a multi-band field

optional bool free = 7 [ default = true];  // free format

optional double constant_value = 8;  // set a field in database, not in the input file

//optional string sepString = 8;  // normally not exposed in external message

optional string lutfile = 9;

optional int32 from = 10;

optional int32 to = 11;  // internally, length is used

optional string strbuf = 12; // not exposed externally in a message

optional string missing = 13;

optional ctm.FieldAlias alias = 14 [default = FA_NONE];

optional string constant_string = 15;

optional string units = 16;

optional string fullname = 17;

optional string comment = 18;

optional ctm.DataType dtype = 19 [default = DT_R8];

optional bool IsX = 20  [default = false];

optional bool IsY = 21  [default = false];

optional bool IsZ = 22  [default = false];

//optional bool hasproj = 22;

optional string proj = 23;

optional string datum = 24;

optional bool nullOnError = 25 [default = true]; // if an error and saving, insert a NULL

optional double scale = 26 [default = 1.0];

optional int32 repeat = 27;

optional int32 nrepeats = 28 [default = 0];
```

```
optional bool nosave = 29 [default = false];  // create the database record/field by default,
optional bool composite = 30 [default = false];// composite field for an object such as tensors
optional int32 composite_buf_index = 31 [default = -1];// index of first FieldInfo entry that is part of this composite
optional FormatHint formathint = 32 [ default = FH_DEFAULT];
// accumulated during import
optional int32 maxlength = 33 [default = 0];
optional int32 decimals = 34 [default = 0];
optional int32 errors = 35 [default = 0];
optional string height_datum = 36;
   optional string dms_format = 37;
}
message GroupBy_DDF {
//  aim to flag which fields have a one to many relationship, as you form the database
repeated string name = 1;
optional double ignore_value = 2 [default = 0]; //  if field has this value, ignore all records, skip on input
optional string ignore_string = 3;
}
message Delimiter_DDF {
required string delims = 1;
}
message PrimaryKey_DDF {  // currently not connected, used for drill holes
repeated string name = 1;
}
message NextInputLine_DDF {
// support for mixed columns  by physical record types
optional int32 logical_record_type = 1;
}
message Record_DDF {
// support for logical record type mixing
required string type = 1;  //  A , B, C etc,,  or collar head, surveys etc
optional int32 logical_record_type = 2;
}

message DataDescriptionKeywords_DDF {
// database/ file based operations
optional  ctm.DfaFileType Type = 1 [default = FT_POINT];
repeated  FieldInfo_DDF Field = 2;  // each column needs a field description
optional  GroupBy_DDF  GroupBy = 3;
optional  Delimiter_DDF Delimiter = 4;
```

```
    optional  PrimaryKey_DDF PrimaryKey = 5;
    optional  NextInputLine_DDF NextInputLine = 6;
    optional  Record_DDF Record = 7;

      optional string log = 255;
    }
```