

CU-QMW-TN-0011

Date: 24 May 2016

Issue: 1

Rev. : 7

Page: i

Syntax for ASCII and Binary Data Files for Use by QSAS

A.J. Allen
Imperial College

<http://www.sp.ph.ic.ac.uk/csc-web/csc.html>
csc-support-dl@imperial.ac.uk

Document Status Sheet			
1. Document Title: QSAS Data File Syntax			
2. Document Reference Number: CU-QMW-TN-0011			
3. Issue	4. Revision	5. Date	6. Reason for Change
0	0	8 July 1996	Draft
0	1	9 October 1996	Add distinct file types.
0	2	16 October 1996	SJS/DHB comments.
1	0	30 October 1996	First release.
1	1	27 November 1996	tidying up.
1	2	24 February 1997	Extensions to formats.
1	3	3 June 1998	formalised for standalone Qtran utility
1	4	29 April 1999	Added Free Time Format specification
1	5	16 April 2002	minor corrections to free time format
1	6	25 September 2013	major changes and extension of free time format (Qie now uses Dvos)
1	7	24 May 2016	minor additions and .qat file explanation

Contents

1	Introduction	1
1.1	File Formats	1
2	File Contents	1
2.1	Time Series Data	2

2.2	Data Series	2
3	File Headers	2
3.1	Metadata Syntax	2
3.1.1	Global attributes	4
3.1.2	Metadata Describing Variables	4
4	Tabular Data Files - “.qft” Files	7
4.1	Data Records	7
4.2	Headers for Tabular files	7
5	Delimited Data Files	7
5.1	Data Records	7
5.2	Headers for Delimited files	8
6	Helper Files, “.qat”	8
6.1	Overview	8
6.2	Extra Syntax	8
6.3	post Processing	9
7	Free Time Format Specification	9
7.1	Overview	9
7.2	Key Strings	10
7.2.1	Date Key Strings	10
7.2.2	Time Key Strings	11
7.2.3	Pad Characters	12
7.3	Examples	12
8	Reading Other ASCII Files	15
8.1	Reading the File	16
8.2	Providing Metadata	16

1 Introduction

This document provides information on the syntax for ASCII data files that can be read and written by the Science Analysis System QSAS. The use of header files make it possible for QSAS to read many ASCII flat file types from widely different missions, with the header describing sufficient information for the data to be machine parseable. The header syntax is described fully here. The same routines also read and write standard NASA CDF format and ESA CEF format files.

The software utility Qtran will translate between these formats.

1.1 File Formats

All data files are record oriented. They have a sequence of records each with the same variables in the same order. Variables that are multi-dimensional take the natural C ordering, and one entry is required for each element. In time series data the records are ordered on the monotonic increasing time variable.

Two distinct file formats are covered.

1. Tabular data files are ASCII files with records of equal length and fixed column start positions for each entry. These files are identified by the extension “.qft”.
2. Delimited data files are ASCII files with variable length records, with a delimiter separating each entry. These files are identified by the extension “.qfd”.

2 File Contents

Files must have a header associated with them that describes the data and formatting within the file (see below). This header may either be at the top of the file itself (attached) or in a separate file (detached). Detached header files are identified by the extension “.qfh”. A detached header may provide formatting information for a series of files which all have the same record structure.

If the header is attached, the data records must be immediately preceded by a line:

```
Start_data = xxx
```

where xxx is an integer specifying the number of records to follow. If the number of data records is unknown, this value may be set to zero. This number is used by QSAS for verifying the expected number of records; if not present, zero or non-numeric the actual number of records will be counted before space is allocated. If the number of records exceeds that specified a warning is issued and processing stops to avoid memory exceptions.

An exclamation mark is used as a comment marker, and all input to the right of this marker up to the newline character is ignored on input.

Blank lines (where the first character is the newline character, or contain only space characters or tabs) are ignored.

Text values may be quoted or unquoted. Note that in space separated files spaces are not allowed within text data types even if the text is quoted.

If a file does not have an attached header, the directory is searched for a detached header (*.qfh). The first header file located is used. This means that the appropriate header must be available and unique within a data directory. Thus no more than one header and its associated

data files should reside in a directory and the header file can take any name with the extension “.qfh”.

2.1 Time Series Data

Time series data have a date/time entry associated with each record. Each other entry in the record is associated with the time in that record. The first date/time (epoch) variable detected in each file is taken as the timeline for that file.

On output, time is represented as a text string in the ISO format adopted by CSDS (see DS-QMW-TN-0003 for detailed specification). On input, time can either be in this same ISO format or as a Free Format Time. Information in the file header containing such a Free Format Time variable describes the components of the time string in the data to convert almost any time specification into the standard cdf (and/or ISO) format. Free Format Time is described in detail below.

Free time formats are supported for input but not output.

2.2 Data Series

Data series have no time entry in a record. If more than one data series is written to the same file then each data series must contain the same number of records and any one-to-one dependency (or otherwise) between variables in a record must be explicitly explained within the file metadata.

3 File Headers

The file header contains formatting information about the file, and metadata that describes the data. A header may be the start of a data file or in a separate (detached) file. A header may apply to a family of data files that contain data in the same format, such as would result from dividing data into files covering distinct time intervals.

The first line in a header may be a comment line starting ! and containing the data file extension (eg “.qft” etc.) followed by a QIE version number. If this information is available it will be used to test on QIE version compatibilities.

On read QSAS will use the file extension as a first guess as to data file type, nevertheless, the presence in the file of a ‘File_type’ parameter will supercede it.

QSAS will always write a file in the format specified by the options in the export window, and will place a commented line containing the data file extension and QSAS version at the start of the header.

3.1 Metadata Syntax

All header entries take the format

parameter = value

where “parameter” and “value” are strings of printable characters. The equals sign may be bedded in space characters, and white space is stripped before the “parameter” token. White space is valid within the “value” string which is terminated by the newline character or comment marker, ‘!’. Leading and trailing white space is removed from the “value” token. In the

case that the “value” token contains only white space a single space character is returned as the token. Parameters are not case sensitive.

Metadata is divided into global information that applies to the whole file and information that describes each variable and its formatting within a record.

Blocks of information describing the variables start with a line

`Start_variable = name`

and end with the line

`End_variable = name`

where the value “name” is the name to be used for the variable. The variable blocks are described in the section “Metadata Describing Variables”. These blocks are required for variable description and header files are not valid without one describing each variable.

The file parameters below are optional for all supported file types, and if omitted QSAS will adopt a default as specified for each item below. Extra formatting parameters needed for the specific file types are listed in the section dealing with that file type.

File_name This specifies the name of the file. No default value is assumed. It should not include the path, but does include the file type extension. If this is a detached header it is the name and extension of the header. Good practice would suggest that the header and the data files to which it applies share the same file name stem up to the non-unique part of the name.

File_type This specifies the format type of the file. A default value is taken according to the file name extension. Valid types are ‘t’ for tabular data files and ‘d’ for delimited, and correspond to the file name extensions ‘.qft’ and ‘.qfd’ respectively.

Attribute_delimiter This specifies the character to be used in the header itself to separate each entry in a multi-entry variable attribute. If no entry is supplied a comma is assumed since space characters are frequently valid within attribute strings.

Start_meta This starts a block of metadata supplying the entries associated with a global attribute. This block is closed by an **End_meta** parameter. They are described more fully below. The value associated with these parameters is the name of the global attribute. No defaults are provided, and no global attributes need be supplied. QSAS will always write a “Parents” attribute. This facility is provided for CSDS and CDF compatibility.

Comment_marker The value associated with this parameter is a character that starts a comment line in a data file or header. This is in addition to empty lines and lines starting with an exclamation mark, which are always ignored as comment lines. Any line which starts with the character token specified in “Comment_marker” will be ignored whether it be in the header or between records.

Start_after The value associated with this parameter is a string token contained in the line preceding the start of the data records. For files with attached headers, all lines will be ignored after the “Start_data” line up to and including a line containing the “Start_after” token. For files with detached headers, all lines in the data file up to and including the line containing with the “Start_after” token will be ignored. This is provided to aid interfacing to packages that add their own header to data files

One further control sequence is available, but which does not take the ‘parameter = value’ syntax:

& The numeric integer value immediately following this parameter is the number of subsequent lines to ignore. It takes effect immediately and may be repeated as needed. It is provided to aid interfacing to packages that add their own header to data files.

3.1.1 Global attributes

All the global metadata available within QSAS may be written to a header, but the default is to write only a “Parents” entry. The Global attribute block starts with a line

Start_meta = name

and ends with the line

End_meta = name

where the value “name” is the name of the global attribute. The global metadata block contains the following entries:

Number_of_entries The value associated with this parameter is the number of attribute entries provided for this attribute. It must be followed by the same number of “Entry” parameters within in the attribute block.

Entry The value associated with this entry is the next entry in the global attribute. Space characters are valid within a text entry. This parameter may be repeated, with each successive value providing the next entry in the attribute.

Data_type The “Data_type” of a global attribute is used to convert the ascii text entry in each “Entry” into the appropriate data type in the data structure. The default value for each global attribute independently is text.

3.1.2 Metadata Describing Variables

The only essential blocks of metadata are those describing the variables. These give metadata and formatting information specific to the named variable. Variables *must* appear in the same order within a record that the variable entries occur within the header. Each block of variable metadata takes the form,

Start_variable = name

parameter = value

⋮ ⋮

End_variable = name

Essential parameters for each file type are listed in the section dealing with that file type. The following parameters are essential for all file types:

Data_type This identifies the data type and is necessary for conversion from the ascii entry.

This parameter may also be called ‘Value_type’). Allowed values are

epoch or ISO_time

ISO_time_range

double or float

int

byte
char

Sizes This is essential for any variable that has more than one element, such as arrays and vectors. The value string must comprise as many ('Attribute_delimiter' separated) integer values as there are dimensions in the variable (in the CDF sense) with the number giving the size of the array in that dimension. Thus an 8 by 54 array would have the entry
Sizes = 8,54.

It is not required for scalars.

Data The CDF concept of a variable that is fixed for all records is supported for flat files. Data for these "non-record-varying" variables must be supplied within the header variable metadata segment, and no entry is then allowed in the data records. The presence of a parameter "Data" will be taken to indicate that this is a non-record-varying variable. The value(s) associated with this parameter are the data for that variable. These are particularly useful for label variables. They are separated by the "Attribute_delimiter" as they are specified within the metadata segment.

Time_format In time series data the time variable must have this parameter to identify the time format used. At present there are two accepted values:

ISO indicates that the time variable is in the standard ISO string format. Files output by QSAS and Qtran always have this Time_format.

FREE_TIME_FORMAT indicates that the time variable in the file is held in a string which is not in ISO form but which can be parsed by QSAS. The parsing information is passed in the parameter "TIME_FORMAT_STRING", augmented other optional parameters (see later section).

TIME_FORMAT_STRING This parameter is mandatory for time variables with

Time_format = FREE_TIME_FORMAT

and describes the time field within the file in terms of key strings such as "YYYY" or "MON". For example, the ISO format could be written here as

TIME_FORMAT_STRING = YYYY-MO-DD HH:MM:SS.sss

A full list and description of key strings is given in a later section. Note that all specifications of both parameters and values are CASE SENSITIVE and POSITION SENSITIVE, and unexpected results will be obtained if the specification does not match exactly that used within the data file. Note also that leading and trailing white spaces are stripped from the 'TIME_FORMAT_STRING'.

YY_offset In the case of a 'FREE_TIME_FORMAT' time format this parameter specifies a year to be added to the year field (needed if two character year format is used). Thus, if the year 1996 is recorded in the file as '96', the YY_offset must be set to '1900'. No default is provided. Files with 'YY' format time strings that cross century boundaries must be broken at the century boundary, and merged inside QSAS.

MM_offset In the case of a 'FREE_TIME_FORMAT' time format this parameter specifies a month to be added to the month field.

DD_offset In the case of a 'FREE_TIME_FORMAT' time format this parameter specifies a day to be added to the day or day-of-year fields.

SS_offset In the case of a 'FREE_TIME_FORMAT' time format this parameter specifies a seconds value as a double to be added to the seconds field.

TIME_START In the case of a 'FREE_TIME_FORMAT' time format this parameter specifies a date/time epoch to add to all resulting date/times and takes ISO text time format.

TIME_COMPONENT In the case of a 'FREE_TIME_FORMAT' time format where the date/time fields are spread over multiple variables this parameter is used to chain the variables together. The first variable in the record that forms part of the date/time must have a TIME_COMPONENT parameter whose value is the name of the next component variable in the record. This in turn must have a TIME_COMPONENT parameter pointing to the next, and so on. The last component variable does not take a TIME_COMPONENT parameter and the chain ends.

All_records_time In the case of a 'FREE_TIME_FORMAT' time format, this is an OPTIONAL parameter specifying a component(s) of time which is the same for all records and not present in the data file itself. For example, for short segments of data the year and date may not be repeated in each record but simply given once in a header (or filename) by some data archival systems. In such cases an entry of, say,
All_records_time = 1984-JAN
supplies the year and month to be used for each record. If this parameter is present, the following parameter 'All_records_format' must also be present to specify its format.

All_records_format This parameter specifies the format of the 'All_records_time' string using the same syntax as the 'TIME_FORMAT_STRING'. For the example above a suitable entry would be
All_records_format = YYYY-MON

Further desirable parameters are the QSAS subset of the CSDS attributes. These will always be written by QSAS if available.

- UNITS
- FIELDNAM
- Frame
- SI_conversion

Other variable attributes may be written within the variable block. Certain attributes will default to being the same data type as the variable (*eg* SCALEMIN and SCALEMAX) but most will default to text attributes.

4 Tabular Data Files - “.qft” Files

4.1 Data Records

Data is record oriented with a newline character denoting the end of each record. All records in a file are of the same fixed width with each variable taking a fixed column start position and column maximum width. This allows data to be tabulated for ease of reading and ingestion into other packages. Each data record must contain an entry for each variable in the order specified by the header descriptors. Missing entries must be padded by space characters or a fill value to ensure correct ordered reading. Tab characters are treated as single characters and must not be used in formatting tables of data (use delimited file syntax with tab as the delimiter).

Each variable may be multi-dimensional with the number of entries per variable being the product of the dimensionalities. Thus a velocity vector has three entries and a two by two array has four entries.

The ordering for arrays within a record is the natural ‘C’ ordering, that is, the last index varies the fastest.

4.2 Headers for Tabular files

In addition to the global metadata and specific variable metadata the header must specify positions for each variable in a record. The column start position of the first element of each variable in a record, together with the (constant for each element in a variable) field width of each entry must be specified as part of each variable description. These have syntax

```
column_start = nn
```

```
column_width = mm
```

where *nn* and *mm* are numbers and the first column position is counted as zero.

Columns of data can be skipped in tabular files by omitting the corresponding variable description. The column_start and column_width are sufficient to locate variables that are required, and entries interspersed between specified variables in each record may be ignored.

5 Delimited Data Files

5.1 Data Records

Data is record oriented with a character (default newline) denoting the end of each record. The entries for each variable in a record must appear in the order specified by the header descriptors. Each entry within a record must be separated from its neighbours by a delimiter (specified in the header). Missing entries should be padded by a fill value. If the delimiter character is a space, then multiple spaces are treated as a single delimiter, otherwise repeated delimiters indicate missing entries.

Each variable may be multi-dimensional with the number of entries per variable being the product of the dimensionalities. Thus a velocity vector has three entries and a two by two array has four entries.

The ordering for arrays within a record is the natural ‘C’ ordering, that is, the last index varies the fastest.

Any printable character may be used as an entry delimiter except for numeric characters and format specifiers 'e', 'f' and 'g', the decimal point, '.' and the operators '+' and '-'. Any character that appears within a formatted number or text data entry will also result in a read failure and must be avoided. A space character is permitted as a delimiter.

Caution must be exercised when writing character data to ensure that a delimiter is available and precluded as a valid data entry. White space will often be a valid data character in text data.

A comma or space character provide for most ease of readability, and are recommended for numerical data, but tab delimited data are also common.

5.2 Headers for Delimited files

In addition to the global metadata and specific variable metadata the header must also specify the delimiter to be used to separate data entries. This delimiter is the same for all variables (but may differ from that used in the header for attributes), and is placed with the global metadata.

It has syntax

```
Data_delimiter = x
```

where x is any printable character. The space character is specified by

```
Data_delimiter =
```

```
or Data_delimiter = ' '
```

The default character if no Data_delimiter line is present is a comma.

6 Helper Files, “.qat”

6.1 Overview

Helper files have the same syntax as detached header files, but are read in addition to headers. Any file ending “.qat” found in the same directory as a data file will be read in addition to the data file header information and specified detached header, and will take effect before the data is read in.

They serve the role of modifying the header information and supplying missing metadata. If the same attribute information is found in the data file and the helper file, then the helper file content takes precedence to allow modification of data header content, so that badly formed metadata in data files can be corrected.

In addition to using the syntax of regular header files, the helper files will accept extra instructions introduced to allow mission specific data structure modifications.

6.2 Extra Syntax

Use_with It can be specified as metadata within a non-record varying variable in the .qat file.

This causes the containing variable to also be read when the named variable is read from the file. If the name is “Always” then this variable is always read in. It thus permits the containing variable to be attached as a cross reference to another variable, or just loaded as though it were in the data file. Thus

```
START_VARIABLE = Energy_Table
```

```
VALUE_TYPE = FLOAT
```

```
sizes = 6
```

Data = 39665, 13155.1, 4212, 2307, 981.7, 433.3
SI_Conversion = "1.60217646E-19>J"
Units = "eV"
Use_With = "NO_OF_COUNTS"
END_VARIABLE = Energy_Table
will cause the variable Energy_table to be read in if the variable NO_OF_COUNTS is present.

6.3 post Processing

There are extra capabilities introduced for Rosetta PDS file syntax handling that apply to post processing the data objects after ingestion. The Rosetta Plasma Consortium (RPC) quicklook data arrays with dimensionality above one are spread over a succession of records, and need re-assembly into arrays. These are recorded here for completeness, but it is not expected that they will be used outside of Rosetta.

DEPEND_i ALIAS identifies the variable holding the bin index for a variable spread over records. The *i* can be 1, 2, 3 etc, and the DEPEND_i will be constructed from this variable when the data has been re-formatted as an array.

BINS provides the values to be used for a Depend.i variable replacing the index.

Fixed_Sizes_i indicates that the dimension *i* should have the size specified, even if the first record does not fill it. The data will be averaged out over the dimension. This allows for instruments that change dimension sizes.

Post_Run_Qtran passes a flag that allows Qtran to perform instrument specific correction. It is specific to Qtran internals and has no purpose without corresponding code added in Qtran.

VECTOR_COMP_i Identifies which scalar variable is to be treated as the *i* component of a vector. VECTOR_COMP_1 = this, and the names of the other variables as, e.g. VECTOR_COMP_2 = "RPC_B_y" etc are used in the variable for giving first component. The other components should not have these attributes specified as they will be located automatically.

VECTOR_NAME Provides the name to be used for the assembled vector when using VECTOR_COMP_i.

7 Free Time Format Specification

7.1 Overview

Time variables of 'Time_format = FREE_TIME_FORMAT' enable most foreign time series files to be imported into QSAS, or translated by Qtran into a standard format (e.g., qft or qfd with ISO time strings, or cdf). The interpretation and conversion of the time field in each data record is accomplished by specifying its components and format in the 'TIME_FORMAT_STRING'

metadata for that variable. This string is made up of 'key strings' to match EXACTLY the entry in the data record. The **last** occurrence of any particular 'key string' is used.

There is, of course, a restriction that the set of 'key strings' plus variable parameters (see above) can provide a complete date specification of the 'date/time' field, but any missing time fields will be treated as zero. A badly formed date/time string will appear in Qtran/QSAS as '2000-01-01 00:00:00.00'.

There is no restriction that the time field in the record be contained within a single variable. However, if the date/time field is split amongst variables the 'TIME_COMPONENT' parameter (see above), must be used, and the first component variable will appear as the complete time variable (the others will appear badly formed and should be ignored). Variables comprising the date/time value need not be consecutive nor in any particular order, but the order of the variable blocks in the header, as always, must reflect their order in the records.

The basic algorithm simply parses the 'TIME_FORMAT_STRING' and compiles the location of each date/time element. This information is then used to extract the values from the time entries in each record and compute the resulting epoch value. As described in the section on Variable Metadata, any elements, such as the year, month, seconds offset, etc., which are common to all records but not present in each record can be specified using offset values, or collectively using a 'TIME_START' flag. The same may also be achieved by using the 'All_records.time' and 'All_records.format' strings which together are applied to all records using the free time syntax.

Once all components have been identified they are combined into a DvTime object within QSAS/Qtran, which stores the value split into a Julian Date as an integer and seconds as a double. The DvTime library is valid for all dates both AD and BCE but ignores leap seconds, and can usually retain nanosecond accuracy. DvTime is also available as a stand-alone library, and as source under an extended GPL licence.

7.2 Key Strings

The following provides an exhaustive list of Key Strings used in Free Time Format input. They need not appear in this order.

7.2.1 Date Key Strings

YYYY 4-digit year specification.

YY 2-digit year. If this format is used the *YY_offset* or *TIME_START* parameter must also be set. This format should be avoided where possible as it is intrinsically not year 2000 compliant.

Doy 3-digit day of the year. Values obtained are converted into month and day in the given year, with leap years treated correctly. Note the lower case 'oy' which avoids conflict with year key strings.

MON 3-character month string. The values expected are: "JAN", "FEB" or "FEV", "MAR", "APR" or "AVR", "MAY" or "MAI", "JUN", "JUL" or "JLY", "AUG" or "AOU", "SEP", "OCT", "NOV", "DEC", but values in the data file are case-insensitive.

MONT or MONTH 4 or 5-character month string. The optional fifth character is ignored, so the leading 4 digits expected are: “JANU” or “JANV”, “FEBR” or “FEVR”, “MARC” or “MARS”, “APRI” or “AVRI”, “MAY ” or “MAI ”, “JUNE” or “JUIN”, “JULY” or “JUIL”, “AUGU” or “AOUT”, “SEPT”, “OCTO”, “NOVE”, “DECE”, but values in the data file are case-insensitive.

MO 2-digit numeric month.

DD 2-digit numeric day.

7.2.2 Time Key Strings

Time key strings can have any length. The descriptions below show the repeat character with an overbar, e.g., \overline{H} means H, HH, HHH, HHHH, etc. are all allowed. Whole and fractional parts may be specified separately, or the floating point value read in its entirety. Upper case letters are read as floats and may contain a decimal point, or may just be used for the whole part of the number. Fractional parts use lower case, and are assumed to contain no decimal point and are multiplied by $10^{-\text{field width}}$. Values are converted from the input string using the C++ library functions `strtof`.

\overline{H} n-digit whole or decimal hour ($n \geq 1$). Typically ‘HH’ but could be ‘HHHHH’ to read in ‘17.37’ hours.

\overline{MM} or \overline{MI} n-digit whole or decimal minutes ($n \geq 2$). ‘MMMM or MIII’ would read in ‘28.4’ minutes.

\overline{S} n-digit whole or decimal seconds ($n \geq 1$).

\overline{L} or \overline{MC} n-digit whole or decimal milliseconds ($n \geq 1$), e.g. LLL or MCC to read 2.1 msec.

\overline{U} n-digit whole or decimal microseconds ($n \geq 1$), e.g. UUU to read 123 microsec.

\overline{N} n-digit whole or decimal nanoseconds ($n \geq 1$), e.g. NNNN to read 1.34 nanosec.

\overline{d} n-digit fractional day ($n \geq 1$). ‘ddd’ would read in ‘667’ as 0.667 of a day.

\overline{h} n-digit fractional hour ($n \geq 1$). ‘hh’ would read in ‘78’ as 0.78 of an hour.

\overline{m} n-digit fractional minute ($n \geq 1$). ‘mmm’ would read in ‘234’ as 0.234 of a minute.

\overline{s} n-digit fractional second ($n \geq 1$). ‘sss’ would read in ‘567’ as 0.567 of a second.

\overline{l} or \overline{c} n-digit fractional millisecond ($n \geq 1$). ‘l’ or ‘c’ would read in ‘9’ as 0.9 of a millisecond.

\overline{n} n-digit fractional nanosecond ($n \geq 1$). ‘nn’ would read in ‘99’ as 0.99 of a nanosecond, but this accuracy is not guaranteed for all dates.

7.2.3 Pad Characters

Any character not used in the Date or Time Key Strings can be used in the 'TIME_FORMAT_STRING' as a pad character. To ease future enhancements, however, we recommend that such characters be restricted to the letters 'X', 'T', 'Z', <space>, and punctuation (e.g, ":"). Note, however, that leading and trailing <space> characters are stripped off of the 'TIME_FORMAT_STRING' and off of the data value itself, thus a non-<space> pad character should be used if required at the leading and trailing ends of the 'TIME_FORMAT_STRING'.

7.3 Examples

Here we give a few examples of entries in data files and the corresponding Free Time Format variable metadata for reading such files. A comment line for counting columns is included in each example to ease readability.

```
!123456789012345678901234567890123456789012345678901234567890
1985-09-11 14:22:46.667 -3.38101 -1.07421 2.34547
1985-09-11 14:23:06.667 -3.16654 -1.41437 1.90424
```

This example is in fact taken from a real .qft file which has its time in ISO format which is understood natively by QSAS and Qtran (using Time_format = ISO), but by way of illustration, in Free Time Format, it could have the following header fragment for the time field:

```
Start_variable = epoch
Column_start = 0
Column_width = 25
Data_type = epoch
Time_format = FREE_TIME_FORMAT
TIME_FORMAT_STRING = YYYY-MO-DD HH:MM:SS.sssXX
End_variable = epoch
```

Note the use of the pad characters 'XX' to ensure that the TIME_FORMAT_STRING is 25 characters wide after being stripped of leading and trailing white space. An alternative would be to specify Column_width = 23. The hyphens, space, colons, AND decimal point are all pad characters here which mimic the time string in the data file, thereby reducing the chance of error. We could have written instead

```
TIME_FORMAT_STRING = YYYYXMOXDDXHHXMMXSSXsssXX
```

which is much harder to read. Other alternative format strings include:

```
TIME_FORMAT_STRING = YYYY-MO-DD HH:MI:SS.LLLXX
```

and

```
TIME_FORMAT_STRING = YYYY-MO-DD HH:MI:SSSSSSXX
```

or

```
TIME_FORMAT_STRING = YYYY-MO-DD HH:MI:SS.sssXX
```

This last form cannot be SS.SSS as would find two seconds fields and use the 'SSS' as the seconds field and IGNORE the 'SS'.

Here is another example:

```
!123456789012345678901234567890123456789012345678901234567890
940921-Time N Vx Vy Vz
00:00:05.161  1.482  -218    13    -13
00:00:17.105  1.543  -216    13    -11
```

This file needs a header which skips the first two lines (using, e.g. ‘&2’ as the last line of the detached .qfh header or pre-pended .qft header).

The date is in the second line, which cannot be read directly, so here we can use the TIME_START or ‘All_records_time’ facility:

```
Start_variable = epoch
Column_start = 0
Column_width = 12
Data_type = epoch
TIME_START = 1994-09-21T00:00:00
Time_format = FREE_TIME_FORMAT
TIME_FORMAT_STRING = HH:MM:SS.LLL
End_variable = epoch
```

or

```
Start_variable = epoch
Column_start = 0
Column_width = 12
Data_type = epoch
YY_offset = 1900
Time_format = FREE_TIME_FORMAT
TIME_FORMAT_STRING = HH:MI:SS.MCC
All_records_format = YYMODD
All_records_time = 940921
End_variable = epoch
```

We could have also written:

```
All_records_format = YYYY-MON-DD
All_records_time = 1994-SEP-21
```

Another example:

```
!123456789012345678901234567890123456789012345678901234567890
 94 207 JUL 26  04:10:22.046  0  16    0.40599    0.01591
```

The leading space character is stripped automatically on reading the data, so the format string should start with the first non-space character.

```
Column_start = 0
Column_width = 28
YY_offset = 1900
TIME_FORMAT_STRING = YY Doy XXX XX  HH:MM:SSSSS
```


however is safest to skip the space by starting the read at the second column:

```
Column_start = 1  
Column_width = 27  
YY_offset = 1900  
TIME_FORMAT_STRING = YY Doy XXX XX HH:MI:SSSSS
```

or and we could have picked up the month/day instead of the date:

```
YY_offset = 1900  
TIME_FORMAT_STRING = YY XXX MON DD HH:MI:SSSSS
```

An example of use within a delimited file is:

```
!12345678901234567890123456789012345678901234567890  
1985-09-11 14:22:46.667 ,-3.381,2.43,27.987  
1985-09-11 14:23:06.667 ,-3.16654,1.346,5.793
```

This is a comma delimited file (.qfd), and can be read by:

```
Start_variable = epoch  
Data_type = epoch  
Time_format = FREE_TIME_FORMAT  
TIME_FORMAT_STRING = YYYY-MO-DD HH:MI:SS.sssX  
End_variable = epoch
```

Here the trailing X is not essential, as the time variable will be parsed to locate positions and lengths of recognised fields and any unused string is ignored. Note, if it were a space delimited file, the date and time components would end up in two variables.

A final example of the date/time separated over three variables. In this case we have day-of-year, year, orbit number, time field then some other data:

```
!12345678901234567890123456789012345678901234567890  
211, 1985, 10, 14:22:46.667 ,-3.381,2.43,27.987  
211, 1985, 10, 14:23:06.667 ,-3.16654,1.346,5.793
```

This is a comma delimited file (.qfd), and must be read by specifying three time variables. Note that the date/time components are chained together using the TIME_COMPONENT parameter.

```
Start_variable = epoch  
Data_type = epoch  
Time_format = FREE_TIME_FORMAT  
TIME_FORMAT_STRING = Doy  
TIME_COMPONENT = year  
End_variable = epoch
```

```
Start_variable = year  
Data_type = epoch  
Time_format = FREE_TIME_FORMAT  
TIME_FORMAT_STRING = YYYY  
TIME_COMPONENT = time  
End_variable = year
```

```
Start_variable = orbit  
Data_type = int  
End_variable = orbit
```

```
Start_variable = time  
Data_type = epoch  
Time_format = FREE_TIME_FORMAT  
TIME_FORMAT_STRING = HH:MI:SS.sss  
End_variable = time
```

Note the last component does not have a `TIME_COMPONENT` parameter as it completes the input. The component variables need not be consecutive or in ISO order (Day of year is before year in this example). The variable declarations must, however, appear in the same order as the variables in the record.

The same structure may be used for space separated data fields where the columns are not aligned (so that a tabular format does not work)

```
!123456789012345678901234567890123456789012345678901234567890  
211 1985   10 14:22:46.667  -3.381    2.43          27.987  
211 1985   10  14:23:06.667 -3.16654 1.346  5.793
```

where it is necessary to specify the delimiter as a space by:

```
Data_delimiter = ' '
```

or simply

```
Data_delimiter =
```

Reading flat files requires accurate header writing, and this is doubly so for Free Time Formats which depend critically on position and case. We recommend headers be checked by running Qtran to convert, for example, the .qft file (or if a large file just the first few records) with a Free Time Format specification into a .qfd file using the command `Qtran -od <file>` and visually comparing the results prior to, e.g., reading directly into QSAS or converting to cdf.

8 Reading Other ASCII Files

It is often possible to use the combination of detached headers and helper files to educate Qtran (and QSAS) in how to read other ASCII file formats, provided they are record based (each record holds data for each variable at a particular time or other dependency). There are two aspects to translating (or ingesting into QSAS) an ASCII file. Firstly it is necessary to provide the descriptions necessary to read the file, and secondly it is well worth ensuring that the minimal set of metadata is supplied that describes the data up to a standard to allow QSAS and other tools to handle the data intelligently.

Note that QSAS uses exactly the same methods to read a data file as Qtran, and if Qtran can translate a file into another format, then QSAS will also be able to read it without translation (but needs the same header or helper files).

Remember that only the first header found in the same directory as a data file will be read, so different data products should be kept in distinct folders with their own appropriate headers and helper files. Note also that all helper files will be read and the last read will take precedence if there are conflicts, so .qat files should be kept away from directories holding other data files otherwise they could impair the reading of those files.

8.1 Reading the File

Comparing the data file with the various formats described in this document is a first step towards determining how to read the file. Firstly decide if the file is tabular (all column starts are fixed with fixed width fields) or delimited (most often comma or space separated). Having chosen the route to take, compare with the relevant qft or qfd file metadata to determine what needs to be supplied, and either write a complete header (.qfh) or helper file (.qat) if a useable header already exists.

The contents of the header or helper file follow exactly the same syntax as required for these file types specified above. Ensure that variables are described in the order they appear within the records, and that tabular files have the start column set for each variable. Extra constraints such as the delimiter to use for data and/or metadata and whether the records are one per line or ended by a delimiter may need to be specified.

The hardest part of most files is assembling the time variable. The section on Free Time Format above will need to be used if the format is anything other than an ISO time string, or if it contains spaces. In space delimited files, a time field containing spaces will need to be read as separate variables and the Free Time metadata specified to allow them to be assembled. See sample file f_arraf.qfd for a time field split over variables.

8.2 Providing Metadata

Once a file can be read, it is worth adding essential metadata to the variable blocks so that the data is well described and software can present and manipulate the data intelligently. Essential metadata and its syntax are described in DS-QMW-TN-0010 and the full Cluster metadata descriptions are in the Data Dictionary, both in the doc folder in this distribution.

There are only a few essential attributes (and these are fully described in DS-QMW-TN-0010):

- SI_conversion (fixed syntax definition of the quantity's units and conversion factor to base SI)
- UNITS (string for axis labelling)
- DEPND_0 (timeline or other variable identifying records)
- Frame (vectors only)
- DEPEND_i (arrays only, where i is 1, 2, up to the array dimension to add a numeric bin value, e.g. energy or angle)
- LABEL_i (arrays only if described by text, e.g. x, y, z)
- FIELNAM (Name to help title plots since variable names can be long)

- LABLAXIS (name to use as axis label for scalars)

Of these, SI.conversion is perhaps the most important since it identifies the units in a machine readable syntax and allows intelligent arithmetic operations. There are many examples available in this document, in the sample files and in DS-QMW-TN-0010.

Sample Header

```
! Hand-edited Header - Sample
! SJS 15 May 1998
!
! Places requiring editing are denoted by  "! <-----"
!
!----- QSAS ASCII File -----|
! Free width data entries separated by delimiters      |
!                                                         |
! ASCII Format                                           |
! Native C ordering, last index varies fastest         |
! Blank lines are ignored                               |
! "\"" escapes rest of line as comment                 |
!-----|
!
File_name = SC_RR_INS_YYYYMMDD_Ext_n_V01.qfd    ! <-----

Attribute_delimiter = ,
File_type = d
Data_delimiter = ,
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                     Global Metadata    !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Start_meta = Logical_file_id
Number_of_entries = 1
Entry = SC_RR_INS_YYYYMMDD_Ext_n_V01.qfd    ! <-----
End_meta = Logical_file_id
!
Start_meta = Project
Number_of_entries = 1
Entry = PROJ>LONG PROJECT NAME      ! <-----
End_meta = Project
!
Start_meta = Discipline
Number_of_entries = 1
Entry = SPACE PHYSICS> MAGNETOSPHERIC PHYSICS    ! <-----
```

```
End_meta = Discipline
!
Start_meta = Source_name
Number_of_entries = 1
Entry = SOURCE ! <-----
End_meta = Source_name
!
Start_meta = Data_type
Number_of_entries = 1
Entry = RES>RESOLUTION ! <-----
End_meta = Data_type
!
Start_meta = Descriptor
Number_of_entries = 1
Entry = INS>LONG INSTRUMENT NAME ! <-----
End_meta = Descriptor
!
Start_meta = Data_version
Number_of_entries = 1
Entry = 01 ! <-----
End_meta = Data_version
!
Start_meta = Generation_date
Number_of_entries = 1
Entry = YYYY-MM-DDTHH:MM:SS.SSSZ ! <-----
End_meta = Generation_date
!
Start_meta = Caveats
Number_of_entries = 0
!Entry = PUT ANYTHING HERE or nothing ! <-----
End_meta = Caveats
!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                     Variables      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Start_variable = epoch
! scalar, no Size entry
Data_type = epoch
Time_format = ISO
!FILLVAL = -1.0E31 ! <----- uncomment if needed
End_variable = epoch

Start_variable = SAMPLE_VECTOR_B_FIELD ! <----- and below
Sizes = 3
Data_type = float
```

```
FIELDNAM = Magnetic field
Frame = vector>gse_xyz
SI_conversion = 1.0e-9>T
UNITS = nT
DEPEND_0 = epoch
DEPEND_1 =
!FILLVAL = -1.0E-31          ! <----- uncomment if needed
End_variable = SAMPLE_VECTOR_B_FIELD

Start_variable = SAMPLE_SCALAR_B_N_SIGMA      ! <----- and below
! scalar, no Size entry
Data_type = float
FIELDNAM = Normalised delta B / B
Frame = scalar>na
SI_conversion = 1.0>(ratio)
UNITS =                                ! no units
DEPEND_0 = epoch
!FILLVAL = -1.0E-31          ! <----- uncomment if needed
End_variable = SAMPLE_SCALAR_B_N_SIGMA

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                               Data                               !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!epoch,SAMPLE_VECTOR_B_FIELD,SAMPLE_SCALAR_B      ! <-----
!
! Specifying 00 data records means "unknown" and defaults to all data

! Extra white space between entries (for readability) is safe but not needed
! Time field must include space or ISO separators as 1995-01-23T02:33:17.235Z

Start_data = 00 ! Data records to follow
1995-01-23 02:33:17.235, 2.7453, -0.15678, 77.456, 2.3475
1995-01-23 02:33:21.124, 2.7453, -0.15678, 72.156, 2.1175
1995-01-23 02:33:25.921, 2.729, -0.15678, 77.456, 3.2128
1995-01-23 02:33:30.012, 2.7453, -0.12343, 72.156, -1e+31
1995-01-23 02:33:34.235, 2.1268, -0.11253, 83.501, 1.1194
1995-01-23 17:44:46.845, 12.341, 5.2345, 83.247, 2.1563
1995-01-23 17:44:50.334, 12.341, 5.2345, 83.247, 2.1563
1995-01-23 17:44:55.112, 12.341, 5.2345, 83.247, 2.1563
1995-01-23 17:44:59.345, 12.341, 5.2345, 83.247, 2.1563
1995-01-23 17:45:03.749, 12.341, 5.2345, 83.247, 2.1563
1995-01-23 17:45:08.153, 12.341, 5.2345, 83.247, 2.1563
```