

dtreflnmerge

SBC01-????-00

February 16, 1996

Prepared by: J.W. Pflugrath
Molecular Structure Corporation
3200 Research Forest Drive
The Woodlands, Texas 77381
(713) 363-1033
(713) 364-3628 FAX
jwp@msc.com

Prepared for: Contract No. 943072401
Mary Westbrook
Argonne National Laboratory
9700 South Cass Avenue
Argonne, Illinois 60439
(708) 252-8914
(708) 252-4021 FAX
westbrook@anl.el.anl.gov

Reviewed by:

Approved by:

Copyright © 1996, 1995, 1994 Molecular Structure Corporation
RESTRICTED RIGHT NOTICE SHORT FORM (JUNE 1987)

Use, reproduction, or disclosure is subject to restrictions set forth in Contract No. W-31
109-ENG-38 and Contract No. 943072401 with the University of Chicago, Operator of
Argonne National Laboratory

TABLE OF CONTENTS

1 SCOPE AND PURPOSE	3
1.1. Definitions and Abbreviations	3
2 BACKGROUND	3
3 D*TREK REFLECTION LIST FORMAT	4
4 RUNNING DTREFLNMERGE	5
4.1. Command line options	5
4.2. Examples	10
4.3. Tips	14
4.4. Incorrect examples	15
APPENDIX A REFLECTION LIST FIELDNAMES	17

1 Scope and Purpose

This document describes how to use the `dtreflnmerge` utility to merge and filter `d*TREK` reflection lists. You use `dtreflnmerge` to:

- merge two or more reflection lists into a single list,
- remove one or more fields from a reflection list,
- add one or more fields to a reflection list,
- set a field to a constant value,
- add or subtract a number to or from any numeric field,
- multiply or divide any numeric field by a number,
- include or exclude reflections with logical operations on fields such as `==`, `!=`, `>`, `>=`, `<`, and `<=`,
- reindex hkl's by multiplying by a 3x3 matrix,
- reduce hkl's to the asymmetric unit and then sort on hkl,
- calculate the resolution in Ångstroms for each reflection,
- prepare a reflection list for input to the `dtscalmerge` program, and
- assign reflections to different scaling batches by a variety of schemes.

You can combine any filtering operation with merging, so that you include only the reflections and fields that interest you in the output reflection list. As you will see, the filtering syntax is general enough that you will invent new uses for `dtreflnmerge`.

1.1 Definitions and Abbreviations

The definitions described in the `dtcollect`, `tCrefine`, and `dtpredict` documents are appropriate for `dtreflnmerge`.

2 Background

Lists of Bragg reflections are used in many crystallographic calculations. For example, during single crystal diffraction data collection you generate a reflection list when you find peaks in images. You also generate a list when you predict reflections in an image. You generate a list when you integrate Bragg reflections in images. Each of these lists can contain different kinds of information, so that reflections in the lists can have many different properties attached to them, such as the Miller index, intensity, observed position, calculated position and so on. Thus lists of reflections can have many different properties depending on the purpose of the reflection list. All reflections in a list have the same properties or fields. Reflections probably have different values for the fields, such as

different Miller indices, but they may also have the identical value for a field such as the crystal name or batch name.

The d*TREK reflection list format described briefly below was developed to encapsulate all possible reflection lists that might be used in crystallographic applications. You can consider a d*TREK reflection list to be a m by n array where m reflections have n fields, with both m and n limited only by the amount of virtual memory on the computer. A reflection list could also be thought of as a spreadsheet of m rows and n columns. The fields may be of type integer, floating point or string. Many crystallographic applications manipulate reflection lists. Each different application may require or expect a different set of properties in the reflection list. Each different application may calculate new properties and add them to the reflection list.

Because d*TREK reflection lists can contain widely different information, the `dtreflnmerge` utility was developed to manipulate lists outside of the standard crystallographic applications. The main use of `dtreflnmerge` is to merge reflection lists that result from integrating different data acquisition scans and to prepare a merged list for input to the `dtscalmerge` program.

3 d*TREK reflection list format

d*TREK reflection lists have a simple format which is described in detail in the Appendix. In a nutshell, a d*TREK reflection list consists of human-readable ASCII characters. The list has a short header followed by the list of reflections. The first line of the header contains 3 integer numbers which denote the number of integer fields, floating point fields and string fields for each reflection. This line is followed by the names of the fields, one per line. After the field names, each reflection occupies one line that contains values for the fields in the header in free format. Integer field names usually begin with a lowercase `n`, as in `nH`, `nK`, and `nL`. Floating point field names usually begin with a lowercase `f`, as in `fIntensity` and `fSigmaI`. String field names usually begin with a lower case `s`, as in `sBatch`. Other fields are optional. Although d*TREK programs recognize and use a set of standard field names, you are free to add or use any field names you wish for your own purposes. In the reflection list, the fields must be ordered so that the integer fields come first, the floating point fields next and string fields last. The first 3 integer fields must be `nH`, `nK`, and `nL`. The first two floating point fields must be `fIntensity`, `fSigmaI`. There are no required string fields. Next is an example of a minimal reflection list with a single reflection.

```
3 2 0
nH
nK
nL
fIntensity
fSigmaI
    4    1    2 8934.24 94.521
```

Each reflection record in a list must contain a value for each and every field specified in the header. That is, all reflections in a list contain the same fields, though they may or may not have different values for the fields. Lists are not assumed to be sorted in any particular order.

If necessary, `dtreflnmerge` reads the header of an input image specified on the command line to get the crystal unit cell dimensions and spacegroup. This is a prerequisite for certain filtering operations such as reducing Miller indices to an asymmetric unit or calculating the resolution of reflections. The image header format is described elsewhere (for example, see the `dtpredict` documentation).

4 Running dtreflnmerge

After `dtreflnmerge` has been installed and placed in your `PATH`, just enter `dtreflnmerge` along with command line options to run it. It parses the command line arguments, reads any required input reflection lists and image headers and writes a new reflection list. Messages are written to `stdout` and `stderr` as required. The syntax for running `dtreflnmerge` is:

```
dtreflnmerge [output_field_options] input_file1 [input_file_options]
              [input_file2 [input_file_options] ...]
              output_file [output_file_options]
```

Only one reflection list (`output_file`) is output per `dtreflnmerge` run. As a safety precaution, `dtreflnmerge` will not overwrite an existing file when it tries to create the output file. Note that if you use any options that require spacegroup information, then the environment variable `DTREK_SPACEGROUP_FILE` should also define a valid d*TREK spacegroup file.

4.1 Command line options

You tell `dtreflnmerge` what to do with options specified on the command line. Options are processed in the order they appear on the command line. The syntax of the command line options is described next.

`output_field_options`

Enter the names of the fields to exclude from the output file each preceded by a - sign. For example with
-fObs_pixel0 -fCalc_Xmm
neither the fObs_pixel0 nor the fCalc_Xmm field will appear in the output_file.

input_file(s)

The name of a d*TREK reflection list file. As many input files as desired may be placed on the command line, they will be combined in the output file in the order presented on the command line. Also the same input file may be repeated with different input file options in order to select different subsets of reflections from the same input file. All reflections in an input file will be copied to the output except those deselected by an input file option. In other words, all reflections are selected by default.

input_file_options

Input file options follow the name of the input file and apply only to that input file. The options are applied in the order given, so care must be taken to achieve the desired result. Input file options take several forms as described below. There can be no whitespace in any input file options. That is, do not use space, tab or newline characters to separate the subfields of an input file option. Also note that with most shells the !, >, <, and * characters must be escaped typically by preceding them with a backslash \.

Summary of forms

-fieldresultBOOLvalue (BOOL: == != > < >= <=)
+fieldresultBOOLvalue
-fieldname=value
+fieldname=value
-fieldnameMATHvalue (MATH: += -= *= /=)
+fieldnameMATHvalue
-reindex=m11,m12,m13,m21,m22,m23,m31,m32,m33

Form 1

Use the following syntax to exclude (deselect) reflections or to include (select) previously excluded reflections:

-fieldresultBOOLvalue (- means **exclude**)
+fieldresultBOOLvalue (+ means **include**)

where

- means exclude reflections which fit selection

+ means include reflections which fit selection

fieldresult

is a either single fieldname or two fieldnames of the same type (integer, float, string) separated by a +, -, *, or / character. With

two fieldnames, the math operation is performed with the values of the fields and the result is compared to `value`. With a single fieldname, the value of that field is compared to `value`. The fieldnames must already exist in the input file.

`BOOL` is one of

`==` (is equal to)
`!=` (is not equal to)
`<` (is less than)
`>` (is greater than)
`>=` (is greater than or equal to)
`<=` (is less than or equal to)

which defines the type of comparison to make between `fieldresult` and `value`.

`value`

is the value to compare to `fieldresult`. If `fieldresult` is of type string, and `value` ends in an asterisk, then the comparison is tested only up to the number of characters before the asterisk in both `fieldresult` and `value`. In effect, the asterisk *if it is the last character* acts like a wildcard character.

Example 1.1: Exclude reflections with intensities between 50 and 100:

```
-fIntensity<=100.0 +fIntensity<=50.0
```

Example 1.2: Exclude reflections in an ice ring around 3.8 Å:

```
-fResolution<3.85 +fResolution<3.75
```

Example 1.3: Exclude reflections with $I/\sigma I < 3$:

```
-fIntensity/fSigmaI<3
```

Example 1.4: Exclude reflections with `sBatch` names beginning with `L0`, but include `L001` in any case:

```
-sBatch==L0* +sBatch==L001
```

Form 2

```
-fieldname=value
```

```
+fieldname=value
```

where

+ - there is no difference between using a - or + character, both mean operate on `fieldname`.

`fieldname`

is any fieldname, either existing or new. Do not use fieldnames with +, -, *, /, =, >, <, or ! characters in them..

= is the equals sign

`value`

is a value to set the field named by `fieldname` to for all reflections in the reflection list.

The type of field (integer, float, or string) is determined from the first letter of fieldname (n, f or s) or from the value type.

Example 2.1: Add sBatch and nDetNum fields to all reflections:
-sBatch=X001 -nDetNum=1

Example 2.2: Make all h's equal to 1 (Why? It is just an example!):
-nH=1

Form 3

-fieldnameMATHvalue
+fieldnameMATHvalue

where

+ - there is no difference between using a - or + character, both mean operate on fieldname.

fieldname

is any *existing integer or float* fieldname. Do not use fieldnames with +, -, *, /, =, >, <, or ! characters in them..

MATH is one of

+= (add to)
-= (subtract from)
*= (multiply by)
/= (divide into)

which defines the math operation to perform between the value of fieldname and value.

value

is a value to use in the math operation.

The type of field (integer or float) is determined from the first letter of fieldname (n, f) or from the value type.

Example 3.1: Multiply all sigma values by 1.4142:
-fSigmaI*=1.4142

Example 3.2: Subtract 1 from all k indices:
-nK-=1

Form 4

-reindex=m11,m12,m13,m21,m22,m23,m31,m32,m33

where

-reindex=

indicates that a reindexing matrix follows

m11,m12,m13,m21,m22,m23,m31,m32,m33

are 9 elements of a 3x3 matrix that each reflection's *hkl* will be multiplied by to get a transformed *hkl*. The matrix elements must be separated by commas *without any whitespace*.

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \times \begin{bmatrix} h \\ k \\ l \end{bmatrix} = \begin{bmatrix} h' \\ k' \\ l' \end{bmatrix}$$

Example 4.1: Switch *h* and *k* and make *l* equal to *-l*:

`-reindex=0,1,0,1,0,0,0,0,-1`

The above would be used to change the polarity of a tetragonal or hexagonal spacegroup. The determinant of the matrix should be positive or you will change the handedness of your anomalous data which you almost never want to do.

`output_file`

The name of the output d*TREK reflection file. As a safety precaution, this file will not be written if it already exists, instead an error message will be output.

`output_file_options`

As above for `input_file_options`, plus the following:

`-hHEADER`

where the file `HEADER` contains a valid d*TREK image header that is used for information such as crystal properties (unit cell dimensions and spacegroup). The header is read in and a crystal object created for potential use with the `-reduce` and `-reso` options.

`-reduce`

Reduce the hkl's to a single asymmetric unit and add the fields `nPackedHKL`, `nReducedH`, `nReducedK`, `nReducedL`, `nAnomFlag`, and `nCentPhase`. Sort the reflections on field `nPackedHKL`. Any or all of these fields may be removed by naming them above in `output_field_options`. The `-hHEADER` option is a prerequisite to this option. In this case the header file must have valid crystal and spacegroup information.

`-reso`

Add the field `fResolution` if it does not already exist in the reflection list, then calculate the resolution of each reflection in Ångstroms and place the result in the `fResolution` field. The `-hHEADER` option is a prerequisite to this option. In this case the header file must have valid crystal unit cell information.

4.2 Examples

A few simple examples were given in the previous section that described the command line options of `dtreflnmerge`. This section will give examples of more complicated scenarios.

Example 1

You have collected two scans of 2° oscillations images on an imaging plate system. The first scan consists of 50 images, while the second consists of 20 images. The reflection intensities in the images have all been integrated. You wish to merge the integrated reflection files in order to scale all the data together in a subsequent step with the `dtscalmerge` program.

```
1 # Example 1
2 dtreflnmerge scan1_001.ref -sBatch=S101 \
3             scan1_002.ref -sBatch=S102 \
4             ...
5             scan1_050.ref -sBatch=S150 \
6             scan2_001.ref -sBatch=S201 \
7             scan2_002.ref -sBatch=S202 \
8             ...
9             scan2_020.ref -sBatch=S220 \
10            scan_1_2.ref -hscan1_001.img -reduce
```

In this example, the first 50 reflection files from the first scan are named `scan1_001.ref`, `scan1_002.ref`, and so on up to `scan1_050.ref`. The reflection files from the second scan are named `scan2_001.ref` through `scan2_020.ref`.

Line 2 Start the program and read in the first input reflection file. Since the input file does not have the `sBatch` field, add one with a unique name for the reflections in this file. The backslash (`\`) is just Unix continuation character. On a VMS system this would be a dash (`-`).

Lines 3-9 These list the other input files and assign unique batch names to them.

Line 10 This gives the output reflection list file, the name of an image file from which a header is read with the crystal spacegroup information and finally the `-reduce` option to reduce the reflections to an asymmetric unit.

Example 2

You discover there is a problem with the integration of image number 34 from scan 1 and wish to remove its reflections from the merged file of Example 1.

```
1 # Example 2
2 dtreflnmerge scan_1_2.ref -sBatch==S134 scan_1_2b.ref
```

Line 2 Reflections with a `sBatch` value of `S134` are deselected and do not appear in the output file.

Example 3

You discover that reflections in the second scan were mis-indexed partly because you had the `b*` axis perfectly aligned with the crystal rotation axis. All the `k`'s are off by +1, so you wish to subtract exactly 1 from all the `k` values of reflections in the second scan. You need to remerge to repair this.

```
1 # Example 3
2 dtreflnmerge scan1_001.ref -sBatch=S101 \
3 scan1_002.ref -sBatch=S102 \
4 ...
5 scan1_050.ref -sBatch=S150 \
6 scan1.ref
7 dtreflnmerge scan2_001.ref -sBatch=S201 \
8 scan2_002.ref -sBatch=S202 \
9 ...
10 scan2_020.ref -sBatch=S220 \
11 scan2.ref
12 dtreflnmerge scan1.ref scan2.ref -nK-=1 \
13 scan1_2c.ref -hscan1_001.img -reduce
```

Lines 2-6 All the reflections from the first scan are merged into a file named `scan1.ref`.

Lines 7-11 All the reflections from the second scan are merged into a file named `scan2.ref`.

Lines 12-13 The two merged reflection lists are merged together, but the `k`'s from the second scan all have 1 subtracted from them. Finally the reflections are reduced to the asymmetric unit.

An alternate way to achieve the same result as the previous example given the result of Example 1 is:

```
1 # Example 3A
2 dtreflnmerge scan_1_2.ref -sBatch==S2* \
3 scan_1_2.ref -sBatch==S1* -nK-=1 \
4 scan1_2c.ref -hscan1_001.img -reduce
```

- Line 2 Exclude all reflections from scan 2 which leaves only reflections from scan 1.
- Line 3 Re-read the input reflection list and exclude all reflections from scan 1 which leaves only reflections from scan 2. Subtract 1 from all the k values.
- Line 4 Reduce the hkl's to the asymmetric unit. Since the `-reduce` option places the reduced hkl's in the fields `-nReducedH`, `-nReducedK`, and `-nReducedL`, the original indices are still intact and thus 1 can be subtracted as on Line 3. The reduced hkl's are simply recalculated from the hkl's.

Example 4

You decide that the ribosome crystal really did not diffract beyond 1.1 Å resolution even though you collected data to 0.9 Å resolution. You wish to exclude reflections beyond 1.1 Å resolution from the result of the previous example.

```
1 # Example 4
2 dtreflnmerge scan1_2c.ref \
3 scan1_2d.ref -hscan1_001.img -reso -fResolution<1.1
```

- Line 3 Since the reflection list did not have a `fResolution` field, you need to add one. Use `-hscan1_001.img` to get unit cell information. Use `-reso` to add the `fResolution` field. Use `-fResolution<1.1` to exclude reflections with a resolution beyond 1.1 Å. Remember that `-reso` is an *output* file option.

Example 5

You decide that the ribosome crystal really died after the first scan, so that it not diffract beyond 2.0 Å resolution during the second scan. You wish to exclude reflections beyond 2.0 Å resolution from the second scan only.

```
1 # Example 5
2 dtreflnmerge scan1_2d.ref -sBatch==2* \
3 scan1_2d.ref -sBatch==1* -fResolution<2.0
4 scan1_2e.ref
```

- Line 2 Exclude all reflections from scan 2 (keep all scan 1 reflections).
- Line 3 Exclude all reflections from scan 1 (leaving only scan 2 reflections), then exclude all reflections with a resolution beyond 2 Å. This works here because the `fResolution` field already exists in `scan1_2d.ref` from the previous example.

Line 4 There is no need for the `-reso` option since the `fResolution` field already exists.

Here is an alternate way to achieve the same result:

```
1 # Example 5A
2 dtreflnmerge scan1_2d.ref -sBatch==2* +fResolution>=2.0 \
3               scan1_2e.ref
```

Line 2 Exclude all reflections from scan 2, then *include* all reflections within 2 Å resolution from all scans. Since only scan 2 reflections were excluded in the first instance, this has the desired effect.

Example 6

You decide that it would be better to have separate scale factors for reflections in the top and bottom halves of the images. The center pixel is at 4095 (you are using the new MSC 8K by 8K detector). If you use the same files as in Example 1 you might have:

```
1 # Example 6
2 dtreflnmerge scan1_001.ref -fObs_pixel0>4095 -sBatch=S101L \
3               scan1_001.ref -fObs_pixel0<=4095 -sBatch=S101H \
4               ...
5               scan2_020.ref -fObs_pixel0>4095 -sBatch=S220L \
6               scan2_020.ref -fObs_pixel0<=4095 -sBatch=S220H \
7               scan_1_2.ref -hscan1_001.img -reduce
```

Lines 2 & 5 Each input reflection list will have to be read in twice, once for each half. In this line, reflections with high pixel coordinates are excluded. The batch names end in `L` to indicate that the reflections come from the low coordinate half.

Lines 3 & 6 Exclude reflections with low pixel coordinates and use batch names ending in `H` to indicate that the reflections come from the high coordinate half.

Example 7

You have too many fields in the reflection list that you no longer need. You want to eliminate these unneeded fields. In any case, you cannot remove the `nH`, `nK`, `nL`, `fIntensity` and `fSigmaI` fields.

```
1 # Example 7
2 dtreflnmerge -nPackedHKL -nCenPhase -nAnomFlag \
3               -fObs_pixel0 -fObs_pixell \
4               scan_1_2.ref \
```

5 scan_1_2f.ref

Lines 2-3 Fields you want eliminated from the output reflection list altogether are specified *before* the first input reflection file.

4.3 Tips

This section describes some things you should know about `dtreflnmerge`

Reflection lists can have any number of fields and different field types. What happens when you merge reflection lists with fieldnames that do not match? The answer is that the resulting output reflection list contains all the fieldnames of the input reflection lists. Those reflections which did not have an original value for a specific field will get a default value. The default value depends on the field type:

Type	Default value
integer	0
float	-999.00
string	?

In many cases, you may not care what the default values are since you may have values for those fields recalculated by a subsequent program. In other cases, these extra fields are a nuisance. See Example 7 above on removing unwanted fields.

Reflection lists can also have their fields in different orders. `dtreflnmerge` takes into account any differences in the order of the fields when merging reflection lists. This can be time-consuming though.

Hint: Merging reflection lists with different fields is very time-consuming. Try to eliminate unneeded fields early on in any merging process. Use an intermediate reflection list if necessary.

It is also very time-consuming to add new fields to a reflection list. It is faster to add fields to smaller lists and merge them than it is to merge them and then add fields.

Hint: Add all required fields early on in any merging process. Use an intermediate reflection list if necessary.

There are some steps we can take to make the `dtscalmerge` (not shown) faster. You are aware that `dtscalmerge` adds the fields `nBatchIndex` and `fSTLSq` to the reflection list it uses if they are not already present. `dtscalmerge` also does not assume the

reflections are reduced to an asymmetric unit and sorted, so it performs this step. If the fields added by the `-reduce` option are not present, they are added at this time which can take quite some time for a large list. If you have to run `dtscalmerge` several times to be satisfied with the results, then you might end up doing this often. **Thus is it a good idea to add these fields to the input reflection list before running `dtscalmerge`.**

If we revisit Example 1 in the previous section and change it to be faster for ultimate purpose, we end up with the following:

```
1 # Example 1 FASTER
2 dtreflnmerge scan1_001.ref s101.ref -sBatch=S101 \
3     -nBatchIndex=0 -fSTLsq=0.0 -hscan1_001.img \
4     -reso -reduce
5 dtreflnmerge scan1_002.ref s102.ref -sBatch=S102 \
6     -nBatchIndex=0 -fSTLsq=0.0 -hscan1_001.img \
7     -reso -reduce
8 #...
9 # Do the above for all 70 integrated reflection lists just once
10
11 dtreflnmerge s101.ref s102.ref s103.ref \
12     ... s220.ref \
13     scan_1_2.ref
```

Lines 2-4 An input list is read in. Fields that will be used by a subsequent `dtscalmerge` run are added explicitly by `-nBatchIndex` and `-fSTLsq`. Also fields are implicitly added with the `-reso` and `-reduce` output file options.

Lines 5-7 The operations of lines 2-4 are repeated for each original input reflection list.

Lines 11-13 The intermediate reflection lists are all merged together. Since the `-reduce` option is not used, the output file `scan_1_2.ref` is not sorted on `hkl`, but rather on the order of the input reflection lists.

4.4 Incorrect examples

Below are simple examples of incorrect syntax.

`-fIntensity>3*fSigmaI` **WRONG!** The right-hand side must be a simple number. Use `-fIntensity/fSigmaI>3` instead.

`-fIntensity=*5.0` **WRONG!** Use `*=` instead.

-fIntensity==100*

WRONG! 100* is not a valid number.

dtreflnmerge scan1_001.ref **-reduce** -fObs_pixel0>4095 -sBatch=S101L \
scan_1_2.ref -hscan1_001.img

WRONG! -reduce is not an input file option.

dtreflnmerge scan1_001.ref -fObs_pixel0>4095 -sBatch=S101L \
scan_1_2.ref **-reduce** -hscan1_001.img

WRONG! -reduce needs to come after the
-hscan1_001.img option.

dtreflnmerge scan1_001.ref -fObs_pixel0>4095 -sBatch=S101L \
scan_1_2.ref -hscan1_001.img **-nPackedHKL**

WRONG! **-nPackedHKL** as shown here needs a
comparison or assignment, it cannot stand alone in this
position.

-nH=nReducedH

WRONG! The value on the right-hand side cannot contain
be a function of a fieldname, it must be a number or string.

-sBatch==J1*1

Not wrong, but * is not a wildcard here since it is not the
last character of the right-hand side

Appendix A Reflection list fieldnames

Below is a list of fieldnames used in various d*TREK programs. Remember: reflection lists are not required to use only these fieldnames.

Integer fieldnames

nH	Miller h index, range $-511 \leq h \leq 511$
nK	Miller k index, range $-511 \leq k \leq 511$
nL	Miller l index, range $-1023 \leq l \leq 1023$
nPackedHKL	Packed hkl of a reflection: $[(h+512) \ll 21] + [(k + 512) \ll 11] + 1 + 1024$
nReducedH	Miller h index when nH, nK, nL are reduced to an asymmetric unit
nReducedK	Miller k index when nH, nK, nL are reduced to an asymmetric unit
nReducedL	Miller l index when nH, nK, nL are reduced to an asymmetric unit
nAnomFlag	+1 when hkl is an F^+ , -1 when hkl is a F^- (even if centric)
nCentPhase	0 when acentric, -1 when systematically absent, >0 phase is restricted to $nCentPhase \div 12 \times \pi$ radians and that plus π radians.
nDetector_number	The detector number of a reflection in a multi-detector system
nNonunf_flag	The non-uniformity type to apply to the reflection
nReflnNum1	Reflection number for difference vector calculation
nReflnNum2	Reflection number for difference vector calculation
nOriginalReflnNum	
nFrequency	Frequency of a difference vector

Floating point fieldnames

fIntensity	Estimated intensity, often corrected for all factors,
fSigmaI	Estimated standard deviation of the intensity
fObs_pixel0	Observed detector coordinate in the fast varying (i.e. first) direction
fObs_pixel1	Observed detector coordinate in the slow varying (i.e. second) direction
fObs_rot_mid	Observed rotation midpoint, may be in images or in degrees
fObs_rot_end	Observed rotation endpoint, may be in images or in degrees
fObs_rot_width	Observed rotation width, may be in images or in degrees
fObs_img_mid	Observed rotation midpoint, in images
fObs_Xmm	Observed X millimeter coordinate
fObs_Ymm	Observed Y millimeter coordinate
fCalc_pixel0	Calculated detector coordinate in the fast varying (i.e. first) direction
fCalc_pixel1	Calculated detector coordinate in the slow varying (i.e. slow) direction
fCalc_Xmm	Calculated X millimeter coordinate
fCalc_Ymm	Calculated Y millimeter coordinate
fCalc_rot_start	Calculated rotation angle start, in degrees
fCalc_rot_mid	Calculated rotation angle mid point or centroid, in degrees
fCalc_rot_end	Calculated rotation angle end point, in degrees
fCalc_rot_width	Calculated rotation angle width, in degrees
fCalc_partiality	Calculated partiality, as a fraction of 1
fResolution	Calculated resolution in Ångstroms
fRecip0	Reciprocal space coordinate in Å^{-1}

fRecip1	Reciprocal space coordinate in \AA^{-1}
fRecip2	Reciprocal space coordinate in \AA^{-1}
fRecipLength	Length of the reciprocal space vector $ d^* $
fFloatH	Miller index h as floating point
fFloatK	Miller index k as floating point
fFloatL	Miller index l as floating point
fCalc_polarz	Calculated polarization factor
fCalc_lorentz	Calculated Lorentz factor
fCalc_oblique	Calculated oblique incidence factor
fDeltaPx0	Difference between two reflection first pixel coordinates
fDeltaPx1	Difference between two reflection second pixel coordinates
fDeltaRot	Difference between two reflection rotation midpoint coordinates

String fieldnames

sRejectString	String describing rejection status
sBatch	String giving batch name of reflection