d*TREK Image Header

Rigaku/MSC, Inc

Revisions

Version	Date	Author	Scope
1.0	04/12/05	Robert Bolotovsky	Initial version
1.1	04/29/05	Robert Bolotovsky	Reorganized keyword annotation format, added syntax, examples. Added descriptions of mask bitmaps and RAXIS compression.

Table of contents

1. Introduction

1.1 Scope of the document

This document describes the d*TREK image header format and annotates keywords used in the header.

1.2 Intended audience

This document is intended for software developers, writing code to read images, written by Rigaku/MSC Instrument Servers.

1.3 References

Kabsch, W. J. Appl. Cryst. (1988), **21**, 916-924

Greenhough T.J. & Helliwell J.R. J. Appl. Cryst. (1982), **15**, 493-508

International Tables for Crystallography. Vol A. (1983)

ANSI X3.4-1986: 7-bit American National Standard Code for Information Interchange.

2. Overall d*TREK Image Header Description

2.1 Introduction

Images written by the Rigaku/MSC Instrument Server consist of an ASCII d*TREK header followed by binary pixel values. The d*TREK header is designed to contain all the parameters of the experiment.

2.2 Header length

The image header can be described as an ASCII string, the string length being a multiple of 512 characters. Space ' ‘ characters are used to pad the image header so its length is a multiple of 512 characters.

The minimum length of a header is 512 bytes. The maximum length of a header is the largest multiple of 512 that is less than 99999, i.e. 512 x 195. Thus, an integer describing the length of a header cannot have more than 5 digits.

Images within the same diffraction experiment scan are not required to have headers of the same length. For example, if the length of an image header is near a 512 byte boundary, extra digits in the various parameters (e.g. scan angle) can push it over the boundary to the next 512 byte increment. It is therefore necessary to read each image header within a scan to determine the header length for that particular image.

2.3 Header keyword-value syntax

The header consists of human readable text in the form:

Keyword= value;^J

Keyword is a case-sensitive string bounded by whitespace on the left, with no whitespace within. A keyword must begin with a letter or underscore and can contain only letters, digits and underscores.

'=' is the equal sign; there is no whitespace on the left side, but may or may not be on the right. So the following are all legal and may appear
Keyword=value;
Keyword= value;
Keyword= value;

value is the value of the keyword, which may be a string, a number (integer or float), or array(s) of numbers. A string value cannot contain the '***{***' , '***}***' or '***;***' characters, but can contain space characters ' '.

Elements in arrays are separated by whitespace.

';' is the semicolon character; it may or may not be preceded by whitespace.

'^J' is the newline character (ASCII 10).

Whitespace is any length sequence of space, tab, and/or newline characters.

Keywords (except HEADER_BYTES, which must appear first, see section 2.4) are currently sorted alphabetically. In older image files, however, they may not be sorted.

2.4 Header start

The header begins with a 2-character sequence *{ ^J* where *'^J'* is ASCII 10.

The third header character is the starting position of the required keyword-value pair:
HEADER_BYTES=*value*;

2.5 Header end

The unpadded part of the header always ends with the 4 character sequence:
} ^J ^L ^J where *'^J'* is ASCII 10 and *'^L'* is ASCII 12. With this one can use the Unix 'more' command to view image headers and avoid viewing binary data.

Padding will occur after the *^L ^J* if necessary.

3. d*TREK Image Header Keywords

3.1 Introduction

This section describes keywords present in the d*TREK image header.

Except for several image format keywords (see section 3.3), the keywords described below don't have to be present in every d*TREK image header. Also, there could be other keywords, not covered by this document. For example, some keywords may be required for processing images with d*TREK, but other processing programs need not use or worry about those additional keywords. It is safe to assume that the information contained in additional keywords will not contradict the information contained in the keywords described in this document.

3.2 Version keywords

The following keywords are related to the header version.

Keyword:	**DTREK_VERSION**
Syntax:	DTREK_VERSION= *version*;
Type:	String
Description:	The version of d*TREK that was used when the header was generated.
Example:	`DTREK_VERSION= d*TREK version 9.4W -- Apr 15 2005;`

Keyword: **HEADER_VERSION**
Syntax: HEADER_VERSION= *version*;
Type: String
Description: The version of the header documentation to which the keywords in the header conform.
Example: `HEADER_VERSION= 1.0;`

3.3 Image file format keywords

The following keywords describe the image file format.

Keyword: **HEADER_BYTES**
Syntax: HEADER_BYTES= *n*;
Type: Integer
Description: The length of the entire header in bytes.
Remarks: This keyword is present in every header and comes first in every header. Please refer to section 2.2 for information on the allowed header lengths. Note: there must be exactly 5 characters between the equal sign and the semicolon character.
Example: `HEADER_BYTES= 2048;`

Keyword: **DIM**
Syntax: DIM= *n*;
Type: Integer
Description: The number of dimensions in the image.
Remarks: This keyword is present in every header and its value is usually equal to 2.
Example: `DIM= 2;`

Keyword: **SIZE1**
Syntax: SIZE1= *n*;
Type: Integer
Description: The number of pixels along the first direction. The first direction is the fastest varying direction during detector read-out.
Remarks: This keyword is present in every header. If a file contains no binary pixel values, then this keyword must have a value of 0.
Example: `SIZE1= 512;`

Keyword: **SIZE2**
Syntax: SIZE2= *n*;
Type: Integer
Description: The number of pixels along the second direction.
Remarks: This keyword is present in every header. If a file contains no binary pixel values, then this keyword must have a value of 0.
Example: `SIZE2= 512;`

Keyword: **BYTE_ORDER**
Syntax: BYTE_ORDER= *byte_order*;
Type: String
Possible values: `big_endian | little_endian`
Description: The byte order of the data.
Remarks: This keyword is present in every header.
Example: `BYTE_ORDER= big_endian;`

Keyword: **Data_type**
Syntax: Data_type= *data_type*;
Type: String
Possible values: `signed char | unsigned char | short int | long int | unsigned short int | unsigned lont int | float IEEE | Compressed | Other_type`
Description: The type of the data.

`signed char`	signed one byte integer value
`unsigned char`	unsigned one byte integer value
`short int`	signed 2-byte integer value
`long int`	signed 4-byte integer value
`unsigned short int`	unsigned 2-byte integer value
`unsigned long int`	signed 4-byte integer value
`float IEEE`	IEEE floating point values
`Compressed`	The data is compressed. There must be a COMPRESSION keyword describing the algorithm.
`Other_type`	some other representation. There should be a DATA_OTHER keyword with more information.

Remarks: This keyword is present in every header.
Example: `Data_type= short int;`

Keyword:	**COMPRESSION**
Syntax:	COMPRESSION= *compression_type*;
Type:	String
Possible values:	`none`
Description:	The type of compression used for the entire image data.
Remarks:	Currently there is no support for entire image data compression.
Example:	`COMPRESSION= none;`

Keyword:	**RAXIS_COMPRESSION_RATIO**
Syntax:	RAXIS_COMPRESSION_RATIO= *r*;
Type:	Number
Description:	The compression factor used in the R-AXIS compression scheme for individual pixels in the image. Please refer to Appendix C for details.
Remarks:	The presence of this keyword indicates that the individual pixel values are compressed using the R-AXIS compression scheme.
Example:	`RAXIS_COMPRESSION_RATIO= 8;`

Keyword:	**SATURATED_VALUE**
Syntax:	SATURATED_VALUE= *n*;
Type:	Integer
Description:	The maximum value that a pixel can have.
Example:	SATURATED_VALUE= 1048544;

3.4 Source keywords

The following keywords are related to the radiation source properties.

Keyword:	**SOURCE_VECTORS**
Syntax:	SOURCE_VECTORS= s_{0x} s_{0y} s_{0z} s_{1x} s_{1y} s_{1z} s_{2x} s_{2y} s_{2z};
Type:	Set of 9 numbers
Description:	Specifies the direction s_0 of the source vector plus two vectors s_1 and s_2 perpendicular to s_0 around which the source vector rotation is refined.
Remarks:	In the d*TREK convention, s_0 points from the crystal *towards* the source, i.e. s_0 is antiparallel to the incident X-ray beam (see Appendix A.). Currently, the 9 values are always: **`0 0 1 0 1 0 1 0 0`**
Example:	`SOURCE_VECTORS= 0.0000 0.0000 1.0000 0.0000 1.0000 0.0000 1.0000 0.0000 0.0000;`

Keyword: **SOURCE_POLARZ**
Syntax: SOURCE_POLARZ= f_p pn_x pn_y pn_z;
Type: Set of 4 numbers
Description: The degree of polarization of the source f_p and a vector *pn* normal to the plane of polarization as described by Kabsch (1988).
Example: `SOURCE_POLARZ= 0.5 1.0 0.0 0.0;`

Keyword: **SOURCE_CROSSFIRE**
Syntax: SOURCE_CROSSFIRE= c_{11} c_{12} c_{12} c_{22};
Type: Set of 4 numbers
Description: The source crossfire in degrees.
Example: `SOURCE_CROSSFIRE= 0.0002 0.0002 0.0000 0.0000;`

Keyword: **SOURCE_WAVELENGTH**
Syntax: SOURCE_WAVELENGTH= *nNum_wavelengths fWavelength1 … fWavelengthN*;
Type: Set of (*nNum_wavelengths* + 1) numbers
Description: The number of wavelengths *nNum* followed by the value of each wavelength in Ångstroms.
Remarks: Currently, only one wavelength per image is supported.
Example: `SOURCE_WAVELENGTH= 1.0000  1.5418;`

Keyword: **SOURCE_AMPERAGE**
Syntax: SOURCE_AMPERAGE= *info*;
Type: String
Description: The operating current of the source.
Remarks: This keyword is a comment.
Example: `SOURCE_AMPERAGE= 45.0000;`

Keyword: **SOURCE_VOLTAGE**
Syntax: SOURCE_VOLTAGE= *info*;
Type: String
Description: The operating voltage of the source.
Remarks: This keyword is a comment.
Example: `SOURCE_VOLTAGE= 45.0000;`

Keyword: **SOURCE_FOCUS**
Syntax: SOURCE_FOCUS= *info*;
Type: String
Description: The focal spot of the source.
Remarks: This keyword is a comment.
Example: `SOURCE_FOCUS= 0.1500;`

3.5 Crystal keywords

The following keywords are related to crystal properties.

Keyword:	**CRYSTAL_UNIT_CELL**
Syntax:	CRYSTAL_UNIT_CELL= *a b c* $\alpha\ \beta\ \gamma$;
Type:	Set of 6 numbers
Description:	Specifies the crystal unit cell in Ångstroms and degrees.
Example:	`CRYSTAL_UNIT_CELL= 11.5519 13.7598 30.3361 74.1758 77.5682 81.7596;`

Keyword:	**CRYSTAL_ORIENT_ANGLES**
Syntax:	CRYSTAL_ORIENT_ANGLES= *angle_1 angle_2 angle_3;*
Type:	Set of 3 numbers
Description:	Specifies the rotation (in degrees) of the crystal away from the standard orientation when the crystal goniometer angles are all 0.
Example:	`CRYSTAL_ORIENT_ANGLES= -172.1934 52.5001 119.4073;`

Keyword:	**CRYSTAL_ORIENT_VECTORS**
Syntax:	CRYSTAL_ORIENT_VECTORS= $c_{1x}\ c_{1y}\ c_{1z}\ c_{2x}\ c_{2y}\ c_{2z}\ c_{3x}\ c_{3y}\ c_{3z}$;
Type:	Set of 9 numbers
Description:	Three 3D vectors c_1 c_2, c_2 *and* c_3 specify the axes of rotation for the CRYSTAL_ORIENT_ANGLES.
Remarks:	Currently, these values are always: `1 0 0 0 1 0 0 0 1`
Example:	`CRYSTAL_ORIENT_VECTORS= 1 0 0 0 1 0 0 0 1;`

Keyword:	**CRYSTAL_SPACEGROUP**
Syntax:	CRYSTAL_SPACEGROUP= *n;*
Type:	Integer
Description:	The spacegroup of the crystal from the International Tables.
Example:	`CRYSTAL_SPACEGROUP= 2;`

Keyword:	**CRYSTAL_MOSAICITY**
Syntax:	CRYSTAL_MOSAICITY= *mos;*
Type:	Number
Description:	The effective mosaic spread of the crystal in degrees, as described by Greenhough and Helliwell (1982). Because this is the effective mosaicity, it may include source divergence and cross-fire.
Example:	`CRYSTAL_MOSAICITY= 1.3900;`

3.6 Crystal goniometer keywords

The following keywords are related to the crystal goniometer properties. Currently only rotation axes are supported.

Keyword: **CRYSTAL_GONIO_NUM_VALUES**
Syntax: CRYSTAL_GONIO_NUM_VALUES= *n;*
Type: Integer
Description: The number of crystal goniometer axes.
Example: `CRYSTAL_GONIO_NUM_VALUES= 3;`

Keyword: **CRYSTAL_GONIO_NAMES**
Syntax: CRYSTAL_GONIO_NAMES= $name_1$ $name_2$ $name_3$ ... $name_n$;
Type: Set of *n* strings
Description: The names of the crystal goniometer axes. The number of names *n* is given by CRYSTAL_GONIO_NUM_VALUES. They must appear in the order from the goniometer baseplate to the crystal. The rotations of the axes will be applied in the reverse order.
Example: `CRYSTAL_GONIO_NAMES= Omega Chi Phi;`

Keyword: **CRYSTAL_GONIO_VECTORS**
Syntax: CRYSTAL_GONIO_VECTORS= g_{1x} g_{1y} g_{1z} g_{2x} g_{2y} g_{2z} g_{3x} g_{3y} g_{3z} ... g_{nx} g_{ny} g_{nz};
Type: Set of 3 x n numbers
Description: The vectors describing goniometer rotation axes. The number of vectors *n* and their order correspond to the number and order of names in CRYSTAL_GONIO_NAMES.
Example:
```
CRYSTAL_GONIO_VECTORS= 1.0000 0.0000 0.0000 0.0000
-1.0000 0.0000  1.0000 0.0000 0.0000;
```

Keyword: **CRYSTAL_GONIO_VALUES**
Syntax: CRYSTAL_GONIO_VALUES= g_1 g_2 g_3 ... g_n;
Type: Set of *n* numbers
Description: The goniometer axes angles at the datum position. That is, the goniometer angles when the crystal is at a rotation angle of 0. The number of values *n* and their order correspond to the number and order of names in CRYSTAL_GONIO_NAMES.
Example: `CRYSTAL_GONIO_VALUES= 0.0000 15.0000 0.0000;`

Keyword: **CRYSTAL_GONIO_VALUES_MAX**
Syntax: CRYSTAL_GONIO_VALUES_MAX= g_{1max} g_{2max} g_{3max} … g_{nmax};
Type: Set of *n* numbers
Description: The maximum allowable goniometer axes angle positions. The number of values *n* and their order correspond to the number and order of names in CRYSTAL_GONIO_NAMES.
Example: `CRYSTAL_GONIO_VALUES_MAX= 180.0000 90.0000 360.0000;`

Keyword: **CRYSTAL_GONIO_VALUES_MIN**
Syntax: CRYSTAL_GONIO_VALUES_MIN= g_{1min} g_{2min} g_{3min} … g_{nmin};
Type: Set of *n* numbers
Description: The minimum allowable goniometer axes angle positions. The number of values *n* and their order correspond to the number and order of names in CRYSTAL_GONIO_NAMES.
Example: `CRYSTAL_GONIO_VALUES_MIN= -180.0000 0.0000 0.0000;`

Keyword: **CRYSTAL_GONIO_UNITS**
Syntax: CRYSTAL_GONIO_UNITS= $units_1$ $units_2$ $units_3$ … $units_n$
Type: Set of *n* strings
Possible values: **`deg | mm`**
Description The units of the goniometer axes. The number of strings *n* and their order correspond to the number and order of names in CRYSTAL_GONIO_NAMES.
Remarks: Currently all these strings must be `deg`.
Example: `CRYSTAL_GONIO_UNITS= deg deg deg;`

3.7 Detector keywords

The following keywords are related to the detector properties.

Keyword: **DETECTOR_NUMBER**
Syntax: DETECTOR_NUMBER= *n;*
Type: Integer
Description: The number of detectors.
Example: `DETECTOR_NUMBER= 1;`

Keyword: **DETECTOR_NAMES**
Syntax: DETECTOR_NAMES= *dname$_1$ dname$_2$ dname$_3$ ... dname$_n$;*
Type: Set of *n* strings
Description: Specifies the names of each of the detectors. The number of strings *n* is given by **DETECTOR_NUMBER.** These strings are used as prefixes to the keywords, describing properties of the corresponding detectors. For readability of the image header, each detector name should end with an underscore character.
Example: `DETECTOR_NAMES= CCD_;`

Keyword: ***dname*_DETECTOR_DIMENSIONS**
Syntax: *dname*_DETECTOR_DIMENSIONS= *dim$_1$ dim$_2$;*
Type: Set of 2 integers
Description: The pixel dimensions of the detector image with *dim$_1$* being the dimension of the fastest varying direction.
Example: `CCD_DETECTOR_DIMENSIONS= 1024 1024;`

Keyword: ***dname*_DETECTOR_SIZE**
Syntax: *dname*_DETECTOR_SIZE= *size$_1$ size$_2$;*
Type: Set of 2 numbers
Description: The nominal size of the detector in millimeters. *size$_1$* is the size in the fastest varying direction.
Example: `CCD_DETECTOR_SIZE= 92.1600 92.1600;`

Keyword: ***dname*_DETECTOR_VECTORS**
Syntax: *dname*_DETECTOR_VECTORS= *p$_{1x}$ p$_{1y}$ p$_{1z}$ p$_{2x}$ p$_{2y}$ p$_{2z}$;*
Type: Set of 6 numbers
Description: The coordinates of two vectors *p$_1$ and p$_2$,* which relate pixel directions 1 & 2 to laboratory coordinates. For example, if pixel direction 1 is along Y and pixel direction 2 is antiparallel to X, the values would be `0 1 0 -1 0 0.`
Example:
```
CCD_DETECTOR_VECTORS= 1.0000 0.0000 0.0000 0.0000
1.0000 0.0000;
```

Keyword: ***dname*_NONUNF_TYPE**
Syntax: *dname*_NONUNF_TYPE= *nonunf_type;*
Type: String
Possible values: `None | Simple_mask`
Description: Specifies the type of non-uniformity correction to apply to images.
`None` Non-uniformity information is not provided.
`Simple_mask` Use a special non-uniformity image that has "bad pixels" set to zero and "good pixels" set to non-zero. Set the "bad" pixels in the image being processed to zero, according to the non-uniformity image.
Example: `See examples under` *dname*_NONUNF_INFO

Keyword: ***dname*_NONUNF_INFO**
Syntax: ***dname*_NONUNF_INFO**= *info;*
Type: String
Description: Provides information on non-uniformity correction which is complementary to the option specified by *dname*_NONUNF_TYPE keyword. Generally, the *info* string is a fully qualified path of the non-uniformity information image file. There are also two reserved *info* string values:

`none` no non-uniformity information provided. If, however, *dname*_NONUNF_TYPE is `Simple_mask`, use the first image in the scan as a non-uniformity image.

`FirstScanImage` Use the first image in the scan as a non-uniformity image.

Examples:

```
CCD_NONUNF_TYPE= None;
CCD_NONUNF_INFO= None;

CCD_NONUNF_TYPE= Simple_mask;
CCD_NONUNF_INFO= FirstScanImage;

RX_NONUNF_TYPE= Simple_mask;
RX_NONUNF_INFO= C:\Data\Lyso\mask.osc;
```

Keyword: **`BitmapSize`**
Syntax: `BitmapSize`= *n*;
Type: Integer
Description: The size of the embedded mask bitmap in bytes.
Remarks: The presence of this keyword indicates that the image is immediately followed by mask bitmap(s). Please refer to Appendix B for the mask bitmap description.
Example: `BitmapSize`= 1024;

Keyword:	**`BitmapType`**
Syntax:	`BitmapType` = *type*;
Type:	String
Possible values:	`BitmapRLE`
Description:	The type of the embedded bitmap.
Remarks:	The size of the bitmap should be given by `BitmapSize` . Please refer to Appendix B for the mask bitmap description.
Example:	`BitmapType = BitmapRLE;`

Keyword:	***dname*_SPATIAL_DISTORTION_TYPE**
Syntax:	*dname*_SPATIAL_DISTORTION_TYPE= *type;*
Type:	String
Possible values:	`Simple_spatial`
Description:	Specifies the type of spatial-distortion correction to apply to images when converting from millimeters to pixels and the reverse. With type `Simple_spatial` a simple linear scaling of pixels is performed to convert to millimeters. Other possibilities may be added in the future.
Example:	`See examples under` *dname*_SPATIAL_DISTORTION_INFO

Keyword:	***dname*_SPATIAL_DISTORTION_INFO**
Syntax:	*dname*_SPATIAL_DISTORTION_INFO= *bc1bc2 ps1 ps2;*
Type:	Set of 4 numbers
Description:	Provides information on spatial distortion correction, which is complementary to *dname*_SPATIAL_DISTORTION_TYPE. If *dname*_SPATIAL_DISTORTION_TYPE is `Simple_spatial`, then the four values specify the primary beam center in pixels and the pixel sizes in millimeters for the fast and slow varying pixel directions in the image.
Examples:	`CCD_SPATIAL_DISTORTION_TYPE= Simple_spatial;` `CCD_SPATIAL_DISTORTION_INFO= 515.8761 514.5211 0.0900 0.0900;`

Keyword:	***dname*_SPATIAL_DISTORTION_VECTORS**
Syntax:	*dname*_SPATIAL_DISTORTION_VECTORS= a_{1x} a_{1y} a_{2x} a_{2y}*;*
Type:	Set of 4 numbers
Description:	Provides elements of a 2 by 2 matrix that relates the image pixel directions (1 - fast, 2 - slow) to the world directions (X, Y).
Example:	`CCD_SPATIAL_DISTORTION_VECTORS=0.0000 -1.0000 1.0000 0.0000;`

3.7.1 CCD detector keywords

The following keywords are related to CCD detectors only.

Keyword: ***dname*_DETECTOR_ADC_OFFSET**
Syntax: *dname*_DETECTOR_ADC_OFFSET= *n;*
Type: Integer
Description: The ADC offset used by the detector.
Example: `CCD`_DETECTOR_ADC_OFFSET= 217;

Keyword: ***dname*_DETECTOR_IDENTIFICATION**
Syntax: *dname*_DETECTOR_IDENTIFICATION= *ID;*
Type: String
Description: The ID of the detector, which may contain information about the type and serial number of the detector.
Example: `CCD`_DETECTOR_IDENTIFICATION= MSC_MED_SATURN92;

Keyword: ***dname*_DETECTOR_OPTIONS**
Syntax: *dname*_DETECTOR_OPTIONS= $option_1$*:*$state_1$ $option_2$*:*$state_2$ $option_3$*:*$state_2$ *...* $option_n$*:*$state_n$*;*
Type: Set of pairs of strings, separated by colon signs.
Possible values:

Option	State
`dezingermode`	`on	off`
`dezingertype`	`average	sum`
`imagekind`	`1	2`
`transform`	`on	off`
`trigger`	`on	off`
`simulator`	`realtime	immediate`
`subtractclosed`	`on	off`

Description: Describes the options used by the detector when taking the image.
Remarks: Notice a colon character ':' between the *option* and *state* values. Images collected with older versions of the CCD detector server software may have options in the form: *-option: state*
Not all options listed here will appear in the actual header.
Other options and values may be added in the future.
Example:
```
CCD_DETECTOR_OPTIONS=imagekind:2 dezingermode:on
dezingertype:average subtractclosed:on transform:on
trigger:off;
```

Keyword: ***dname*_FIELD_OF_VIEW**
Syntax: *dname*_**FIELD_OF_VIEW** = *type;*
Type: String
Possible Values: Round | Square
Description: The shape of the field of view for the detector. Images collected with older versions of the CCD detector server may not contain this keyword. In such a case, the shape is implied to be Square.
Remarks: This keyword is a comment.
Example: `CCD_FIELD_OF_VIEW = Round;`

Keyword: ***dname*_DETECTOR_TEMPERATURE**
Syntax: *dname*_DETECTOR_TEMPERATURE= *temp;*
Type: String
Description: The temperature at which the detector is running, in degrees Celsius. If the detector does not have temperature reporting capabilities, or the detector reports a non-physical value, then the value may be 99999.0**.**
Remarks: This keyword is a comment.
Example: `CCD_DETECTOR_TEMPERATURE= -45.0 deg C;`

Keyword: ***dname*_TAPER_ORIENTATION**
Syntax: *dname*_TAPER_ORIENTATION= $orient_1$ ... $orient_n$*;*
Type: Set of strings
Possible values (for each string):
`+x+y | +x-y | -x+y | -x-y | +y+x | -y+x | +y-x | -y-x`
Description: The orientation of each taper in the detector. The values for the tapers relate the taper pixel directions (fast, slow) to the image pixel directions. For example, if a taper's fast pixel direction is parallel to the image's slow pixel direction and the taper's slow pixel direction is anti-parallel to the image's fast pixel direction, then the orientation would be `+y-x`.
Example: *dname*`_TAPER_ORIENTATION= +x+y –x-y +x+y –x-y;`

Keyword: ***dname*_UNBINNED_BEAM_POSITION**
Syntax: *dname*_UNBINNED_BEAM_POSITION= p_1 p_2*;*
Type: Set of 2 numbers
Description: The pixel position of the direct beam on an unbinned (1x1 binned) detector image.
Example: `CCD_UNBINNED_BEAM_POSITION= 1031.7522  1029.0422;`

Keyword: ***dname*_UNBINNED_DIMENSIONS**
Syntax: *dname*_UNBINNED_DIMENSIONS= dim_1 dim_2*;*
Type: Set of 2 Numbers

Description: The pixel dimensions of an unbinned (1x1 binned) detector image with dim_1 being the dimension of the fastest varying direction.
Example: `CCD_UNBINNED_DIMENSIONS= 2048 2048;`

Keyword: **DARK_PEDESTAL**
Syntax: DARK_PEDESTAL= *n;*
Type: Number
Description: The pedestal, or value added to all pixel intensities, after applying the dark current subtraction.
Example: `DARK_PEDESTAL= 20;`

3.8 Detector goniometer keywords

The following keywords are related to the detector goniometer.

Keyword: ***dname*_GONIO_NUM_VALUES**
Syntax: *dname*_GONIO_NUM_VALUES= *n;*
Type: Integer
Description: The number of detector goniometer axes.
Example: `CCD_GONIO_NUM_VALUES=6;`

Keyword: ***dname*_GONIO_NAMES**
Syntax: *dname*_GONIO_ NAMES= $name_1$ $name_2$ $name_3$ ... $name_n$*;*
Type: Set of *n* strings
Description: The names of the axes. The number of axes *n* is given by *dname*_GONIO_NUM_VALUES. Rotations must appear in the sequence first and must be in the order they will be applied. Translations must appear after all rotations. Translations can be in any order as their values will be summed to arrive at the detector translation vector.
Example: `CCD_GONIO_NAMES=RotAboutBeam 2Theta RotY XShift YShift Distance;`

Keyword: ***dname*_GONIO_VECTORS**
Syntax: *dname*_GONIO_VECTORS= g_{1x} g_{1y} g_{1z} g_{2x} g_{2y} g_{2z} g_{3x} g_{3y} g_{3z} ... g_{nx} g_{ny} g_{nz}*;*
Type: Set of 3 x *n* numbers
Description: The vectors describing goniometer rotation and translation axes. The number of vectors *n* and their order correspond to the number and order of names in *dname*_GONIO_ NAMES.

Example: `CCD_GONIO_VECTORS= 0.0000 0.0000 1.0000 1.0000 0.0000 0.0000 0.0000 1.0000 0.0000 1.0000 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 -1.0000;`

Keyword: ***dname*_GONIO_VALUES**
Syntax: *dname*_GONIO_VALUES= g_1 g_2 g_3 ... g_n;
Type: Set of *n* numbers
Description: The goniometer values for the angles and translations at the datum position. That is, the position of the detector at the start of the scan. The number of values *n* and their order correspond to the number and order of names in *dname*_GONIO_ NAMES.
Example: `CCD_GONIO_VALUES= -0.0336 40.0000 -0.0665 0.0375 -0.0281 45.0000;`

Keyword: ***dname*_GONIO_VALUES_MAX**
Syntax: *dname*_GONIO_VALUES_MAX= g_{1max} g_{2max} g_{3max} ... g_{nmax};
Type: Set of *n* numbers
Description: The maximum allowable goniometer values for the angles and translations. The number of values *n* and their order correspond to the number and order of names in *dname*_GONIO_ NAMES.
Example: `CCD_GONIO_VALUES_MAX= 1.0 90.0000 1.0 0.5 0.5 100.0000;`

Keyword: ***dname*_GONIO_VALUES_MIN**
Syntax: *dname*_GONIO_VALUES_MIN= g_{1min} g_{2min} g_{3min} ... g_{nmin};
Type: Set of *n* numbers
Description: The minimum allowable goniometer values for the angles and translations. The number of values *n* and their order correspond to the number and order of names in *dname*_GONIO_ NAMES.
Example: `CCD_GONIO_VALUES_MIN= -1.0 -10.0000 -1.0 -0.5 -0.5 35.0000;`

Keyword: ***dname*_GONIO_UNITS**
Syntax: *dname*_GONIO_UNITS= $units_1$ $units_2$ $units_3$ ... $units_n$;
Type: Set of *n* strings
Description: The units of the goniometer values. The number of strings *n* and their order correspond to the number and order of names in *dname*_GONIO_ NAMES. Rotation axes must have units `deg`, while translations must be in `mm`.
Example: `CCD_GONIO_UNITS=deg deg deg mm mm mm;`

3.9 Rotation keywords

The following keywords are related to the crystal rotation during the diffraction experiment. These keywords may describe either the whole scan rotation parameters or parameters of the image itself. In the first case, they will have a prefix **SCAN_**. Normally an image is part of a scan, so both prefixed and non-prefixed rotation keywords will appear in the header.

Keyword: **ROTATION**
Syntax: ROTATION= *start end incr time nosc ndark ndarkup dlim ndc ndcup;*
Type: Set of 10 numbers
Description: *start* and *end* - start and end angles in degrees relative to the CRYSTAL_GONIO_VALUES value of the scan rotation angle.
incr - angular increment per image.
time - exposure time in seconds
nosc - number of oscillations
ndark - number of dark images taken at the beginning of the rotation
ndarkup - dark image update interval
dlim - dark image change limit
ndc - number of DC offset images taken at the beginning of the rotation
ndcup - DC offset image update interval.

Remarks: If this keyword applies to a scan (SCAN_ROTATION), then *start* and *end* specify the overall start and end of the scan. If this keyword applies to a single image (ROTATION), *start* and *end* specify the rotation angles within a scan for the image.
Example:
```
SCAN_ROTATION= -90.000  69.000 0.500  20.000 0.000 0.000
0.000  100.000 0.000 0.000;
ROTATION= -90.0000  -89.5000 0.5000  20.0000 0.0000
0.0000 0.0000  100.0000 0.0000 0.0000;
```

Keyword: **ROTATION_VECTOR**
Syntax: ROTATION_VECTOR= r_x r_y r_z*;*
Type: Set of 3 numbers
Description: Specifies the rotation axis vector in world coordinates (see Appendix A) at the crystal goniometer datum position.
Example:
```
ROTATION_VECTOR= 1.0000 0.0000 0.0000;
SCAN_ROTATION_VECTOR= 1.000 0.000 0.000;
```

Keyword: **ROTATION_AXIS_NAME**
Syntax: ROTATION_AXIS_NAME= *name;*
Type: String
Description: Specifies the axis name. Usually the crystal is rotated around a crystal goniometer hardware axis and this name reminds which one was used.
Example:
```
ROTATION_AXIS_NAME= Omega;
SCAN_ROTATION_AXIS_NAME= Omega;
```

3.10 Scan keywords

In addition to the scan keywords derived from the **ROTATION** keywords (see section 3.9) there are a few keywords specific to a scan.

Keyword: **SCAN_SEQ INFO**
Syntax: SCAN_SEQ INFO= *seq_start seq_increment seq_reserved;*
Type: Set of 3 numbers
Description: *Seq_start* - the image sequence number for the image that begins at the *start* angle specified by the SCAN_ROTATION keyword.
Seq_increment should always be 1.
Seq_reserved is reserved.
Example: `SCAN_SEQ_INFO= 1 1 360`

Keyword: **SCAN_TEMPLATE**
Syntax: SCAN_TEMPLATE= *scan_filename_template;*
Type: String
Description: Provides an image file path with question marks that will be filled in with the image sequence number in order to get the image filename.
Examples:
```
SCAN_TEMPLATE=
D:\Data\reb\MS1749a\Images\MS1749a????.img;
SCAN_TEMPLATE= C:\data\lyso.???
SCAN_TEMPLATE= ../MyImage??.???
```

3.11 Overload correction keywords

The following keywords are related to the type of overload correction applied.

Keyword: **OVERLOAD_TYPE**
Syntax: OVERLOAD_TYPE= *type;*
Type: String
Possible values: `Time | Attenuator`
Description: Specifies the type of overload correction applied.
When *type* is `Time` a correction image of a shorter time is taken and the pixels above the overload threshold in the original image are replaced with the corresponding pixels in the correction image, appropriately scaled. When *type* is `Attenuator` a correction image of the same time is taken, but with the attenuator in the beam , and the pixels in the original image above the overload threshold are replaced with the corresponding pixels in the correction image, appropriately scaled. Other possibilities may be added in the future.
Example: `OVERLOAD_TYPE= Time;`

Keyword: **OVERLOAD_ATTENUATORFACTOR**
Syntax: OVERLOAD_ATTENUATORFACTOR= *factor;*
Type: Number
Description: Specifies the factor used when scaling the pixels in the overload correction image when OVERLOAD_TYPE is `Attenuator`.
Example: `OVERLOAD_ATTENUATORFACTOR= 0.1`

Keyword: **OVERLOAD_FACTOR**
Syntax: OVERLOAD_FACTOR= *factor;*
Type: Number
Description: Specifies the factor used when scaling the pixels in the overload correction image when OVERLOAD_TYPE is `Time`.
Example: `OVERLOAD_FACTOR= 5.000`

Keyword: **OVERLOAD_THRESHOLD**
Syntax: OVERLOAD_THRESHOLD= *threshold;*
Type: Number
Description: Specifies the threshold which triggers the overload correction. If a pixel in an image is above this value, then an overload correction is applied. It is not guaranteed that any pixels will be above this value after the overload correction is applied.
Example: `OVERLOAD_THRESHOLD= 55000`

3.12 Collision keywords

The following keywords are related to the collision avoidance feature in controlling software.

Keyword: **COLLISION_INFO**
Syntax: COLLISION_INFO= *filename;*
Type: String
Description: Specifies the fully qualified name of an XML file that contains the collision avoidance information.
Example: `COLLISION_INFO= C:\CC136\Collision_info.xml`

Keyword: **CRYSTAL_GONIO_COLLISION_OFFSET**
Syntax: CRYSTAL_GONIO_COLLISION_OFFSET= g_1 g_2 g_3 ... g_n;
Type: Set of *n* numbers
Description: Specifies the offsets (in degrees) that should be applied to the crystal goniometer axes positions before determining collision avoidance information. The number and order of the axes is the same as in CRYSTAL_GONIO_NAMES. The units are the same as in CRYSTAL_GONIO_UNITS (see section 3.6).
Example: `CRYSTAL_GONIO_COLLISION_OFFSET= 0.0000 0.0000 0.0000`

Keyword: ***dname*_GONIO_COLLISION_OFFSET**
Syntax: *dname*_GONIO_COLLISION_OFFSET= g_1 g_2 g_3 ... g_n;
Type: Set of *n* numbers
Description: Specifies the offsets that should be applied to the detector goniometer axes positions before determining collision avoidance information. The number and order of the axes is the same as in *dname*_GONIO_NAMES. The units are the same as in *dname*_GONIO_UNITS (see section 3.8).
Example: *dname*`_GONIO_COLLISION_OFFSET= 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000`

3.13 Optics keywords

The following keywords are related to the optics system.

Keyword: **OPTICS_COLLIMATOR**
Syntax: OPTICS_COLLIMATOR= *size;*
Type: String
Description: Describes the optics collimator size.
Remarks: This keyword is a comment.
Example: `OPTICS_COLLIMATOR= 0.3 x 0.3 double-pinhole;`

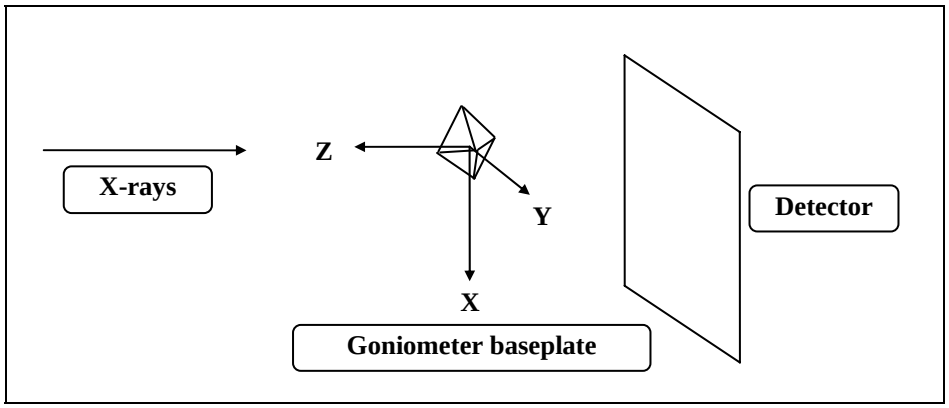
Keyword: **OPTICS_TYPE**
Syntax: OPTICS_TYPE= *type;*
Type: String
Description: Describes the type of the collimator optics
Remarks: This keyword is a comment.
Example: `OPTICS_TYPE= confocal;`

Appendix A

d*TREK coordinate system

The d*TREK default laboratory coordinate system shown in the next figure is defined as follows:

1. The origin of the laboratory coordinate system is at the crystal.
2. The **X** axis goes from the crystal toward the goniometer baseplate. This may be horizontal, vertical or neither.
3. The **Z** axis goes from the crystal toward the source.
4. The **Y** axis make a right-handed system after **X** and **Z** are chosen.
5. All positive hardware rotations are right-handed.
6. The crystal to detector distance is positive even though it is usually at a negative Z value.
7. The direct beam position on the detector in pixels is specified when the detector is perpendicular to the source (rotations are at zero) AND the detector translations in **X** and **Y** are zero. In the actual experiment, the detector may be rotated or translated away from this position.



d*TREK laboratory coordinate system. Note: The source *vector*, s_0, is along +Z and is *not shown* in the above figure.

Appendix B

Mask Bitmap Structure

Bitmaps of a mask are stored using a variation of the run-length encoding (RLE) scheme.

The traditional RLE encoding scheme is a simple form of compression and is based on the assumption that the data being compressed has large numbers of consecutive characters (or bytes), or runs, that contain the same value. Runs are converted into a code consisting of the character and the number of characters in the run (*i.e.*, the length of the run). The longer the run of consecutive characters, the greater the compression. RLE is a lossless compression scheme.

The variation of the RLE scheme used in the bitmap structure uses the fact the image of interest contains two types of pixel values; either zero, or not zero. This allows the type of value and the length of the run to be encoded in a single value. The most significant bit (MSB) of the value is used to indicate if the value is zero or non-zero. If the bit is set, the value of the repeated pixel is non-zero, and if the value is not set, the value of the repeated pixel is zero. The remainder of the bits in the value is used to specify the length of the run. Note that this is a lossy compression scheme, since only zero/non-zero pixel value information is retained.

Format of Encoded Bitmap

The encoded bitmap is stored as an array of unsigned short integers, stored in a big-endian fashion. The first two elements of the array, composing 4 bytes, is a marker composed of the characters “BRLE”. The encoded bitmap values immediately follow the marker. The maximum run length in a value using this scheme is 32767. Runs of longer than 32767 are broken up into multiple values.

The embedded bitmap is placed immediately following the image data.

Appendix C

Image Data Compression

The presence of the RAXIS_COMPRESSION_RATIO keyword in the headers indicates that the individual pixel values may be compressed using the R-AXIS scheme. The value of the keyword indicates the multiplicative factor to be applied to those pixels (taken as unsigned 2-byte integers) with values greater than 32767. If the RAXIS_COMPRESSION_RATIO is not present, then the pixels do not have any compression applied, and are just treated as unsigned 2-byte integers.

For images with the RAXIS_COMPRESSION_RATIO keyword defined, they are interpreted as:

```
unsigned short          pixel;
long                    corrected_value;

if ( pixel > 0x7fff )
      corrected_value = (long)(pixel&0x7fff)*ratio;
else
      corrected_value = pixel;
```

where ratio is the value of the RAXIS_COMPRESSION_RATIO keyword (typically 8 or 32).

Note that the COMPRESSION keyword refers to the compression scheme applied to the entire image as a whole (i.e., Lempel-Zev or Huffman) and not the individual pixels. We currently do not write out any images that use compression on the images as a whole.

Also, note that a dataset can have some images that contain the RAXIS_COMPRESSION_RATIO keyword and some that don't contain the keyword. The RAXIS_COMPRESSION_RATIO keyword has to be handled on an image-by-image basis, and not on a dataset-by-dataset basis.

Appendix D

Example of a d*TREK Image Header.

```
{
HEADER_BYTES= 2048;
TYPE=mad;
SIZE1=512;
SIZE2=512;
COMMENT=Header edited by dtheaderedit;
SCAN_TITLE=start end inc time nOsc nDark nDup nDlim nDC nDCup;
SCAN_ROTATION=0.0 12.0 0.2 4 0 1 0 100 1 0;
SCAN_ROTATION_AXIS_NAME=omega;
SCAN_ROTATION_VECTOR=1.0 0.0 0.0;
SCAN_TEMPLATE=protein???.img;
SCAN_SEQ_INFO=1 1 1;
ROTATION=0.0 0.2 0.2 4 0 1 0 100 1 0;
ROTATION_VECTOR=1.0 0.0 0.0;
ROTATION_AXIS_NAME=Omega;
CRYSTAL_UNIT_CELL=82.34 88.29 103.65 90.00 90.00 90.00;
CRYSTAL_MOSAICITY=0.145;
CRYSTAL_DESCRIPTION=Test of dtpredict;
CRYSTAL_SPACEGROUP=19;
CRYSTAL_REFINE_FLAGS=0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
CRYSTAL_ORIENT_ANGLES=10.04 -26.37 31.81;
CRYSTAL_ORIENT_VECTORS=1 0 0 0 1 0 0 0 1;
SOURCE_VECTORS=0.0 0.0 1.0 0 1 0 1 0 0;
SOURCE_POLARZ=0.5 1.0 0.0 0.0;
SOURCE_SPECTRAL_DISPERSION=0.0002 0.0002;
SOURCE_SIZE=0.0 0.0 0.0 0.0;
SOURCE_CROSSFIRE=0.0 0.0 0.0 0.0;
SOURCE_WAVELENGTH=1 1.54178;
SOURCE_REFINE_FLAGS=0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
DETECTOR_NUMBER=1;
DETECTOR_NAMES=D0_;
D0_NONUNF_TYPE=Simple_mask;
D0_NONUNF_INFO=FirstScanImage;
D0_SPATIAL_DISTORTION_TYPE=Simple_spatial;
D0_SPATIAL_DISTORTION_INFO=256.8761  256.5211 0.0900 0.0900;
D0_DETECTOR_DIMENSIONS=512 512;
D0_DETECTOR_SIZE=50.0 50.0;
D0_DETECTOR_VECTORS=1 0 0 0 1 0;
D0_DETECTOR_DESCRIPTION=ANL-SBC gold detector single module;
D0_DETECTOR_REFINE_FLAGS=0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
D0_GONIO_NUM_VALUES=6;
D0_GONIO_NAMES=RotX RotY RotZ TransX TransY TransZ;
D0_GONIO_UNITS=deg deg deg mm mm mm;
D0_GONIO_VECTORS=1 0 0 0 1 0 0 0 1 1 0 0 0 1 0 0 0 -1;
D0_GONIO_VALUES=0.0 0.0 0.0 0.0 0.0 102.3;
D0_GONIO_DESCRIPTION=A simple detector goniostat;
CRYSTAL_GONIO_NUM_VALUES=3;
CRYSTAL_GONIO_NAMES=Omega Chi Phi;
CRYSTAL_GONIO_UNITS=deg deg deg;
CRYSTAL_GONIO_VECTORS=1 0 0 0 1 0 1 0 0;
```

```
CRYSTAL_GONIO_VALUES=0.0 0.0 0.0;
CRYSTAL_GONIO_DESCRIPTION=A 3-circle eulerian crystal goniostat;
FILENAME=proteinX01.img;
BYTE_ORDER=big_endian;
Data_type=short int;
COMPRESSION=None;
}
^L
(Rest of header is padding to [in this case] 2048 bytes total size)
```