IOS Header

User Manual

Produced by Matthew Nicoll,
Cypher Consulting
menicoll@CypherConsulting.com

May 29, 2007

Table of Contents

1 DOCUMENTATION

This document is available as a (read-only) Microsoft Word document file, located on the DFO network at:

\\paciosfp2\osapshare\libraries\ios_header\doc\UserManual.doc

The master copy of the manual is in the doc\gen folder. See ...\doc\SystemManual.doc for details on how the manual is mantained.

Most of this manual was converted from a VMS Help file. To preserve the formatting, a Courier font was used. As the document is updated in Word, it will gradually be converted to the Arial font.

Please send your comments about this document via Email to LinguantiJ@dfo-mpo.gc.ca and/or menicoll@CypherConsulting.com

2 HEADER DEFINITION

This section defines the IOS Header, independent of programming language and data-processing system. The structure, formatting rules and data types are defined.

For information on the FORTRAN implementation of the IOS Header, see the USAGE NOTES and FORTRAN PROGRAMMER's GUIDE sections.

This definition does not dictate whether the header for a data file resides in its own separate file, or whether it precedes the data in the data file.

Nor does this definition define file naming conventions, since this also will be data-processing system and operating system dependent.

2.1 STRUCTURE

The IOS Header is composed of:

- a Time Stamp record
- an Header ID record
- 1 or more Sections
- an End-Of-Header record ('*END OF HEADER' or just '*END ')

2.1.1 TIME STAMP

The Time Stamp record is the obligatory first record of an IOS Header.
Its format is as follows:

```
        *YYYY/MM/DD hh:mm:ss.ss
eg:     *1993/07/05 16:45:12.34
```

The primary purpose of the Time Stamp record is to identify separate data and header files as a pair. I.e. if both a data file and a header file have an identical time stamp as a first record, then you can be confident that the header belongs to the data.

It is intended that the time stamp be the approximate date and time at which the files were created.

It is acknowledged that it is annoying in some circumstances to have a non-data record on the data file, however:

- it is usually less inconvenient than having the entire header on the data file,
- it is merited to avoid header and data file mismatches.

Questions of when a time stamp on the data file is first required in a data processing system will have to be addressed separately for each system.

When the header is in the same file as the data, the time stamp has no purpose other than indicating when the file was created.

2.1.2 HEADER ID

The second line of the header identifies the header as an IOS Header.

The record is formatted as follows:

```
    *IOS HEADER [VERSION 1.0 yyyy/mm/dd yyyy/mm/dd]
eg: *IOS HEADER  VERSION 1.0 1993/06/01 1993/08/15
```

The section in square brackets is optional. Computer programs which write the header should include this information to identify the software, but if the header is being typed by hand, only the "*IOS HEADER" part must be entered.

After "*IOS HEADER", extra spaces or tabs may be inserted between character strings.

The version number is the version number of the header. As changes are made to the header, the version number will increase.

The date following the version number is the date that this version of the header came into effect.

The second date is for identifying the version of the software used to write the header. For the IOS FORTRAN implementation of the header it is the 'generation date' - the date the FORTRAN data structures and I/O code was generated from the header definition file.

2.1.3 SECTIONS

A Section is composed of:

```
    - a Section-Name record     (mandatory)
    - Item records              (optional)
    - Tables                    (optional)
    - Arrays                    (optional)  - added in version 2.0
    - a Remark Table            (optional)
```

Sections are delimited as follows:

The **start** of a section is identified by a Section-Name record - a record starting with an asterisk and followed immediately by a section name.

The **end** of a section is identified by a non-remark record starting with an asterisk.

(Note that 'starting with' means the first non-blank/tab character, which may or may no be the first byte on the record. See the Format Rules section.)

Sections:

- may appear in any order,
- may be defined as optional or mandatory within each data-processing system.

The COMMENTS section is a special case. It contains only an implicit Remarks Table (i.e. the $REMARKS and $END records are not included.)

Note that the first 2 and the last header records also start with asterisks, but are not header SECTIONS.

2.1.3.1 SECTION NAMES

These are the section names currently defined:

```
    COMMENTS
    FILE
    ADMINISTRATION
    LOCATION
    DEPLOYMENT
    RECOVERY
    INSTRUMENT
    HISTORY
    RAW
    CALIBRATION
```

2.1.3.2 SECTION NAME RECORDS

On a Section-Name record, the Section Name starts immediately after the asterisk, and extends to the first blank or tab. (I.e. it may NOT contain embedded blanks.)

The first three characters of the section name (not counting the asterisk) must be unique. The first three characters of a section name are referred to as the Section ID.

2.1.3.3 SECTION ID

The SECTION_ID is the first three characters of a section name.

2.1.4 ITEMS

A header Item is a single item of header data, identified with a Label. The item data must be of one of the defined header data types.

An header item record consists of:

- an Item Label,
- a colon, (optionally preceded and followed by spaces),
- item data.

Item Labels:

- appear between the start of the header record and a colon,
- are case insensitive,
- leading and trailing blanks and tabs are not significant,
- embedded tabs are equivalent to single blanks,
- multiple embedded blanks are equivalalent to a single blank,
- must be unique only within each Section.
- must not contain a colon.
- must not start with * or $. (Starting with a letter or number is recommended.)

Items:

- may appear in any order in a section.
- may be classified as Optional or Mandatory, within each data-processing system.
- may have look-up tables of data associated with them within each data-processing system. Such a data item is classified as a 'lookup' item.

See the FORMAT RULES section for item data formatting rules.

2.1.4.1 Continuation Lines

Header items of type Character may be continued on following lines, by coding a header label of CONTINUED. For example:

```
FORMAT      :  AAAA
  CONTINUED :  BBBB
  CONTINUED :  CCCC
```

will result in the string: `'AAAABBBBCCCC'`

Note that trailing blanks are lost when continuation strings are concatinated, even if they are in quotes. Embedded blanks can be preserved by putting them at the beginning of a quoted string. For example:

```
FORMAT      :  AAAA
  CONTINUED : '  BB   '
  CONTINUED :  CCCC
```

will result in the string: `'AAAA  BBCCCC'`

`"CONTINUED"` is thus a reserved item label.

2.1.4.2 Examples

```
NUMBER OF RECORDS:1234
MISSION          : 93-12A
START DATE       : PST 1993/08/25 12:30
LATITUDE         : 49 30.1234 N
SCIENTIST        : Felix Bloggs
SCIENTIST:Felix Bloggs
SCIENTIST:'Felix Bloggs'
```

2.1.4.3 Standard Items

Standard items have pre-defined labels, are in designated sections, and have pre-defined data types. See the sample full header for a complete list of currently defined standard items.

2.1.4.4 Custom Items

Custom Header Items cover the inevitable cases where additional header information is required (coded in a more formal manner than in a comment). When looking at a header file, Custom Items are not distinguishable from Standard Items. They simply are not on the list of pre-defined items.

Custom Items differ most in how they are handled by header I/O software, how they are stored in memory, and how they are accessed in user programs.

See the PROGRAMMER GUIDE for details of how Custom Items are handled in the IOS FORTRAN implementation of the IOS Header.

2.1.4.5 Lookup Items

A Lookup item is one which has a table of pre-defined values associated with it. The item may be further classified as an 'Open' or a 'Closed' Lookup item. If it is Open, values other than the pre-defined values may also be entered. If it is Closed, only the pre-defined values should be entered.

Header data entry programs, and header verification routines should check that a Lookup item value is on the associated table. If it is not, then a warning or an error message should be displayed, depending upon whether it is an Open or Closed Lookup item.

NOTE: The Lookup item concept has NOT been IMPLEMENTED (As of Ag-1993).}

2.1.5 TABLES

Tables are a fixed entities in the header definition - i.e. you cannot add your own table (as you can with arrays).

Table names are unique only within sections. E.g. both the RAW and FILE sections have a CHANNEL table.

Tables:

- start with a record of the form:

```
$TABLE: table_name
```

- the colon may be preceded by blanks or tabs,
- there must NOT be any blanks between $ and TABLE
- the table_name may have embedded blanks (multiple embedded blanks are reduced to single embedded blanks for name identification). No quotes are allowed.

- end with a record starting with $END (case insensitive).
- are column independent; i.e. the table column to which a field belongs is determined by that field's relative position on that row, not by its absolute position in the record.
- blanks (or tabs) separate fields in a row.
- right-most table entries may be omitted by leaving them blank, but null entries elsewhere in a table row must be indicated by coding a space in quotes, '?', or 'n/a'.
- an entry in a row may be one of the defined header data types, or, in special cases, it may be a List and/or an Array reference. (Lists are defined in the following section.)

2.1.6 LISTS

A List is one or more data items, all of the same type, separated by blanks, and all enclosed in parentheses.

In the current definition of the IOS header, the only lists which occur are coefficient lists and source-channel lists in the tables which define data calibration or correction.

A List:

- may be continued over following records, but must start on the same record as the rest of the table row.
- blanks following the opening parenthesis and preceding the closing parenthesis are optional.
- a null list may be denoted by coding left and right parentheses with nothing between them, or by omitting the list completely.

When there are TWO lists in the same table row (e.g. source channels AND coefficients):

- they must either be BOTH present, or BOTH absent.
- if one is null, and the other is not, the null list must be coded as '(b)', where b is zero or more blanks or tabs.
- the first list must start on the same record as the rest of the table row.
- the second list may start on the same record as the first (if there is room), or on a following record.

2.1.7 Remark Tables

Any section in the header except for the COMMENTS section may include a Remarks Table.

A Remarks Table starts with a record which starts with the string: $REMARKS, and ends with a record which starts with the string: $END

Remark text is entered on one or more lines between these two delimiting records.

An 8 space indentation, and a right margin of 80 is the suggested format. Limiting these lines to 72 characters (not counting any indentation) will allow the header WRITE routine to indent 8 spaces without going past column 80.

Comments of length 80 are allowed, however. As in the COMMENTS section, long remarks may just prevent indentation.

2.1.8 ARRAYS

An IOS Header ARRAY is simply an array of single precision real numbers, with rows and columns. It is coded in the header as follows:

```
$ARRAY: array name
    x11 x12 ... x1m
    x21 x22 ... x2m
    :   :         :
    xn1 xn2 ... xnm
$END
```

Arrays may occur in any section except COMMENTS.
Arrays may occur anwhere within a section.

2.1.8.1 EXAMPLE

```
$ARRAY: COMPASS CORRECTIONS
     5.  .123
    10.  .234
    15.  .345
$END
```

In this example, the ARRAY NAME is 'COMPASS CORRECTIONS'
It has three rows, and two columns.

2.1.8.2 NAMES

- are subject to the same coding rules as table names.
- must be unique within the entire header. (I.e. the same array name can NOT be duplicated in another section, as item labels can.)

2.1.8.3 RULES

- `arrays may have as many columns as will fit on a header record.`
- the number of columns in the first row defines the number of columns of the array.
- `no row may have more columns than the first row.`
- short rows: the missing columns have the assumed value of NULL_BLANK.
- array elements may be coded like any item of type R4: as an integer, F format, E format, or any of the header null values: ?, n/a and ' '

2.1.8.4 REFERENCES

- an array reference is an array name enclosed in square brackets.
- e.g.: [COMPASS CORRECTIONS]
- the only place where array references are currently allowed is in the Coefficients column of a calibration table, where the array reference must precede the coefficient list.

2.2 HEADER DATA TYPES

Symbol	Name	Description
I	Integer	default length
I2	Integer*2	2 bytes
I4	Integer*4	4 bytes
R4	Real*4	4 bytes
R8	Real*4	8 bytes

```
C       Character          character string data.

FT      Full-Time          Composed of:
                           C*3     : Zone,
                           Integer: year, month, day, hour, minute
                           Real    : seconds

                           Example: PST 1993/08/15 12:30:12.345

LL      Lat-Lon            Composed of:
                           Integer: degrees
                           Real    : minutes
                           C*1     : hemisphere  (N, S, E or W)

                           Example: 49 12.34567 N

TI      Time-Increment     Composed of 5 real numbers, for time in:
                           Days, Hours, Minutes, Seconds and Milliseconds.
                           The sum of the five times is the time-increment.

                           Example: 1 0 0 1.5 0    (1 day and 1.5 seconds)
```

See the FORMAT RULES section for coding rules for these data types.

Use the character strings 'YES' and 'NO' for boolean (logical) data.

2.3 FORMAT RULES

The objective in designing the formatting rules for the IOS header was that the same header file could be:

1. easy to read on a computer screen,
2. easy to type in or edit on a computer screen,
3. printed un-modified as a well formatted 'report',
4. read by computer programs.

2.3.1 General rules

- Leading blanks or tabs in a header line are not significant, so the 'first' or 'starting' character in a header record is the first character which is neither blank nor tab.
- There is no column dependence.
- Blank lines may appear anywhere in the header after the first line of the header.
- When there are multiple fields on a line, the fields must be separated with one or more blanks or tabs. Fields may be ommitted at the end of a line, but missing fields in the middle must be indicated with the character representation of a null value (n/a, ?, or ' '.)

2.3.2 Record length

Header records may be up to 132 bytes long, however:

- using only columns 1 to 80, allows the header to be viewed on an 80 column screen/window, or printed in protrait mode on paper.
- no more than 80 bytes may be spanned from the leftmost to rightmost nonblank character in the COMMENTS section or a REMARKS table.

2.3.3 In-Line Comments

Annotations may be added to any header record (except for a remark record) by coding an exclamation mark, and following it, on the same line, by text. There is no requirement, however, for a program which reads the header, to keep these annotations or pass them on to an output header.

Text from an ! to the end of line is ignored

EXCEPT: - if it occurs after column 1 in the COMMENTS section,
- if it occurs after column 1 in a REMARKS table,
- if the ! is in a quoted string.

This feature is primarily for use by a header WRITE routine for labelling the header information, e.g. table column headings.

This type of comment could be added manually with an editor, but the comments may not be carried forward through READ and WRITE routines.

A user could annotate a header with In-Line comments before sending a file offsite, for example.

2.3.4 Real data

Real numbers may coded in the usual ways:

```
 1.234
 1.56E-2
 0.0003
-123456.7
 8.0
 8.
 8
```

The last example shows that if there is no fractional part, the decimal point may be omitted.

There must be no embedded blanks (after a minus sign, for example).

2.3.5 Character data

Character data may be put in single quotes, but quotes are not necessary if the string contains no blanks or exclamation marks. Rules on when quotes are required are more lenient for Items than for strings within Tables:

Items:

- If the string is NOT quoted, leading and trailing blanks are stripped, but all embedded blanks and tabs are retained.
- Quotes are only required if leading blanks or embedded exclamation marks are to be preserved.

Tables:

- Quotes may only be omitted if there are no embedded blanks. This is because blanks separate columns within a table.

2.3.6 Date & Time

The Full-Time data type is used for all time-related items. A Full- Time is fully qualified - with time zone, date and time all together, e.g.:

```
zon yyyy/mm/dd hh:mm.ss.sss
PST 1992/08/12 12:33:12.123
```

This is the standard way of formatting a full-time, but extra blanks or tabs may be inserted, and less accuracy in the time field is allowed, e.g:

```
PST   1992/08/12     12:33
UTC   1992/08/12     12:33:12
```

Any of the three fields may be coded null. For example if the zone is unknown and the time is irrelevant:

```
    ?  1992/08/12 n/a
```

Note that the dates and times in the PROGRAM table of the HISTORY section are not Full-Times: they have no time zone, and no fractional seconds.

2.3.7 Lat Lon

Latitudes and Longitudes are expressed in three fields:

```
    Degrees:    Integer
    Minutes:    Real
    Hemisphere: N, S, E or W
```

Minutes may be expressed with up to as many digits as can be stored in a single precision real number (typically 7).

The Hemisphere may be put in quotes, but it is not necessary to do so.

The three fields may be separated by one or more blanks or tabs, but header write routines should format them as follows, separated by one blank:

```
    123 30.1234 W
     49 30.1234 N
```

Note that header write routines should format the degrees as a three character field, with leading blanks if necessary, so that latitudes and longitudes line up vertically when put on successive lines.

2.3.8 Time increment

A time increment is coded as 5 real numbers: day, hour, minute, seconds, milliseconds - separated by one or more blanks or tabs.

E.g., for a time increment of one day and 1.5 seconds:

```
       1  0  0 1.5 0
or:    1  0  0 1 500
```

2.4 NULL VALUES

Any header data - either an Item, or an element of a table - may be coded as one of the following 'null' values:

```
    ?       - unknown
    n/a     - not applicable
    ' '     - blank
```

Header Items of any data type may be simply left blank - no quotes required. However a column in the middle of a table may not be simply left blank, because the position in the row must be marked in some way. In this case, code one blank in single quotes.

The null symbols may be put in single quotes, but it is not necessary to do so. E.g. '?' is equivalent to ?.

These symbols may be used regardless of the data type. A header READ routine must read in every data field as a character string, and check for a field being blank, '?' or 'n/a' before trying to decode it as a number.

See the Usage Notes section for more information on the use and implementation of null values (including information on the internal 'null-absent' null value).

For composite (structure) data types, null values are handled as follows:

```
LL - Lat/Lon       - a Latitude or Longitude is either all null,
                     or all not-null
                   - the null value is stored in the Degrees field.
                   - thus to code a latitude as 'unknown', a single
                     ? is sufficient.

TI - Time Increment- the null value for the whole is stored in the
                     Day field.
                   - thus to code a time increment as 'unknown', a
                     single ? is sufficient.

FT - Full Time     - Zone, Date, and Time may be independently null,
                     using Zone, Year and Hour.
                   - thus to if the zone is unknown, and the time
                     is not applicable, code:
                       ? 1993/08/10 n/a
```

3 SECTION DESCRIPTIONS

This section describes each of the Header Sections. Under each Section, the Tables and some of the Items of that section are described. Items which require no explanation other than given in the EXAMPLE FULL HEADER are not included here.

See the EXAMPLE FULL HEADER for a header containing all Sections, Items, and Tables.

3.1 COMMENTS

This section is for remarks which pertain to the entire header.

Each line may be up to 80 characters long, however limiting the length to 76 will allow the header WRITE routine to indent 4 spaces, without pushing text to the right of column 80. The output routine will not indent if doing so would push text past column 80.

Note that each section may also have its own remarks in a Remarks Table within the section.

3.2 ADMINISTRATION

Note that PLATFORM is a more general term for Vessel.

3.3 LOCATION

3.3.1 EVENT NUMBER

EVENT NUMBER may be used for things such as CTD Cast Number.

This is an Integer field.

Making it a character field to allow non-numeric event identifiers was considered, but this was not done, to avoid the possible confusion over whether strings such as:

```
    '001'     '  1'      ' 1 '       ' 01'
```

would be considered identical or not!

3.3.2 DATUM

This item can be used to specify the ellipsoidal reference system.

3.3.3 Latitudes and Longitudes

The 'LATITUDE 2' and 'LONGITUDE 2' items may be used to define a second location. Thus a rectangle can be defined, or the first and last locations of a series (e.g. drifter track) etc.

3.3.4 UTM

UTM ZONE, UTM NORTH and UTM EAST can be used to define a position in UTM (Universal Transverse Mercator) coordinates. As with Latitude/Longitude, a second position can be defined.

3.3.5 MAGNETIC DECLINATION

This item enables the magnetic declination at the location and time of the data to be recorded. This is relavent for data, such as current meter data, which is logged with direction relative to magnetic north.

3.4 DEPLOYMENT and RECOVERY

These sections contain information specific to the instrument deployment and recovery.

Some of the information traditionally considered to be deployment and recovery information - such as number_of_manual_triggers, and first_good_data_time, have been put into the RAW section, because they are part of the description of the raw data.

3.5 INSTRUMENT

Instrument DEPTH should be in metres.

REVS PER COUNT and ROTOR TYPE were included with Aanderra current meters in mind.

The SENSOR table may be included to define multi-sensor instruments. The sensor NAME may be coded in the same manner as channel names, with a primary name, and qualifiers, separated by colons. The primary name could be a lookup field. The Sensor Names need NOT be the same as the names of the raw channels they produce.

The sensor DEPTH should be an absolute depth in metres.

3.6 HISTORY

This section has no header 'items' at present, but may contain a PROGRAMS table (a log of programs which have changed the data) and/or a REMARKS table.

The date and time in the PROGRAMS table are independent character fields - they are not part of a Full-Time data type. The date format is 'YYYY/MM/DD'. The time format is 'HH:MM:SS'.

While not every program (especially old ones) will set the Records In and Out columns, this information could provide a trace of where data records have gone.

3.7 RAW

The RAW section contains information describing the original raw file. It should not be modified by any subsequent data processing programs.

Notes:

```
START TIME          - time of the first data record
FIRST GOOD REC TIME - this is typically the time when the instrument
                      was actually deployed.  Records prior to this
                      time may have been logged on the bench during
                      setup.
END TIME            - the time of the last data record
TIME INCREMENT      - coded as 5 real numbers, as defined under
                      DATA TYPES.
NUMBER OF RECORDS   - number of records in the raw file
FIRST RECORD        - up to 80 bytes of the first raw record.
START MANUAL RECS   - the number of manually triggered records at
                      the beginning of the file.
END MANUAL RECS     - the number of manually triggered records at the
                      end of the file.
VOLUME ID           - identifies the physical volume upon which the
                      raw data is stored: e.g. the tape reel number.
```

The CHANNELS table should define all the raw data channels.

For additional notes on observation times, and a definition of the Averaging Interval and Time Lag columns of the CHANNELS table, see the **Usage Notes / Timing Variation Between Channels** section below.

3.8 CALIBRATION

3.8.1 TIME UNITS and TIME ZERO

If the calibration program generates a time channel, one or both of these items should be included in the header.

See 'Time Channel Definition' in the USAGE NOTES section for details.

3.8.2 START TIME and TIME INCREMENT

If the calibration program generates a time channel, it should copy the START TIME and TIME INCREMENT into the CALIBRATION section from the FILE section of the input file header. This serves as a record of how the time channel was calculated.

The START TIME is the time of the first data record.

3.8.3 RECORDING BITS

This item is included for compatibility with the Arctic Multi-Line Header. Some Arctic Group programs use this number to determine Pad values, and for calibration.

3.8.4 RAW CHANNELS TABLE

This table:

- instructs the calibration program how to calibrate the raw data channels,
- serves as a record, after calibration, of how the calibration was done.

Raw channels which require no calibration need not be listed in this table.

The table has four columns:

```
Name           - see "Channel Names" in the USAGE NOTES section
                 of this document.
               - may contain a full channel ID (name [units to uniquely
                 identify a channel.  (The units column cannot be used for
                 channel identification purposes, since it must contain the
                 output units.)
               - the channel name length restriction (40) applies, even if
                 even if units are included.

Units          - units after calibration is done.
               - should conform to a list of standard units

Formula number - an integer identifying a pre-defined formula type.

Coefficients   - this column may be omitted, or must contain:
                    - a coefficient list, or
                    - an array reference AND a coefficient list.
               - a coefficient list consists of zero or more
                 numbers, separated by blanks, enclosed in brackets.
               - an array reference is an array name enclosed in
                 square brackets.
               - referencing an array is an alternative to coding a
                 large number of coefficients in a list.
               - several lines may be used.
```

3.8.5 CALCULATED CHANNELS TABLE

This table defines channels which are to be calculated from other channels.

This table is coded in the same manner as the Raw Channels table, except that the coefficient list must be preceded by a 'source channels' list, where the other channels required for the calculation are listed. In a source channels list:

- only enough of the Channel Name to uniquely identify the channel need be entered.
- channel names with embedded blanks must be in quotes,
- a full channel ID (with units in square brackets) may be coded if necessary, but may not be longer than 40 characters.

The pseudo-source-channel name CALCTIME is suggested for telling a calibration program to calculate a time for each record for use in calibration.

If an array reference is to be included, it must be put between the 'source channels' list, and the coefficient list.

The Coefficients column may be left entirely blank, or must contain:

- a source-channel list AND a coefficient list, or
- a source-channel list, an array reference, AND a coefficient list.

(Confused? If you code ALL or NOTHING, you won't go wrong.)

```
- Examples:     (Temp Sal) (1.123 2,345)
                () (1.234 2.345)
                () [] (1.234 2.345)
                (Temp Sal) ()
                (Temp Sal) [EXAMPLE ARRAY] ()
```

3.8.6 CORRECTED CHANNELS TABLE

This table was included to describe data corrections done as a result of water samples, but may be used for any secondary calibrations or corrections.

This table is coded identically to the Calculated Channels Table.

3.9 FILE

This section describes the current state of the data file.

(The RAW and CALIBRATION sections remain static after calibration.)

3.9.1 START TIME and END TIME

- for time series data
- are the times of the first and last records of the data file.

3.9.2 TIME ITEMS

`TIME_INCREMENT, TIME UNITS and TIME ZERO`

These header items are described in the USAGE NOTES / TIME CHANNEL DEFINITION section of this document.

3.9.3 FILE TYPE

- indicates the type of the data file.
- ASCII and BINARY are the two recommended file types to use. (ASCII corresponds to the Fortran 'FORMATTED' file type, BINARY corresponds to the Fortran 'UNFORMATTED' file type.)
- Other file types are possible, such as EBCDIC.
- (Note that Microsoft Fortran has a BINARY file type option which is different from UNFORMATTED. If this file type is ever used, invent a file type other than BINARY to use in the header.)
- this item is most useful in systems where the header and data are in separate files. (When header and data are together, a program must know the file type before opening the file.)

3.9.4 FORMAT

- is a character string containing a description of the format of the file, using whatever conventions are most convenient for the user's data processing system.
- for Fortran applications, when there is no Channel Details table, this will be a FORTRAN format (including surrounding parentheses). If the format string does not fit on one line, continue it with one or more header items labelled CONTINUED.
- should be included if the FILE TYPE is ASCII, unless the format is described in the Channel Details table.
- may be included for BINARY files, to indicate how the data should be formatted if it is converted to ASCII.
- code FREE (upper case) to indicate free-format: data which has no column-dependency. (I.e. the data - if real, integer or character, can be read with FORTRAN * format.)
- code DELIMITED (upper case) to indicate data is in delimited fields (e.g. a CSV file). See notes below on the delimiter and quote characters.

Format information can be specified more rigorously, and in a programming language independent manner in the Channel Details table. If format information is in both places, the FORMAT Item shall take precedence.

<u>DELIMITED options</u>:

```
DELIMITED d q
 where:
   d   - may be a single delimiter character, or COMMA or TAB (spelled out)
       - optional - default is comma.
       - must be coded if q is coded.
   q   - used only for output of character data with leading blanks.
       - may be a quote character, or SINGLEQUOTE or DOUBLEQUOTE
       - optional - default is double quote
       - ignored for input
       - for input, character data may be:
            - unquoted: leading blanks will be removed,
            - in single or double quotes.
```

When FORMAT: DELIMITED is specified, the channel detail table:
- may be omitted if all channels are type REAL*4
- must be coded to define special data channels
- type CHARACTER: width and format columns are ignored
- types REAL and INTEGER: format information will be used for output only.
- types LAT,LON,date,time: format information will be used as usual.

Examples:

FORMAT: DELIMITED
- delimiter is comma, output quote character is double quote

FORMAT: DELIMITED TAB SINGLEQUOTE
- delimiter is tab (ASCII 9), the output quote character is a single quote

FORMAT: DELIMITED |
- delimiter is "|", output quote character is double quote

3.9.5 DATA TYPE

This item is for describing how the data in the data file should be stored in a computer.

If all data channels are the same type, code that data type in this item. (e.g. REAL*4, or R4)

See the USAGE NOTES section of this document for a more detailed discussion of this topic.

3.9.6 DATA DESCRIPTION

This item is for describing the general type of the data, e.g. 'CTD', 'Current Meter', 'Shoe Size'.

3.9.7 PAD

- the data value used to indicate missing (null) data in the data file.
- if not all channels use the same pad value, code the pad values for each channel in the CHANNEL DETAIL table.
- if both the PAD item and CHANNEL DETAIL table are included, any non-null PAD values in the table will take precedence.
- code as an integer, a real number, or one of the header null values.
- coding a header null value (? n/a ' ') means that there is no pad value. It does NOT mean that the pad value is the numeric equivalent of the null pad value!
- pad values do not apply to character-type channels.

3.9.8 RECORD LENGTH

- the record length in bytes.

In VAX Fortran OPEN statements for binary files, the record length has to be expressed in 4-byte words. In this case, the program will have to divide RECORD LENGTH by 4.

3.9.9 NUMBER OF CHANNELS

This must be the number of channels of data in the data file. It should match the number of rows in the CHANNELS table.

3.9.10 CRC

CRC stands for "Cyclic Redundancy Check".
This is a text field, so that a CRC value can be displayed in hex.
At IOS, there are CRC calculation utilities in the ocean physics fortran library.
The intention is that a CRC will be calculated early in a dataset's processing cycle, and remain unchanged after subsequent processing. The CRC will thus serve as a tracing identifier, allowing a positive link to be made between various versions of a dataset.

3.9.11 CHANNELS Table

All channels of the current data file must be described in this table.

The Name and Units columns are the same as in the tables described in the CALIBRATION section.

The Minimum and Maximum columns are for the minimum and maximum data values in the data. Being at the end of the line, they may be left out if not available.

Channel numbers must start with one and increase sequentially, OR they may be coded ' ' or ? to imply the positional channel number.

3.9.12 CHANNEL DETAILS Table

This table is optional. If included, its rows need not correspond one-to-one with the CHANNELS table. Only channels with detail information need to be included. The channel number in column one is used to link the detail information to a channel.

Not every channel must be included, but channel numbers must be in increasing order, with no duplicates.

Consult the administrator or guidelines of your data-processing system to find out how or if this table should be included.

While the columns in the details table could have been included in the CHANNELS table, they were separated for the following reasons:

1. to help keep the header within 80 bytes wide for easy viewing, printing etc.
2. to keep the header as simple as possible for the simple cases (i.e. have complexity only when the situation is complex).

The PAD column allows different pad values to be specified for each channel. See the description of the PAD item for coding rules. A non-null pad value in the details table will over-ride a value specified in the PAD item.

The START, WIDTH and DECIMAL_PLACES columns specify the location of individual channels in the data record, without using any programming language dependent codes.

The FORMAT column may contain either a programming-language-dependent format specifier for the channel, or a single character format type: E, F, I, or A (case independent). If the single character code is used, the START, WIDTH and DECIMAL_PLACES fields should be coded to complete the format description.

The TYPE columns may be used to describe the type of the data, in a programming-language-dependent manner (e.g. REAL*4), or with one of the IOS_HEADER type descriptors (e.g. R4). (See - DATA TYPE, in the USAGE NOTES section.)

The TYPE and FORMAT columns may be used to describe special channels - channels which are not formatted as simple real numbers in the data file. See the SPECIAL_CHANNEL_TYPES section for details.

If width and decimal places information is the FORMAT field, AND in the individual WIDTH and DECIMAL PLACES fields, the information in the FORMAT takes precedence. E.g. if WIDTH = 10, DECIMAL PLACES = 7, and FORMAT = F8.3,then the 8 and 3 from the FORMAT will be used

4 USAGE NOTES

This section contains information on general header usage topics (those which are not related to single Sections, Items or Tables), including notes related to the IOS FORTRAN implementation.

4.1 Pad Values

A 'pad' data value is a null data value. It takes up space in the data file, while indicating that there is no data for that space.

Pad values are specified in the CALIBRATION and FILE sections.

Pad values cannot be specified for raw data. Formally describing the various possible definitions of 'invalid' raw data would not be simple. This type of information is suitable for the REMARKS table in the RAW section of the header.

4.2 Null values

Any header data - either an Item, or an element of a table - may be coded as one of the following 'null' values:

```
    ?       - unknown
    n/a     - not applicable
    ' '     - blank
```

Instead of leaving a field blank, it is recommended that one of the other null symbols be used, to make exlicit WHY there is no data coded.

There is a fourth null value used to indicate in memory that an item was not (or will not be) in the header at all (i.e. no label or data). This null value is called null-absent.

The various types of 'null' values are stored internally as follows:

```
  Null Value Type                        Int/Real   Character
  ------------------------------         --------   ---------
  1 - not present in the header            -9991      'o'
  2 - not applicable                       -9992      'n/a'
  3 - unknown                              -9993      '?'
  4 - blank                                -9994      ' '
```

(See the PROGRAMMER GUIDE section of this document for information on programming aids for handling null values.)

This scheme dictates the following restrictions:

- excludes these numbers from being used as data or pad values,
- excludes any header character string from being 'o' (a lower case oh)
- character strings defined with length 1 or 2 could not have the value 'n/a'.

For composite (structure) data types, null values are handled as follows:

```
  LL - Lat/Lon       - a Latitude or Longitude is either all null,
                       or all not null
                     - the null value is stored in the Degrees field.

  TI - Time Increment- the null value for the whole is stored in the
                       Day field.

  FT - Full Time     - if absent, only the Zone need be null-absent
```

```
                    - otherwise Zone, Date, and Time may be
                      independently null, using Zone, Year and Hour.
```

4.3 Channel names

The maximum length for Channel Names is currently (Aug 93) set at 40.

How channels are named is data-processing-system dependent, however the following scheme is recommended for use at IOS.

A Channel Name consists of a general parameter name, e.g. TEMPERATURE, followed by one or more detailed qualifiers e.g. AIR, or THERMISTORn. A colon is used to separate the sections, e.g.

```
'TEMPERATURE:AIR'
'TEMPERATURE: AIR : HI_RES'
```

A standard list of names and qualifiers could be defined, and programs which allow entry of channel names could verify names against this list.

The idea is that the general parameter names be quite strictly controlled, but that more flexibility be allowed in the qualifiers.

Channel Names may contain blanks. If a Channel Name contains blanks, the blanks are significant, and the name must be specified in single quotes.

If a channel needs more description than can be contained in the 40 characters of the Channel Name, the COMMENTS section or REMARKS tables can be used.

4.4 Time increments

For time series data, time increments may be specified in the CALIBRATION and FILE sections.

In the CALIBRATION section the Time Increment can be used to instruct a calibration program how to generate a time channel, or to record, after the fact, what time increment the calibration program used to generate the time channel).

In the FILE section, the time increment should indicate the current increment between records, whether or not there is a time channel in the data file.

Note the following about header time increments:

```
- expressed as 5 real numbers (decimal points optional)
- Days Hours, Minutes, Seconds, Milliseconds
- this allows for flexibility, e.g.:
    .5 0 0 0 0          ! half day intervals
    .5 0 0 0 -16        ! half day (less 16 milliseconds)
    0  0 30 -1 0        ! 30 minutes (less 1 second)
```

4.5 Time Channel Definition

A data file may have the time included in each data record. There are various ways this can be done, using one or more channels, of various types. For example the year, month, day, hour, minute and second could all be defined as separate channels; or the date and time could be stored in a channel each. In such cases, the time channels may be defined in the header just like any other data channels.

It is also possible to code dates and times as formatted strings in each record. See the section called SPECIAL CHANNEL TYPES for information on formatted date and time channels.

It is often convenient to store the time as a single real number. Two methods of doing this are allowed for, using special header items:

DAY_OF_YEAR method:

- the day_of_year is represented in the integral part of the real number, and the time in that day is represented as a fraction of a day in the fractional part of the real number.
- E.g. 1.5 is noon on January the first.
- if the time series spanned a year end, the time channel would jump from 365.??? back to 1.???
- this is the method used in the old Arctic group's Multi-Line Header system, and is referred to as 'Decimal Days' in that system.
- the START TIME header item should be coded to identify the time_zone and year of the data.

Offset method:

- time is recorded as an offset from a specified 'Time Zero'.
- the time units of the offset are specifiable.

The TIME UNITS header item defines both the method used and the units used. TIME UNITS may have one of the following values:

`DAY_OF_YEAR`	-	day_of_year method, as described above
`DAYS`	-	offset in days
`HOURS`	-	offset in hours
`MINUTES`	-	offset in minutes
`SECONDS`	-	offset in seconds.

If the Offset method is used, the TIME ZERO header item must be included to specify the base of the offset.

The offset method with a specifiable time zero allows for flexibility. For example time zero might be set at:

- the time of the first record (so the time on the first record would be zero),
- one time increment prior to the first data record,
- the beginning of the day in which the time series begins,
- the beginning of the year in which the time series begins.

If time zero was set at yyyy/12/31 00:00, where yyyy is the year prior to the year in which the time series begins, then the result would be the same as using the DAY_OF_YEAR method, except that the time would continue to increase over the year-end.

If there is a time channel, the FILE section of the header should include the TIME UNITS item and, if the offset method is used, the TIME ZERO item.

If the header includes a CALIBRATION section, these items should also be included there, to record how the time channel was originally generated. If the time channel was re-calculated subsequent to calibration, these header items would be updated in the FILE section only.

In the FILE section, there will be a small redundancy: the time units will also appear in the CHANNELS table (if the data file has a time channel). In this case it is up to the user to keep the TIME UNITS item consistent with the units of the time channel in the CHANNELS table.

Programs which perform time calculations should rely on the TIME UNITS item, so that the time channel itself need not be in the file.

(Note that the term 'Day_of_year' is used instead of 'Julian Day', to avoid confusion with the 'Julian Day' which is the number of days since Nov 24, 4713 BC.)

4.6 Timing Variation Between Channels

The RAW section CHANNELS table has optional columns called **Averaging Interval** and **Time Lag,** to account for data timing variations between channels.

These columns were added in March 2006, with Time Lag originally named Time Offset. "Offset" was changed to "Lag" in May 2007.

These times are stored and displayed in the same manner as the Time Increment header items: five numbers, one each for days, hours, minutes, seconds and milliseconds.

In a Fortran program, these time values are defined in the Raw Channel table defined in include file SR_RAW.INC. For channel i, the time values are referenced as:

```
H.FIL.RAW.CHANNEL(i).AVERAGING_INTERVAL
H.FIL.RAW.CHANNEL(i).TIME_LAG
```

Awareness of such timing differences has long been a part of Aanderaa RCM culture, and not acknowledged in documentation. Runs of the tidal analysis programme required special pre-processing of Aanderaa data to "correct" this problem.

Each data record has a "Reference Time", which is computed from the starting time and the time increment as always. The actual observation time, or "Datum Time" will differ from the "Reference Time" if the data are time averages, or if sampling is sequential. The latter is the situation with the SBE19, for example. Note that in the past, as with the Aanderaa speed/direction, idiosyncrasies have either been ignored (common if the lags between sequential data are small relative to the interval between records) or been managed using ad hoc software. Here we describe a systematic approach to timing of observations in data documentation and processing.

Name	**Sym**	**Description**	**Applies to**	**Stored in**
Datum Time	[TD]	The true time of the observation, defined as the time at the mid-point of the averaging interval.	Data Value	
Reference Time	[TR]	The nominal time of the observation, recorded by the instrument or calculated from the Record Number, Start Time and Time Increment.	Data Record	each data record. (or calculated for each record.)
Start Time	[TS]	The Reference Time of the first record in the data file.	File	RAW section item.
Time Increment	[TI]	The interval of time between successive measurements of the same variable.	File	RAW section item.
Averaging Interval	[TA]	The length of time taken to acquire the observation. In many modern instruments, this is comparable in length to the Time Increment.	Channel	RAW section CHANNELS table
Time Lag	[TL]	The time interval between the Reference Time and the Datum Time. The value is positive if the Reference	Channel	RAW section CHANNELS table

		Time occurs before the Datum Time. In some instances the Time Lag may equal half the Averaging Interval (e.g. for Aanderaa RCM4) but this is not true in general.		

[TR] = [TS] + [RecNo - 1] * [TI]
[TD] = [TR] + [TL]

EXAMPLES

Aanderaa RCM4

[TI] = 60 min
[TL] = 5 * [TE] for direction and [TL] = 6 * [TE] - 0.5 * [TI] for speed.

Here [TE] is the time taken for the encoder to digitize, sequentially, each of the six signals from sensors; [TE] ~ 0.5 min The time lag for speed equals half the time increment, since the speed is averaged over the time increment. It is negative because the averaging precedes writing to tape. Typically, [TE] << [TI] and is assumed to be zero.

RDI ADCP

[TI] = 30 min
[TA] = 28 min for velocity and scattering.
[TA] ~ 0 for temperature, pressure.
[TL] = 0.5 * [TA] since the reference time is the start of the ADCP averaging interval.
[TL] = +14 min for velocity and scattering.
[TL] ~ 0 for temperature, pressure.

SBE19 CTD (sequential sampling)

[TI] = 0.5 s
[TA] = 0.1 s for pressure, temperature, conductivity (example only)
[TL] = 0.05 s for pressure, 0.20 s for temperature, 0.35 s for conductivity.

For an Aanderaa RCM4 sampling every 30 minute (nominal), the RAW section would look like:

```
*RAW
  START TIME          : UTC 1996/09/09 19:00:00.000
  TIME INCREMENT      : 0 0 30 0 50  ! (day hr min sec ms)
  NUMBER OF RECORDS   : 19202

  $TABLE: CHANNELS
  !                                     Averaging  (day hr min sec ms)
  !Name                       Raw Units Interval       Time Lag
  !-------------------------- --------- -------------- -------------
   Speed                      ' '       0 0 30 0 50    0 0 -15 0 -25
   'Direction : Geog(to)'     ' '       0 0  0 0  0    0 0 0 0   0
  $END
 ! Reference Time = START TIME + (Rec#-1) * TIME INCREMENT
 !    Values generated in a TIME channel by IOSSHELL CALIBRATION are reference times
 ! Datum Time (the time at the mid-point of an observation) = Reference Time + Tim Lag.
 !    These values are temporarily generated by time-critical IOSSHELL applications
 !    (e.g. harmonic analysis)
 ! Averaging Interval = length of time taken to acquire the observation.
```

4.7 Data Type

The type of the data in the data file can be defined for all channels at once in the DATA TYPE item of the FILE section, or individually by channel in the CHANNEL DETAIL table of the FILE section.

Data type naming conventions will depend upon the data processing system. Programming-language-specific (e.g. FORTRAN) type names may be used, but to be programming language independent, use the IOS HEADER type names: R4, R8, I, I2, I4, C

See the SPECIAL CHANNEL TYPES section for data type options for special channels such as formatted dates, times, latitudes and longitudes.

For BINARY files DATA TYPE should describe the type of the actual data in the data file. For ASCII files DATA TYPE should be the internal type from which the data was written, and thus the type into which it can safely be re-read.

Data processing systems which require that all channels are REAL*4 can check the DATA TYPE item to verify that the file is readable.

If both the DATA TYPE item and CHANNEL DETAIL table are included, any non-null TYPE values in the table will take precedence.

4.8 FORMAT

Data file format information can be stored (1) using the FILE section FORMAT item, or (2 & 3) using the Channel Detail table. If the FORMAT Item is used, it will take precedence. I.e. if you want programs or people to extract format information from the Details table, make sure the FORMAT Item is null (or possibly FREE or DELIMITED).

The three options are defined in detail below:

1. Code a complete, language specific format in the FORMAT item.
```
eg: FORMAT: (F7.2,E15.7,F8.3)
```

2. Code language specific format specifiers for each channel in the DETAIL.FORMAT table column, such that when they are all put together, they form a complete format statement. eg:
```
! NUM   PAD   START WIDTH   FORMAT  TYPE  DECIMAL_PLACES
! ---   ---   ----- -----   ------  ----  --------------
   1    ' '   ' '    ' '     F7.2
   2    ' '   ' '    ' '     E15.7
   3    ' '   ' '    ' '     F8.3
```

   Using language specific format specifiers, allows for special format features, e.g.:
```
   1PE14.7     -     1.234567E+01   instead of 0.1234567E+02
```

   The START column may be coded to allow for extra spaces between fields which are not included in the field width in the format specifiers.
```
! NUM   PAD   START WIDTH   FORMAT  TYPE  DECIMAL_PLACES
! ---   ---   ----- -----   ------  ----  --------------
   1    ' '     2    ' '     F6.2
   2    ' '     9    ' '     E14.7
   3    ' '    25    ' '     F7.3

... which would translate to: (1X,F6.2,1X,E14.7,2X,F7.3)
                          or: (T2,F6.2,T9,E14.7,T25,F7.3)
```

See the caution about using the START column in the START_Column section below.

3. To be programming language independent, code one of the following in the FORMAT column:
```
E - Exponential
F - Floating point
I - Integer
A - ASCII (Character)
```
and code the field width, and the number of decimal places and (optionally) the start column,

```
!  NUM   PAD   START WIDTH   FORMAT  TYPE  DECIMAL_PLACES
!  ---   ---   ----- -----   ------  ----  --------------
    1    ' '    2     6        F      ' '      2
    2    ' '    9     14       E      ' '      7
    3    ' '    25    7        F      ' '      3
```

Note that the TYPE column is not used for formatting. The Decimal_Places column is located after the Type column because Decimal_Places did not exist in version 1 of the Header, and adding it on the end simplified backwards compatibility of header read and write routines!

See the SPECIAL CHANNEL TYPES section for formats for special channels such as formatted dates, times, latitudes and longitudes.

4.8.1 Method Identification

1. If the FORMAT item is coded, it will be used.
2. A DETAIL.FORMAT string longer than 1 character, indicates that it is a complete, language specific format specifier.
3. If the DETAIL.FORMAT string is 1 character long, then the DETAIL.FORMAT, WIDTH, and the DECIMAL_PLACES fields can be used to create a format item.

4.8.2 START Column

If a START field is null, it can be assumed that the data field for the channel follows immediately after the previous field (or in column 1 for the first field).

All the START column information could be omitted if the field widths account for the entire record.

Coding the start column allows a single field to be located without having to look at the data for previous fields. It also allows for extra spaces between data fields without having to exaggerate the format width.

CAUTION: using the START column greatly complicates the manipulation (e.g. re-ordering) of channels in data-processing programs. Before inserting START columns, make sure it will not cause problems in the target data-processing system.

An example of where the START column could be useful is if you have a dense data file (e.g. few or no spaces between fields), and you want to define certain fields to be picked out when the file is read.

4.8.3 DECIMAL PLACES

- `may be null, e.g. for non-floating point format types.`
- `if null for a real data type, assume zero decimal places in the format.`

4.9 MANDATORY ITEMS and SECTIONS

A 'mandatory' item is one which MUST be present in the header. If it is not present, software should produce some kind of error or warning message.

The designation of sections and items as mandatory is a data-processing-system dependent feature.

Sections and Items are specified as mandatory in a 'Mandatory Item File' which is read when a header is checked. Thus the 'mandatory' designation is not hard-coded, and can be adapted to different and changing requirements.

Mandatory data items are not 'mandatory' if the whole section is omitted.

In the IOS FORTRAN implementation, the user can define what sections and items are mandatory, and the programmer must explicitly call the routine which does the checking.

See the PROGRAMMER GUIDE section of this document, and the description of library routine HRC_CHECK_HEADER for more details of how this feature is implemented in the IOS FORTRAN implementation if the IOS HEADER.

4.10 Lookup Items

Lookup Items are defined in the HEADER DEFINITION section of this document. This header feature has not been implemented. I.e. no lookup lists have been created, and no code has been written to check lookup items against lists.

5 EXAMPLE HEADERS

5.1 *Full header*

```
*1996/03/26 14:29:12.12
*IOS HEADER VERSION 1.1 1995/02/17
*COMMENTS
    This sample header shows all possible header items and tables.

    Attributes of each item are listed at the right:
        - the data type (I/R/C/LL/FT/TI)
        - Units
        - Look-up: lu/o if 'open'   look-up (choices on a list,
                                             but others possible)
                   lu/c if 'closed' look-up (MUST be on a list).

*FILE
    START TIME         : PST 1990/09/29  12:30:59.00  ! FT
    END TIME           : PST 1990/09/30  00:45:00.00  ! FT

    TIME INCREMENT     : 0 0 2. 0 -16                 ! 5xR*4
    TIME UNITS         : HOURS                        ! C*12
    TIME ZERO          : PST 1990/09/29  00:00:00.00  ! FT

    NUMBER OF RECORDS  : 1234                         ! I
    DATA DESCRIPTION   : Current Meter                ! C*80
    FILE TYPE          : ASCII                        ! C*8  lu/c
    FORMAT             : (1X,                         ! C*160
           CONTINUED   :  5E14.7,
           CONTINUED   :  I2)
    DATA TYPE          : 'REAL*4'                     ! C*10 lu/o
    PAD                : -1.E32                       ! R*4
    CRC                : A13F094C                     ! C*12
    RECORD LENGTH      : 40                           ! I    bytes
    NUMBER OF CHANNELS : 3                            ! I
    $TABLE: CHANNELS
    !                             ---- optional -----
    ! NUM  NAME          UNITS        MIN        MAX
    ! I    C*40          C*40         R          R
    ! ---- -----------   -----------  ---------  ---------
      1    Time          Days         .81234E0   .12345E1
      2    Temperature   'Degrees C'  2.0        8.3123
      3    Salinity      ?            29.2345    35.3456
    $END

    $TABLE: CHANNEL DETAIL     ! optional table
    !
    !NUM   PAD    START WIDTH    FORMAT  TYPE    DECIMAL_PLACES
    !I     R      I     I        C*8     C*10        I
    !---   -----  ---   ---      -----   ------  --------------
      1     n/a   1     17       E       REAL*8     9
      2    -9999  19    14       E       REAL*4     7
      3    -9999  34     6       F       REAL*4     2
    $END

*ADMINISTRATION
    MISSION     :                                     ! C*10
    AGENCY      : IOS                                 ! C*80  lu/o
    COUNTRY     : Canada                            ! C*70
    PROJECT     : IMP                               ! C*50  lu/o
    SCIENTIST   : Felix Bloggs                      ! C*20  lu/o
    PLATFORM    :                                   ! C*25  lu/o
    $REMARKS                                        ! C*80
        Administration-specific comments can be added here as desired.
        To keep this header looking nice, they would be indented 8 spaces.
        Leading spaces past the leftmost character in any line will be
        preserved.
        No indenting will be done if it would push any line past col 80.
    $END

*LOCATION
    GEOGRAPHIC AREA      : Beaufort Sea             ! C*80  lu/o
    STATION              : AA11-7                   ! C*40
    EVENT NUMBER         : 123                      ! I
    DATUM                :                          ! C*12
    LATITUDE             :  45 30.12345 N           ! LL
    LONGITUDE            : 129 30.12345 W           ! LL
    LATITUDE 2           :                          ! LL
    LONGITUDE 2          :                          ! LL
    UTM ZONE             :                          ! I
    UTM NORTH            :                          ! I*4
    UTM EAST             :                          ! I*4
    UTM ZONE 2           :                          ! I
    UTM NORTH 2          :                          ! I*4
    UTM EAST 2           :                          ! I*4
    WATER DEPTH          : 123.45                   ! R    metres
    ICE THICKNESS        : 1.234                    ! R    metres
    MAGNETIC DECLINATION : 1.2345                   ! R    deg E
    MAJOR AXIS           :                          ! R

*DEPLOYMENT
    LATITUDE             :  45 30.12345 N           ! LL
    LONGITUDE            : 129 30.12345 W           ! LL
    MISSION              :                          ! C*10
    TYPE                 :                          ! C*40
    TIME ANCHOR DROPPED  :                          ! FT
    TIME IN WATER        :                          ! FT

*RECOVERY
    LATITUDE             :  45 30.12345 N           ! LL
    LONGITUDE            : 129 30.12345 W           ! LL
    MISSION              :                          ! C*10
    TIME ANCHOR RELEASED :                          ! FT
    TIME OUT OF WATER    :                          ! FT


*INSTRUMENT
    TYPE                 : ABCDEFG                  ! C*80      lu/o
    MODEL                :                          ! C*20      lu/o
    SERIAL NUMBER        : 12345678                 ! C*20
    DEPTH                : 12.34                    ! R     metres
    REVS PER COUNT       :                          ! I
    ROTOR TYPE           :                          ! C*20      lu/o
    TRACK DESIGNATOR     :                          ! C*10
    DROGUE TYPE          :                          ! C*20
    DROGUE LENGTH        :                          ! R     metres
    DROGUE MID DEPTH     :                          ! R     metres
    DROGUE DIAMETER      :                          ! R     metres

    $TABLE: SENSORS ! (optional, for multi-sensor instruments)
    !   NAME           ABS DEPTH      SERIAL NO
    !   C*40          R             C*20
    !   -----------   ------------- --------
        Temperature    101.           12345
        Xyzmeter       110.           23456
    $END

*HISTORY
    $TABLE: PROGRAMS
        !                                                ---- optional ---
        ! Name                 Version  Date       Time     Recs In  Recs Out
        ! C*32                 C*6      C*10       C*8      I        I
        !------------------   -------  ---------- -------- -------  --------
         CALIBRATION          2.0      1990/09/23 08:30:12 1234567  1234567
         DECIMATION           1.4      1990/09/24 08:23:00 1234567  1234567
    $END
    $REMARKS
        More information about how programs were run, or on manual processes
        (e.g. manual editting) may be added here.
    $END

*RAW          ! describes the original raw data file
    START TIME         : GMT 1990/08/25 12:30:00.000 !  FT
    FIRST GOOD REC TIME: GMT 1990/01/01 12:35        !  FT
    END TIME           : PST 1992/08/01 12:00        !  FT

                       ! day hr min sec ms
    TIME INCREMENT     : 0   0  2.  0   -16          !  5xR
    NUMBER OF RECORDS  : 1235                        !  I
    VOLUME ID          : 1234567890                  !  C*10
    FIRST RECORD       : 'XXXXXXXXXXXX'              !  C*80

    START MANUAL RECORDS  : 3                        !  I
    END MANUAL RECORDS    : 2                        !  I

    $TABLE: CHANNELS  ! must define all channels in the raw data
    !   Name                   Raw Units     Averaging    Time
    !                                        Interval     Offset
    !   C*40                   C*40          TI           TI
    !   -----------            ---------     -----------  -----------
        Temperature            volts         0 0 0 5 0    0 0 0 0 300
        Pressure               counts
        Conductivity           hertz
    $END

*CALIBRATION  ! This section should never change after calibration

    START TIME         : GMT 1990/08/25 12:30:00.000 !  FT
    TIME INCREMENT     : 0 0 2. 0 -16                !  5xR

    TIME UNITS         : HOURS                       !  C*12  lu/c
    TIME ZERO          : PST 1990/09/29 00:00:00.00  !  FT

    RECORDING BITS     : 16                          !  I

    $TABLE: RAW CHANNELS  ! include only those channels requiring calibration

    !   Name            Units      Formula    Pad     Coefficients
    !   C*40 lu/o       C*40 lu/o  I lu/c     R       20xR
    !   -----------     ---------- -------    ----    --------------
        Temperature     Celsius      10       -9999   (1.23 2.34 3.45)
        Pressure        Pounds/Acre  1        -9999   (1.23 2.34 4.56 7.7 3.4E4
                                                       3.45 5.678)
        Conductivity    ?            11       -8888   ( .1234567E1
                                                           .2345678E2
                                                           .1234567E3 )

    $END

    $TABLE: CALCULATED CHANNELS
    !   Name          Units      Formula  Pad  From Channels, Coefficients
    !   C*40 lu/o     C*40 lu/o  I lu/c   R    10xC*40, 20xR
    !   -----------   ---------- ------- ----- --------------------------
        Conductivity    ?        20     -9999  (Pressure Temperature)
                                               (1.23 3.45 6.78)
        Salinity      ppt        44     -9999  (Conductivity Temperature
                                                Pressure)
                                               (1.234)
    $END

    $TABLE: CORRECTED CHANNELS
    !   Name          Units      Formula  Pad  From Channels, Coefficients
    !   C*40 lu/o     C*40 lu/o  I lu/c   R    10xC*40, 20xR
    !   -----------   ---------- ------- ----- --------------------------
        Salinity      ppt        44     -9999  (Conductivity Temperature
                                                Pressure)
                                               [DEMO ARRAY]
                                               (1.234)
    $END

    $ARRAY: DEMO ARRAY
       1.1  1.2  1.3
       2.1  2.2  2.3
    $END
!-------1------- ------2------ --3--
!     Time        Temperature  Sa~ty
!--------------- ------------- -----
*END OF HEADER
```

5.2 Small Header

```
*1992/09/16 17:35:12.23
*IOS HEADER
*FILE
    NUMBER OF RECORDS  : 1234
    FILE TYPE          : BINARY
    DATA TYPE          : REAL*4
    RECORD LENGTH      : 8
    NUMBER OF CHANNELS : 2
    $TABLE: CHANNELS
    ! NUM  NAME               UNITS
    ! ---- -----------        -----------
      1    Depth              Metres
      2    Temperature:Water  Celsius
    $END
*END OF HEADER
```

5.3 CTD Header

```
*1993/08/12 13:37:51.95
```

```
*IOS HEADER VERSION 0.4  1992/11/17

*FILE
    START TIME          : UTC 1992/09/10 05:02:01.000
    NUMBER OF RECORDS   : 135
    FILE TYPE           : ASCII
    FORMAT              : (f11.3,f9.5 ,f9.6 ,f8.4 ,f5.0 )
    DATA TYPE           : REAL*4
    RECORD LENGTH       : 37
    NUMBER OF CHANNELS  : 5

    $TABLE: CHANNELS
    ! No Name                    Units           Minimum        Maximum
    ! -- ----------------------- --------------- -------------- --------------
       1 Pressure                DBAR            1.35           134.675
       2 TEMPERATURE             'DEG C (ITS68)' 7.17299        12.53264
       3 CONDUCTIVITY_RATIO      n/a             0.798218       0.876783
       4 SALINITY                PSS-78          32.2586        33.658
       5 Number_of_bin_records   n/a             3              13
    $END

*ADMINISTRATION
    MISSION             : 92-15
    PROJECT             : WOCEPR5&6

*LOCATION
    STATION             : MP01
    EVENT NUMBER        : 1
    LATITUDE            :  48  30.76000 N  ! (deg min)
    LONGITUDE           : 124  42.10000 W  ! (deg min)
    WATER DEPTH         : 120

*INSTRUMENT
    TYPE                : Guildline CTP
    MODEL               : 8705
    SERIAL NUMBER       : 58,483

    $TABLE: SENSORS
    !   Name                  Abs Depth       Serial No
    !   --------------------  --------------  ----------
        Pressure              n/a             ?
        Temperature           n/a             ?
        Conductivity_Ratio    n/a             ?
    $END

*HISTORY

    $TABLE: PROGRAMS
    !   Name     Vers   Date       Time     Recs In Recs Out
    !   -------- ------ ---------- -------- ------- -------
        RAW_CNVT 3.0    1993/05/19 11:08:12 ' '     ' '
        CALIB    4.0    1993/05/21 11:28:00    5347    5347
        DESPIKE  1.0    1993/06/13 14:52:12    5347    5347
        TIMECOMP 2.0    1993/06/14 10:55:59    5347    5347
        DELETE   4.0    1993/06/15 10:08:56    5347    1264
        CTDEDIT  3.11   1993/06/28 13:42:11    1264    1264
        METERAVE 2.0    1993/07/05 12:10:46    1264     135
        CALIB    5.0    1993/08/12 13:37:51     135     135
    $END
    $REMARKS
        -The following DESPIKE parameters were used:
                                        DESPIKE TABLE
         CHANNEL                FIT   OVER-  MIN     MAX    MIN     MAX     SPIKE
```

```
                          WIDTH LAP    VALUE   VALUE  STDDEV  STDDEV  TOL.
      -------------------- ----- ----- ------- ------- ------- ------- -----
      PRESSURE                25     5    0.00 3500.00  0.1000 30.0000  2.50
      TEMPERATURE             25     5    0.00   16.00  0.0010  3.0000  2.50
      CONDUCTIVITY            25     5    0.50    1.00  0.0005  0.0050  2.50
     -The following TIMECOMP parameters were used:
      Temp. probe Dist (m):  0.0700  Sample period:  0.0400
     -The following DELETE parameters were used:
      Swells deleted
     -The following METERAVE parameters were used:
      Averaging interval =    1.0 .  Bin type = press
      Average press  was used
      Interpolated values were NOT used for empty bins
     -The following CALIB parameters were used:
      Calibration type = C
      Pressure offset =        0.0
      Calibration file = COR_9215.DAT
    $END

*RAW
    START TIME          : UTC 1992/09/10 05:02:01.000
    END TIME            : UTC 1992/09/10 05:05:33.000
    TIME INCREMENT      : 0 0 0 0.4E-01 0  ! (day hr min sec ms)
    NUMBER OF RECORDS   : 5347

    $TABLE: CHANNELS
    !                                      Averaging  (day hr min sec ms)
    !Name                 Raw Units       Interval       Time Offset
    !-------------------- --------------- -------------- -----------
     Pressure             Percent_FS
     Temperature          'DEG C'
     Conductivity_Ratio   n/a
    $END

*CALIBRATION
    START TIME          : UTC 1992/09/10 05:02:01.000
    TIME INCREMENT      : 0 0 0 0.4E-01 0  ! (day hr min sec ms)

    $TABLE: RAW CHANNELS
    !   Name                 Units Fmla Pad    Coefficients
    !   -------------------- ----- ---- ------ ------------
        PRESSURE             n/a     10 ' '    (-1 3000)
        TEMPERATURE          n/a     10 ' '    (0.433E-02 0.9995)
        CONDUCTIVITY_RATIO   n/a     10 ' '    (-0.33E-03 1)
    $END

    $TABLE: CALCULATED CHANNELS
    !   Name       Units    Fmla Pad    Coefficients
    !   ---------- -------- ---- ------ ------------
        SALINITY   PSS-78     30 ' '    (PRESSURE TEMPERATURE
                                         CONDUCTIVITY_RATIO)
                                        ()
    $END

    $TABLE: CORRECTED CHANNELS
    !   Name          Units Fmla Pad    Coefficients
    !   ------------- ----- ---- ------ ------------
        TEMPERATURE   ' '     11 ' '    (Pressure)
                                        (0.54E-02 0)
        SALINITY      ' '     11 ' '    (Pressure)
                                        (0.144928E-01 0.378645E-05)
    $END
```

```
*COMMENTS
    MARIE, REG, FIRST CTD
*END OF HEADER
```

5.4 Current Meter Header

Following is the header of a Current Meter file, converted from an Arctic group Multi-Line header:

```
*1991/02/13 16:25:39.03
*IOS HEADER VERSION 1.0  1993/07/15 1993/08/16

*COMMENTS
     - SET-UP
         Comments: Batt #1 = 11.34V.

     - DEPLOYMENT INFORMATION
         First record time/date (GMT)     : Centered around 18:43
         First record printout            :
         Other printout                   :
         Geographical description         :
         Instr. in water time/date (GMT)  : 00:58 March 18   17:58 (MST)
         Mooring complete time/date (GMT) :     01:55 March 19
         Comments: Power on 18:41:00  March 18, 1987
                   Topham/Pite Red Mooring.  Position: 3rd from bottom.
                   Direction reference completed at MAG N.  Meter cold during
               start-up due to a one minute mental lapse.  This meter is
               taking data in between the A2 times.  Mooring put in twice,
               too long first time.  Meters dries out in tent in between.
               Installed in mooring rotor DOWN.

     - RECOVERY INFORMATION
         Release time/date (GMT)     :
         Instr. out time/date (GMT)  : 17:43:00 Apr 2, 1987
         Last record printout        :
         Last record time/date (GMT) :
         Current meter condition     :
         Comments: Not running on recovery.  Batts=9.8V.  2/3 of tape used.

     - EDITING INFORMATION
         Comments:

*ADMINISTRATION
    PROJECT             : Beaufort Sea Moorintg 1987
    $REMARKS
        Orginal project name: Beaufort Sea Moorintg 1987
    $END

*LOCATION
    GEOGRAPHIC AREA     : Beaufort Sea
    STATION             : RED
    LATITUDE            :  70  32.76000 N  ! (deg min)
    LONGITUDE           : 127  49.30000 W  ! (deg min)
    WATER DEPTH         : 20.12
    ICE THICKNESS       : 1.35

*INSTRUMENT
    TYPE                : VACM
    MODEL               : VACM
    SERIAL NUMBER       : 217
    DEPTH               : 14.23
    $REMARKS
        Secondary sensor (sensor depth in meters): 1.4230000E+01
    $END

*RAW
    NUMBER OF RECORDS   : 10789

    $TABLE: CHANNELS
    !   Name                    Raw Units
    !   --------------------    ----------
        'TIME : CODE'           HOUR
        TEMPERATURE             ' '
        'CONDUCTIVITY RATIO'    ' '
        SPEED                   ' '
        DIRECTION               ' '
    $END

*CALIBRATION
    RECORDING BITS      : 8

    $TABLE: RAW CHANNELS
    !   Name                 Units  Fmla Pad    Coefficients
    !   -------------------- ------ ---- ------ ------------
        'TIME : CODE'        HOUR     10 ' '    (0 0.2777778E-03)
        TEMPERATURE          ' '      31 ' '    (0.753445E-05 0.42217E-08
                                                 -29.3502 3901870.
                                                 -0.124821E+12 0.182982E+16)
        'CONDUCTIVITY RATIO' ' '      60 ' '    (-0.793939E-02 0.397433E-05
                                                 102.4)
        SPEED                ' '      41 ' '    (200 3.5 0.787 16 2.5 0 1
                                                 0.912E-01)
        DIRECTION            ' '      52 ' '    (366.3897 -1.391829
                                                 -0.291081E-03 0.171617E-04
                                                 -0.295444E-06 0.214475E-08
                                                 -0.701395E-11 0.850592E-14)
    $END

    $TABLE: CALCULATED CHANNELS
    !   Name       Units     Fmla Pad    Coefficients
    !   ---------- --------- ---- ------ ------------
        SALINITY   ' '         30 ' '    (TEMPERATURE 'CONDUCTIVITY RATIO')
                                         ()
        DENSITY    kg/m**3     40 ' '    (TEMPERATURE SALINITY)
                                         ()
    $END

*HISTORY
    $REMARKS
        Files are processed with the following programs before conversion:
        Program CREATE_HEADER Ver. 2.0 YY/MM/DD HH:MM:SS  91/ 2/13 14:43:38.95
        Program UPDATE_RECORDS Ver 1.0 YY/MM/DD HH:MM:SS  91/ 2/13 16:25:39.03
    $END

*FILE
    START TIME          : GMT 1987/03/18 18:43:00.000
    TIME INCREMENT      : 0 0 2 0 0  ! (day hr min sec ms)
    NUMBER OF RECORDS   : 100
    FILE TYPE           : ASCII
    FORMAT              : (5I5)
    RECORD LENGTH       : 255
    NUMBER OF CHANNELS  : 5

    $TABLE: CHANNELS
    ! No Name                 Units  Minimum        Maximum
    ! -- -------------------- ------ -------------- --------------
```

```
       1 'TIME : CODE'        HOUR   ' '            ' '
       2 TEMPERATURE          ' '    ' '            ' '
       3 'CONDUCTIVITY RATIO' ' '    ' '            ' '
       4 SPEED                ' '    ' '            ' '
       5 DIRECTION            ' '    ' '            ' '
    $END
*END OF HEADER
```

6 SPECIAL CHANNEL TYPES

6.1 *Introduction*

A "special channel" is a channel which is of a non-Real type. A special channel is identified by its **Type** in the Channel Details table. For example a channel with type **LAT** is coded in the data file as a latitude in degrees, minutes and a hemisphere: 49 35.123 N.

(Note that an Integer channel is considered a "special" channel type, unless *all* channels are integers.)

This section describes schemes and options for describing Latitude, Longitude and various forms of date and time channels in the header. Also described are the I/O routines in the Fortran library which can read and write such data, and convert it back and forth from REAL numbers, or otherwise communicate data values with Fortran programs.

Using special channel types may make the data file easier for a person to read, but it means that programs which use the file must use the header library I/O routines described at the end of this section. The regular Header library I/O routines (e.g. HR_READ_DATA_R4) detect the presence of special channel types, and use the special I/O routines. For each special channel type, the programmer must be aware of how and where the special channel data is stored.

6.2 *Types and Formats*

The types and formats of each such special channel are coded in the TYPE and FORMAT columns of the Channel Details table. The following table summarizes the options:

```
Type                Format
----                ------
I     Integer       null         - I is assumed
                    I            - width (and decimal places - see Iw.d below)
                                   will come from their detail table columns.
                    Iw           - eg I5 - Width taken from format,
                    Iw.d         - d forces leading zeros, eg 12 using
                                   format I4.4 will yield 0012
C     Character     null         - output: quotes written only if there are
                                   embedded blanks in the string.
                                 - input: quotes are removed if present.
                    Q            - forces single quotes on all output strings
                    NQ           - no quotes written on output, no quotes
                                   looked for or removed on input.  No left-
                                   justification on input or output
                                 - 1st character in an input field is assumed
                                   to be a leading space, and is ignored.
                                 - written with one leading space.

LAT   Lat/Lon       DMH          Degrees, Minutes, Hemisphere
LON                                e.g.  49 12.12 N
                    DMSH         Degrees, Minutes, Seconds, Hem.
                                   e.g.  49 12 20.12 N
                                 Decimal places:
                                    Input:  variable
                                    Output: specify in the Decimal
                                            Places column.
                                            - default is 6

FT    Fulltime      *            ZON yyyy/mm/dd hh:mm:ss.sss
                                 * format is ignored on input
                                 * a time format may be coded to reduce
```

```
                                       detail on output (e.g. HH:MM)
                                     * do NOT include a date format: it is fixed
                                       as YYYY/MM/DD
                                     - if no format, Decimal Places, if coded,
                                       will be used for seconds.
                                       e.g. setting D.P. = 1 is equivalent to
                                       coding format HH:MM:SS.S

   D      Date         *             * - any IOSLIB date format
                                       - see IOSLIB routine SET_DATE_FORMAT
                                       - default is YYYY/MM/DD
                                       - see examples below

   T      Time         *             * - a time format mask
                                       - default is HH:MM:SS.SSS
                                       - see examples below

   DT     Date & Time n/a            - like a fulltime, without the zone
                                     - time zone of item FIL.TIME_ZERO is assumed
```

6.3 Integer Data

Code a **Type** of `I` in the channel detail table to define a channel of Integer data.

The **Width** of the field may specified either in the Width column of the channel detail table, or as part of the Format (as in Fortran I - format descriptors). The width should include any desired leading spaces.

Leading zeros can be forced on output by specifying a number in the Decimal Places column, or in the Format. If Width is 5 and Decimal Places is 4 (or the Format is `I5.4`), then the number 12 would be written as " 0012".

The IOS Header Fortran library routines which do special channel I/O (HRC_READ/WRITE_DATA_SPECIAL_R4/8) impose a limit on the magnitude of the integer, because the integers are actually stored in REAL variables. (Storing in the REAL data array greatly simplifies handling of the data, in the library routines, and in users' programs.) When using the R4 routines, the limit is 16,777,000. When using the R8 routines, the limit imposed is that of INTEGER*4 storage: 2,147,483,647

Example: If channel 3 is an Integer channel, then data(3) returned from HRC_READ_DATA_R4 will contain the value stored as a real number. Thus the value can be used in Real expressions just as if the channel was a Real channel. If the value in Integer form is needed in the Fortran program, it can be obtained with the expression: NINT(data(3)).

As of November 2002, the IOS Header library _READ_DATA and _WRITE_DATA routines have an I4 entry in addition to the R4 and R8 entries. They also handle files with all-integer channels which have an all-integer FORMAT item and no details table. Thus an integer channel is not treated as a "special" channel if all channels are integer.

6.4 Character Data

Code a **Type** of C in the channel detail table to define a channel of character data.

Under most circumstances a **Width** should also be coded in the details table. Determine the minimum value for the Width as follows:

- the maximum length of any string to be stored in the channel
- plus 1 for a leading space (required for all special channels)

- if the Format is NOT **NQ**, add 2, for 2 quote characters.

If no Width is coded, the IOS Header data read routines will still be able to read the data (if strings with embedded blanks are quoted), but when the data is written, no column headings can be produced, and the data will be 'free format' - meaning that the data will not appear in columns.

The **Format** field of the details table may be Q, NQ or blank.
The following table defines how the Format affects character channel input and output, for fixed-format I/O. The data field for a channel is described using the following byte-pointers:

S	=Start if coded, else the first byte past the end of the previous channel's field.
F	S + 1
L	S + Width -1

Format	**Input String**	**Input Behaviour**	**Output Behaviour**
blank	quoted	- bytes S to L are scanned - string inside quotes is extracted, - leading blanks preserved, - embedded blanks preserved, - trailing blanks removed. - quotes removed	- quoted only if there are any leading or embedded blanks. - first quote or character goes in byte F.
	unquoted	bytes S to L are extracted and left-justified. - embedded blanks preserved. - leading and trailing blanks removed.	- quoted only if there are any embedded blanks. (Leading blanks will have been removed on input.) - first quote or character goes in byte F.
Q	quoted	same as for blank format, quoted	quoted, 1st quote at F
	unquoted	same as for blank format, unquoted	quoted, 1st quote at F
NQ		bytes F to L are extracted - quotes are treated like any other character. - leading & embedded blanks, and quotes (if any) are all preserved. - trailing blanks removed.	the string is inserted starting at byte F. No quotes are added. The string is truncated, at, or blank filled to byte L, as is necessary. (Thus, if the channel width is not changed, trailing blanks are restored.)

Leading Space

Note that byte S - the byte reserved for a leading space in special chanels - is never written to. It is only scanned for input when leading spaces are going to be removed anyway.

Quoted output

When writing quoted character channel data, the first quote is put at byte F. The last quote is put after the last character. (No right trimming is done at this stage, but it may have been done at input time.) If that would put the last quote past byte L, then the string *inside the quotes* is truncated so that the last quote goes in byte L. (See routine HRC_WRITE_DATA_SPECIAL_R4.)

Accessing Character Data in Fortran Programs

To write records with character data using the IOS Header library data I/O routines, call routine HR_PUT_CDATA once for each character channel before each call to HRC_WRITE_DATA_R?.

When reading, (with HRC_READ_DATA_R?), all the character data for the record is put into variable HR_CDATA in the common defined in file HR_CDATA.INC. Character data for individual channels is obtained by calling HR_GET_CDATA once for each character channel, after the HRC_READ_DATA_R? call.

(See OPHYSLIB routines WRITE_SCRATCH4 and READ_SCRATCH4 for storing records with character data in an unformatted scratch file.)

6.5 Date Formats

Following are some examples of the possibilities:

Format	Example
YYYY/MM/DD	1995/12/31
YYMMDD	951231
DD-Mon-YY	31-Dec-95
'Month DD, YYYY'	December 31, 1995

Mask characters are case-sensitive.

See IOSLIB routine SET_DATE_FORMAT for details.

6.6 Time Formats

Time formats are specified as a mask using H, M or S, with separators.

```
Examples:   HHHH:MM:SS     - lots of hours, no fractional seconds
            HHMMSS         - no separators
            HH-MM-SS       - alternate separators
            HH:MM:SS.SSS   - three dec places of seconds (DEFAULT)
            HH:MM          - no seconds
            hh:mm          - not case sensitive
```

6.6.1 Input

If no format is specified, the default HH:MM:SS.SSS is assumed.
On input, the data may be a truncated version of the format.
If there is more accuracy in the data than in the format, the extra digits are ignored, but no problems are caused.
Decimal_Places (in the details table) is ignored.

Examples:

```
Time (Data)     Format          Internal, after read
-------------   ------------    --------------------
14:59:59        HH:MM:SS.SSS    14:59:59.000
14              HH:MM:SS        14:00:00.000
14:59:59.55     HH:MM:SS.SS     14:59:59.550
14:59:59.55     HH:MM:SS        14:59:59.000
```

6.6.2 Output

When an abbreviated format is specified, the time is rounded and normalized before encoding for output.

Decimal_Places (in the details table) is ignored if the Format is specified. If the Format is NOT specified, then Decimal_Places is used to format the Seconds portion of the output time.

Examples:

```
Time internal   Format         Dec_Places    Output
-------------   ----------     ----------    ----------
14:59:59.5555   HH:MM          (ignored)     15:00
14:59:59.5555   HH:MM:SS       (ignored)     15:00:00
14:59:59.5555   HH:MM:SS.S     (ignored)     14:59:59.6
14:59:59.5555   HHH:MM:SS.S    (ignored)     014:59:59.6
14:59:59.5555   <null>          0            14:59:59
14:59:59.5555   <null>          2            14:59:59.56
14:59:59.5555   <null>          <null>       14:59:59.556
14:59:59.5555   <null>          3            14:59:59.556
```

6.6.3 Limitations

- only the seconds may have a decimal point, e.g. HH:MM.MMM is NOT allowed.
- a single H, M or S is not allowed, e.g. HH:MM:S is INVALID.

6.7 START and WIDTH

The start column and width information in the channel details table is used by the I/O routines if it is coded. Otherwise, on input the data is scanned for in a non-column-dependent manner, and on output, default widths are used.

Specify widths large enough to include a leading space.

Note that free-format (non-column-dependent) scanning for input data fields is possible only if a) there are special channels, or b) if the FORMAT item in the header is specified as FREE.

```
START
   Input:
      - if coded, scanning starts at this column
      - if null, scanning starts after the end of the previous
        field.
   Output:
      - if coded, this is the first column of the output field.
      - if not coded, output starts after the end of the
        previous field.

WIDTH
   Input:
      - if coded, only this number of columns are scanned from
        the start column; UNLESS: it is not possible to determine
        the start column of the field, (e.g. if a preceding width
        was missing) in which case the field is read as if the
        width was NOT coded.
      - if not coded, the data record is scanned, using blanks as
        delimiters, until all the required data for the field is found.
   Output:  (for non-integer special channels)
      - if coded less than the data width plus one, the output field
        will be written with one leading space, and truncated on
        the right.
      - if coded greater than the data width plus one, the output field
        will be written with one leading space, and trailing spaces
        will be inserted to fill to the specified width.
      - if NOT coded, the field will be written after the previous
        field, with one leading space,

        Note: "data width" is the non-blank extent of the data string,
              e.g. for a time 12:30, the data width is 5.

   Output of Integer channels:
      - this works just like Real channels: the output is written right-
        justified in the specified width, using the entire width if
        necessary.
```

6.8 Converting Date & Time to/from REAL

6.8.1 Reference Time

A "Reference Time" may also be referred to as "Time Zero", or, in IOSLIB terminology, a "Zero Consecutive Day", which may be abbreviated to "Zero Conday".

Dates, and combined Date_Times are converted to/from real numbers in a data vector as a time interval in days, by the Header library routines HRC_READ_DATA... and HRC_WRITE_DATA...

The reference time for the time intervals is set by routine HRC_SET_ZERO_CONDAY, which works as follows:

- If the TIME ZERO item in the FILE section of the header is specified, it is used as the reference time.
- If item TIME ZERO is NOT specified, the IOS Header library "Fixed Time Zero" (FTZ) is used. (Prior to 2003-06-26 the current IOSLIB zero conday was used.) FTZ is hard-coded as 1900-01-01 00:00 in header library routine HR_SET_FIXED_ZERO_CONDAY. That routine simply sets the IOSLIB *zero consecutive day* to 1900-01-01. (It is important to be aware of this - if you are using the IOSLIB *zero consecutive day* for other purposes in your program, you may have to re-set it after using IOS Header routines.)

For combined Date_Time channels, (type FT or DT) both the date AND the time of header item TIME ZERO are used. For example, if the TIME ZERO item contained 1990-01-01 12:00, then 1990-01-03 00:00 in a DT channel would be represented by the real value 1.5.

For Dates (type D), only the date portion of the TIME ZERO header item is used.
For simple Time channels, no reference calculations are done. The time data is converted into a number of days. E.g. 12:00:00 is converted to 0.5. For example, if time zero is 1990-01-01 12:00, the date channel contains 1990-01-02, and the Time channel contains 12:00, then the real number for the date channel will be 1.0, and for the time channel 0.5.

Note that a header time zero can include a time, but the fixed time zero can not.

6.8.2 Time Zone

The reference time zone is determined as follows:

```
Item TIME ZERO coded   Item START TIME coded    reference zone
--------------------   ---------------------    --------------
    yes                    (ignored)            TIME ZERO zone
    no                     yes                  START TIME zone
    no                     no                   null
```

For FullTime channels (where zone, date & time are included in one channel), the time in the data vector will be converted to the reference zone if necessary (if the reference zone is not null).

For Date, Time and Date_Time channels (no zone included), it is assumed that the data is in the reference zone - i.e. no time zone conversions are done.

In all cases, the data put into the structure DATA_FT in common FTLATLON will be exactly as in the data file: no conversions will be done.

6.9 Header Library I/O Routines

The standard I/O routines HRC_READ_DATA_R4 and HRC_READ_DATA_R8 check to see if there are any special channels (by looking in the TYPE column of the Details table) and automatically call the special I/O routines if necessary.

If you use these routines, reading data with special channels will not require any changes to your programs, as long as sufficient accuracy can be maintained. (E.g. if you are using REAL*4 data arrays, there may be unacceptable accuracy loss in the conversion to REAL*4 - depending upon the TIME ZERO.)

You also have the option of calling the special I/O routines directly from your program:

```
          HRC_READ_DATA_SPECIAL_R4
```

```
        HRC_READ_DATA_SPECIAL_R8
        HRC_WRITE_DATA_SPECIAL_R4
        HRC_WRITE_DATA_SPECIAL_R8
```

Function HRC_SPECIAL_CHANNELS can be used to determine whether there are any special channels defined in a header.

Routine HRC_DEFAULT_FORMAT can be used to fill in the details table with default format information, using the TYPE information.

See the HR_TS and HR_TS8 GET and PUT routines for alternate ways of extracting and inserting data record times. These are summarized in the **Fortran Programmer's Guide** / **Time Series Handling** section of this manual.

6.10 Common Block for Data Structures

A common block called FTLATLON contains header structures for a Fulltime, a latitude, and a longitude; named DATA_FT, DATA_LAT and DATA_LON.

In addition to converting any fulltime, date, time, lat or lon channels to real numbers to be returned in the parameter vector of the read routine, the data is put into the structures in the common block. The user's program can include the same common block to access the data. This may be desirable because:

- it will be the only way of getting the time zone of a fulltime channel
- a single precision real number may not contain enough accuracy for dates and times.

Data of separate Date and Time channels are put into the fulltime structure.

This method of getting data imposes the limitation of only one each of date, time, latitude and longitude channels per data record. If there are more than one channels of a type, then the *first* channel of that type which will appear in the structure in common. (Prior to 20-May-2003, it was the last channel of each type which was put into the structure in common.)

When reading data, the special channel data is available both as a real number in the data vector, and via the data structures. For writing, however, the write routine writes from the data vector unless the value for the channel is the header NULL_ABSENT value, in which case the data is taken from the structure in common.

To set element one of REAL array DATA to null_absent:

```
      INCLUDE 'nulvals.inc'
      DATA(1) = HN_ABSENT
```

For related information, see the **Time Series Handling** section in the **Fortran Programmer's Guide** chapter of this document.

*6.11 Z8 Real*8 Times*

A "Z8" time is a number of days from the IOS Header library Fixed Time Zero (FTZ), converted to the time zone UTC. This method of storing times is useful for coordinating data from multiple data files, since it is independent of header time zeros and time zones.

See the HR_*Z8* routines for obtaining and working with Z8 times.. These are summarized in the **Fortran Programmer's Guide** / **Time Series Handling** section of this manual.

6.12 Example

Suppose a data file with a Fulltime, a Latitude, a Longitude, Temperature, an integer flag and a character station name:

A data record may look like this:
(The numbers underneath are column numbers)

```
     PST 1995/08/29 12:20:10.001 49 30.00 N 125 24.12 W  1.1234  3  STNA
    1                            2          4           5       6
                                 9          0           2       0
```

The header channel and detail tables could be coded as follows:

```
    $TABLE CHANNELS
      1  Time         n/a
      2  Latitude     n/a
      3  Longitude    n/a
      4  Temperature  'Degrees C'
      5  Flag         n/a
      6  Station      n/a
    $END

    $TABLE CHANNEL DETAIL
    !Num   Pad   Start Width  Format  Type  Decimal_Places
      1    n/a    ' '   28     n/a     FT
      2    n/a    ' '   11     DMH     LAT    2
      3    n/a    ' '   12     DMH     LON    2
      4    -99    ' '   8      F       R4     4
      5    -1     ' '   3      I       I
      6    n/a    ' '   6      NQ      C
    $END
```

Alternatively the Channel detail table could be coded with the minimum required information, as follows:

```
    $TABLE CHANNEL DETAIL
    !Num   Pad   Start Width  Format  Type  Decimal_Places
      1    ' '    ' '   ' '    ' '     FT
      2    ' '    ' '   ' '    ' '     LAT
      3    ' '    ' '   ' '    ' '     LON
      4    ' '    ' '   ' '    ' '     R4
      5    ' '    ' '   ' '    ' '     I
      6    ' '    ' '   ' '    ' '     C
    $END
```

The FORTRAN code to read a record and access the data as structures:

```
      RECORD/SR_FULLTIME/ DATIM         ! declare structures
      RECORD/SR_LATLON/   LAT, LON
      REAL*4              DATA(6)
      INTEGER             IFLAG
      CHARACTER*6         STATION

C     INCLUDE 'ftlatlon.inc'                      !dvf  !unix
      INCLUDE 'ios_header$inc:ftlatlon.inc'       !vax
```

.... header opening and reading code....

```
C     --- Read a record, and return 5 reals in DATA ---
      CALL HRC_READ_DATA_R4(U,RN, DATA, RSTAT)
```

```
C     --- Copy special channel data as structures from common ---
      DATIM = DATA_FT
      LAT   = DATA_LAT
      LON   = DATA_LON
C     --- Copy out the integer and character data ---
      IFLAG = NINT(DATA(5))
      CALL HRC_GET_CDATA(NINT(DATA(6)),STATION)
```

Note that if you want to use time, lat, lon as real numbers instead of structures, it is recommended that you declare DATA as REAL*8, and use the HRC_READ_DATA_R8 routine, to avoid accuracy loss.

7 UTILITY PROGRAMS

7.1 Introduction

The IOS_HEADER utility programs' files are located in VMS with the following logical names:

```
    IOS_HEADER$UTIL     - source code and .COM files
    IOS_HEADER$BIN      - executables.
```

All files for the DOS versions are in IOS_HEADER$DOS.

In the following sections, the logical name for locating the command procedures has been omitted from the examples. So instead of typing
"@HEADCOPY", enter "@IOS_HEADER$UTIL:HEADCOPY".

7.2 HEADREAD

HEADREAD just reads a header, allowing you to verify that the header is correct. Error and warning messages are written to unit 6.

To use:

```
    VMS:  @HEADREAD <infile> [MAND]
    WIN:  HEADREAD  <infile> [MAND]

where:
     <infile> - is the file to read.  Data following the header will be
                ignored.
     MAND     - (optional) will cause a mandatory item check, using the
                mandatory item file named in:
                   VMS:  logical:  IOS_HEADER$MANDITEM
                   WIN:  env. var: MANDITEM
              - see routine HRC_CHECK_HEADER for details.
```

7.3 HEADCOPY

HEADCOPY reads and writes a header, and copies any data which follows the header. This may be useful if the header was entered manually, and you want it formatted in the standard manner of the IOS HEADER write routine.

To use:

```
    VMS:  @HEADCOPY <infile> [<outfile>]
    WIN:  HEADCOPY  <infile> <outfile>

    where:
        <infile>  - is the name of the input file
        <outfile> - is where the output is to go.
                    If omitted (VMS), a new version of <infile> is created.
```

7.4 HEADBLNK

HEADBLNK creates a header containing all possible sections and items. All items are set blank. All tables have one blank row.

The resulting blank header file may be useful for checking what is in the latest version of the header. It would also be useful for entering header data manually (with an editor) by filling in the blanks. Unwanted sections, items and tables could just be deleted.

To use:

```
    VMS:  @HEADBLNK <outfile>
    WIN:  HEADBLNK  <outfile>

    where:
        <outfile> - is where the output is to go.
                    If "*", output is to the screen.
```

You might want to take a copy of HEADBLNK.FOR, and modify it so that it does not produce unwanted sections. For example, if you want all sections except DEPLOYMENT and RECOVERY, insert the following statements in the program:

```
    INCLUDE 'IOS_HEAER$INC:HEADER_COMMON.INC'    ! near the top

    H.DEP.INCLUDE_OUT = .FALSE.      ! before HRC_WRITE_HEADER
    H.REC.INCLUDE_OUT = .FALSE.
```

7.5 HEADEDIT

This program allows the contents of a header (called an 'Edit' header) to be added to an 'Input' header, producing an 'Output' header.

This is useful if you want to add or replace one or more items, tables, or remarks in a series of headers.

Note the following:

- custom items may also be added/replaced
- tables are added/replaced in their entirety. Modifications within tables can not be done.
- see IOS Shell program HDREDIT2 for an enhanced version of HEADEDIT which allows modifications to channel names and units.
- remarks for each header section can be either replaced, or added to.
- if the 'Input' header is followed by any data, it is copied to the 'Output' file as well.

If the Input header contains items or tables which are also in the Edit header, those items and tables will be REPLACED with the information from the Edit header.

If the Edit header contains remarks, then if the first remark in each section is:

`APPEND`	- the remarks will be appended to those in the Input header
`REPLACE`	- the remarks will replace those in the Input header
`else`	- the remarks will be appended to those in the Input header

The APPEND or REPLACE keyword may be anywhere in the first remark line of the section, but must be upper case, and must be the only characters on the line.

HEADEDIT uses subroutine HR_EDIT_HEADER in the IOS_HEADER library. See the documentation for that subroutine for more details of how to code the 'Edit' header.

7.5.1 Using in VMS

To edit a single file in VMS:

```
    @HEADEDIT [-P] [-T] <edfile> <infile> <outfile>

    Where:
        -P        - (optional): add HEADEDIT to the program table
        -T        - (optional): update the time stamp
                    (-P and -T may go anywhere in the command line)
        <edfile>  - is the header containing the new or replacment
                    header information.
        <infile>  - is the file containing the header to be edited.
        <outfile> - (optional) is where the edited output is to go.
                    If omitted, a new version of <infile> is created.
```

To edit a set of files specifiable in a single VMS file specification (e.g. *.1MA):

```
    @HEDITALL [-P] [-T] <edfile> <inspec> [<outext>]"

    Where:
        -P        - (optional): add HEADEDIT to the program table
        -T        - (optional): update the time stamp
                    (-P and -T may go anywhere in the command line)
        <edfile>  - is the header containing the new or replacement
                    header information.
        <inspec>  - specifies all the files to be edited.
        <outext>  - (optional) the file name extension to be used
                    for the edited files.  (preceding "." optional)
                    If omitted, an extension of .HDT will be used
```

Note that HEDITALL.COM calls HEADEDIT.COM for each file to edit. Thus the edit file is read repeatedly.

7.5.2 Using from the Windows command line

To edit a single file

```
    HEADEDIT [-P] [-T]   <edfile> <infile> [<outfile>]

    Where:
        -P        - (optional): add HEADEDIT to the program table
        -T        - (optional): update the time stamp
                    (-P and -T may go anywhere in the command line)
        <edfile>  - is the header containing the new or replacment
                    header information.
        <infile>  - is the file containing the header to be edited.
        <outfile> - (optional) is where the edited output is to go.
                    If omitted, a new version of <infile> is created."
```

No Batch file has been written for multiple edits. The "FOR" command could be used for multiple execution of HEADEDIT in a batch file.

7.5.3 Customizing HEADEDIT

In some cases, you may want to:

- avoid repeated Edit file reads,
- make modifications within a header table,
- control file name input and generation from within a FORTRAN program.

In any of these cases, it would be fairly easy to take a copy of the HEADEDIT.FOR file, change the file naming and I/O code, and/or add special code to modify header tables.

8 FORTRAN PROGRAMMER's GUIDE

8.1 Platforms

The IOS Header has been implemented in the DOS, UNIX, VMS (VAX and AXP) and Win32 environments at the Institute of Ocean Sciences. The DOS and UNIX versions are no longer being maintained, however.

8.2 Environment Setup

8.2.1 VMS

In VMS, a set of IOS Header logical names must be defined before compiling, linking or running programs which use the IOS_HEADER library. These logical names can be set by issuing the following command:

```
@CCS$PHYSICS2:[PHYSIC.LIBRARYS.IOSHEADR.VMS.FORTRAN]LOGICALS
```

Following are some of the logicals defined:

```
IOS_HEADER$INC  - search path for IOS Header include files
IOS_HEADER$UTIL - utility programs
IOS_HEADER$DOC  - documentation
IOS_HEADER$ROOT - root of header directories
                - location of MESSAGES.TXT file
IOS_HEADERLIB   - points to the IOS_HEADER object library.
```

The command SHOW LOG IOS_HEADER* will provide a complete list.

8.2.2 Win32, DOS, Unix

The following environment variables are used by the Win32, DOS and Unix versions of the IOS Header system:

```
HEADPATH - directory containing the messages.txt file.

MANDITEM - needed only if routine HRC_CHECK_HEADER is used, or
           if the MAND option of the HEADREAD utility is used.
         - must contain the complete mandatory item file name.

IOSLIB   - must contain the complete name of the IOSLIB object
           library.
         - used for linking.

HEADLIB  - must contain the complete name of the IOS HEADER
           object library
```

8.3 Compiling

It may be necessary to compile your Fortran with the same compile options used when compiling the header library routines, to ensure that structures are packed the same way in memory.

To see what options were used, look at the following files:

```
VMS:  IOS_HEADER$SRC:F1.COM
       (default alignment is used)

UNIX: $HEADPATH/headenv.csh  (look for the definition of FOPT.)

Win32: \\paciosfp2\osapshare\isutil\dfenv.bat

       With Visual Fortran, use the following:
            /align:dcommons
       and optionally:
            /check:(bounds,overflow,underflow) /warn
```

8.4 Accessing Header Include Files

To include IOS Header include files in your Fortran program:

VMS: Prefix the include file name with the logical name IOS_HEADER$INC, e.g.:

```
INCLUDE 'IOS_HEADER$INC:HEADER_COMMON.INC'
```

Win32:

The IOS Header include files, which define the header structure, are named with the same long names as in VMS. Thus the INCLUDE statements in your Fortran programs will look like the VMS INCLUDE statements, but with no logical name on the include file name.

```
INCLUDE 'header_common.inc'
```

The IOS Header include files are stored with the object library in \\paciosfp2\osapshare\libraries\dvf. (This directory contains other libraries as well.)

The compiler looks for include files first in the current directory, then in the path specified on the /include:path compiler option, then in the path specified in the INCLUDE environment variable.

UNIX:

The include files can be located using two system-defined environment variables: $HEADPATH/$OS_TYPE However, you can not use environment variables in the Fortran include statement, so you will have to define links to the include files from your directory:

```
ln -s $HEADPATH/$OS_TYPE/*.inc .
```

Then the file can be included as if it was in the current directory.

```
INCLUDE 'header_common.inc'
```

8.5 Linking

A program which uses IOS_HEADER library routines must link to the header library, and to the IOSLIB library.

8.5.1 VMS:

```
$LINK your_program, -
    your_libraries, -
    IOS_HEADERLIB/LIB, -
    IOSLIB_UPDATES/LIB, -
    CCS$IOSLIB/LIB
```

See the preceding section to see how to set logical name IOS_HEADERLIB.

IOSLIB_UPDATES is a logical which is defined along with the header logicals, and which points to a library of new routines required by the header routines, but which have not yet been put into IOSLIB.

CCS$IOSLIB is a system logical.

8.5.2 Win32

```
(Digital) Visual Fortran
```

The DVF version of the IOS Header library is located in
`\\paciosfp2\osapshare\libraries\dvf, and is called headlib.lib.`

The MicroSoft linker searches the directories in the path defined in the LIB environment variable, so:

```
    if you have drive o: mapped to \\paciosfp2\osapshare

    SET LIB=c:\your_library_directory;o:\libraries\dvf

    LINK your_program your.lib headlib.lib ioslib.lib
```

The linker accepts wild-cards, so the following would also work:

```
    LINK your_program your.lib o:\libraries\dvf\*.lib
```

It is also possible to compile and link with one df command.
See the Visual Fortran on-line Programmer's Guide for details.

8.5.3 UNIX:

```
    f77 your_program.o $IOSLIB $HEADLIB $IOSLIB
```

The first $IOSLIB is necessary only on beetle (SunOS), but it does no harm on the other systems.

This creates an executable file called a.out. Use the -o option to name your executable.

8.6 Example Fortran Program

```
      PROGRAM EXAMPLE

C-----------------------------------------------------------------------
C  This is a simple program to demonstrate the basic IOS Header library
C  routines.  The program reads a data file with a channel named
C  'Pressure', and copies the data records with Pressure < 100. to an
C  output file.
C
C  The program is written for files where the IOS Header is in the same
C  file as the data.  Consequently the data is stored in an intermediate
C  scratch file so that the header can be updated and written after
C  the data is read.
C-----------------------------------------------------------------------


C     INCLUDE  'ios_header$inc:header_common.inc'  !vax
      INCLUDE  'header_common.inc'                 !dvf

      REAL     data(max_file_chan)

      INTEGER         inunit, outunit, scratch        ! unit numbers
      INTEGER         nrecin, nrec, nscr             ! record counters
      INTEGER         nchan,ppos
      INTEGER         irec,i,stat
      CHARACTER*22    time_stamp

      INTEGER         HRC_CHANNEL_POS1    ! header library function

C-----------------------------------------------------------------------

      scratch = 3  ! scratch file unit number

      !-----------------------------------------------------------------
      !  Set header message unit numbers.
      !  Could use CALL HR_OPEN_MESSAGE_FILE('header.msg',stat) instead.
      !-----------------------------------------------------------------

      CALL HR_SET_ERROR_UNITS(1,6)                ! in: 1, out: 6

      !-----------------------------------------------------------------
      !  Make sure you have compiled with the same version of the header
      !  structure as the library.
      !-----------------------------------------------------------------

      CALL HR_CHECK_SIZE(h, h.last_byte)

      !-----------------------------------------------------------------
      !  Open the input header/data file.  (You can use your own
      !  Fortran OPEN statement if you want.)
      !-----------------------------------------------------------------

      !                      ---in----    -out-   -out-
      CALL HR_OPEN_FOR_READ('myfile.dat', inunit, stat)
      IF (stat .NE. 0) STOP '*** Open Error ***'

      !-----------------------------------------------------------------
      !  Read the IOS Header.
      !-----------------------------------------------------------------

      CALL HRC_READ_HEADER(inunit,stat)
      IF (stat .EQ. 1) WRITE(*,*)'*** Header Read Warnings ***'
      IF (stat .EQ. 2) STOP'*** Header Read Errors ***'

      !-----------------------------------------------------------------
      !  Extract the number channels, number of records, and location
      !  of the pressure channel.
      !-----------------------------------------------------------------

      nchan = h.fil.number_of_channels
      nrecin = h.fil.number_of_records
      ppos = HRC_CHANNEL_POS1('Pressure','*')

      !-----------------------------------------------------------------
      !  Read data until pressure is 100., and write to the scratch file.
      !
      !  The HRC_READ_DATA_R4 routine handles all sorts of file
      !  and data format possibilities.
      !  If you know the file format is in the header FORMAT item,
      !  you could replace it with:
      !     READ(inunit,FMT=h.fil.format,IOSTAT=stat) (data(i),i=1,nchan)
      !     nrec = nrec + 1
      !-----------------------------------------------------------------

      nrec = 0        ! number of records read
```

```
nscr = 0        ! number of records written to scratch file

CALL HRC_READ_DATA_R4(inunit,nrec,data,stat)
DO WHILE(stat .EQ. 0 .AND. data(ppos) .LT. 100.)
    WRITE(scratch) (data(i),i=1,nchan)
    nscr = nscr + 1
    CALL HRC_READ_DATA_R4(inunit,nrec,data,stat)
ENDDO

CLOSE(inunit)

!------------------------------------------------------------------
!  Modify the header in common for output.
!------------------------------------------------------------------

CALL HRC_SET_TIME_STAMP(time_stamp)
CALL HRC_PROGRAM_HISTORY('EXAMPLE','1.00' ,time_stamp,nrecin,nscr)
CALL HRC_ADD_REMARK('HIS','Reduced to pressures under 100.')
h.fil.number_of_records = nscr            ! new number of records

!------------------------------------------------------------------
!  Open the output header/data file.  (You can use your own
!  Fortran OPEN statement if you want.)
!------------------------------------------------------------------

!                      ---in----     -in-  -out-   -out-
CALL HR_OPEN_FOR_WRITE('newfile.dat', 'F', outunit, stat)
IF (stat .NE. 0) STOP '*** Open Error ***'

!------------------------------------------------------------------
!  Write the output header.
!------------------------------------------------------------------

CALL HRC_WRITE_HEADER(outunit,stat)
IF (stat .EQ. 1) WRITE(*,*)'*** Header Write Warnings ***'
IF (stat .EQ. 2) STOP'*** Header Write Errors ***'

!------------------------------------------------------------------
!  Copy the data from the scratch file to the output file.
!
!  As with reading, the HRC_WRITE_DATA_R4 routine could be
!  replaced with a Fortran WRITE statement, e.g.:
!  WRITE(outunit,FMT=h.fil.format,IOSTAT=stat) (data(i),i=1,nchan)
!  nrec = nrec + 1
!------------------------------------------------------------------

REWIND(scratch)
nrec = 0
DO irec = 1, nscr
    READ(scratch) (data(i),i=1,nchan)
    CALL HRC_WRITE_DATA_R4(outunit,nrec,data,stat)
ENDDO

CLOSE(scratch)
CLOSE(outunit)

WRITE(*,*)'Program Complete'
WRITE(*,*)nrec,' data records written.'

END
```

8.7 Routing Header Messages

Various header routines produce warning and error messages. (Especially the header READ routine.) By default, these messages are written to unit 0. This unit is not explicitly opened, so in VMS a file called FOR000.DAT is created, and in Win32, the messages go to the screen.

Header messages come from a file called MESSAGES.TXT, located in the IOS_HEADER$LIB (VMS) or HEADPATH (Win32) directory. By default, unit 1 is used for this file. The library routine HR_SET_ERROR_UNITS can be called to specify unit numbers to be used for message input and output:

```
      CALL HR_SET_ERROR_UNITS(uin,uout)

 eg:  CALL HR_SET_ERROR_UNITS(2,6)
         - will cause unit 2 to be used for input,
                        unit 6 to be used for output.
```

If the output unit is set (to anything other than zero), some exceptional warning and error messages are also written to a second output unit, which by default is the "standard output" - unit zero. In Windows programs, this can cause crashes. To control this "standard output", use routine HR_SET_UNITS instead of HR_SET_ERROR_UNITS:

```
    CALL HR_SET_UNITS(uin,uout,ustd)

        - uin and uout are the same as the HR_SET_ERROR_UNITS parameters.
        - ustd:  6 or 0 - for console output
                        - can cause crashes for Windows programs.
                -1      - for none
                -2      - for Windows message box
                        - there may be multiple message boxes for each message!
            - may be the unit of any open output file.
            - output to ustd is suppressed if ustd = uout
```

If you want the output messages to go to a file, you can open the file in your program, then call HR_SET_ERROR_UNITS or HR_SET_UNITS, or you can use the routine in the following example:

```
    CALL HR_OPEN_MESSAGE_FILE('my_header_messages.txt',STAT)
```

Routine HR_OPEN_MESSAGE_FILE will open the specified file using some free unit number greater than 10, and return the open status in STAT. You never have to know what unit number the header library is using for output messages. If you want to close the message file before your program ends, you can call HR_CLOSE_MESSAGE_FILE().

8.8 Subroutine Naming Convention

All IOS Header library routines start with the characters "HR_" or "HRC_". Most start with "HR_". Routines named "HRC_" are routines which work on the header structure stored in common block IOS_HEADER. (The C in "HRC_" stands for Common).

For each "HRC_" routine there is a routine of the same name except with the "HR_" prefix. The "HRC_" routine just includes the header common, then calls the "HR_" routine with the header structure in common. The "HR_" routine has the same calling sequence except that the header structure name is added as the first parameter.

For example:

```
    HRC_READ_HEADER(  U,HSTAT) - reads a header into common
     HR_READ_HEADER(H,U,HSTAT) - reads a header into header
```

```
                              structure H
```

In VMS, the module file names are the same as the subroutine names. In most cases there is just one subroutine per file.

Note the following, however:

- an HRC_* and its related HR_* routine are in the same file, named HRC_*.FOR
- in a few cases, the subroutine name is actually an ENTRY in another routine. E.g. all the HRC_GET_ITEM_* routines are entries in file HRC_GET_ITEM_I.FOR

In DOS, 8 character names had to be used. In each module, there is a line just before the copyright notice of the form:

```
    CDOS: <dosname>
```

`<dosname>.FOR` is the name that file was given when converted to DOS.

8.9 Summary of The High Level Library Routines

The main routines a programmer will use are summarized below. For a complete list, and more details, see the FORTRAN LIBRARY ROUTINES section.

```
HR_SET_ERROR_UNITS
    The library routines write warning and error messages.
    Many messages come from a disk file.
    Unit numbers for this I/O may be specified with this routine.
    If they are not specified, messages are read from unit 1, and
    are written to unit 0.

HR_OPEN_MESSAGE_FILE
    Opens a specified file for output header messages.

HR_CHECK_SIZE        - when provided with H.LAST_BYTE of the header
                       declared in your program, this routine checks
                       that your header structure is the same size as
                       the one used by the header library routines.
                     - I.e. checks that your program doesn't need
                       re-compiling !

HRC_READ_HEADER      - reads in a header file into common IOS_HEADER.
                     - the file must already be open.
                     - if data immediately follows the header in the
                       same file, the file pointer is left pointing to
                       the first data record.

HRC_WRITE_HEADER     - writes the header out to a file.
                     - the file must already be open
                     - does NOT close the file, so data can be written
                       after the header by the user's program.

HRC_SET_TIME_STAMP   - gets the system date and sets the time stamp
                       in the header.
                     - the time stamp is also returned.

HRC_GET_ITEM_<type> - where <type> is one of C I I2 I4 R4 R8 LL FT TI

    Given a section name (or just the 1st 3 characters) and an item
    label, the item information is extracted from the header,
    converted (if necessary) to type <type>.

    These routines work for both standard and custom items.
    These routines allow a user to get header data without having to
    include the header common, without having to know the exact data
    type, and without having to know whether the item is a standard
    or a custom item.

HRC_PUT_ITEM_<type>

    This set of routines does the opposite of the HRC_GET_ITEM_<type>
    routines - putting information INTO the header.

HRC_ADD_REMARK - adds a remark to a specified section (including the
                 COMMENTS section.)
               - do not try and add remarks directly into the header
                 structure from your Fortran program!
               - see also the other remark handling routines.

HRC_INIT_HEADER - sets all header item values to null-absent, and
                  table row counters to zero.
                - HRC_READ_HEADER calls this routine, so it is not
                  necessary to call INIT_HEADER before reading.
```

8.10 Structure Definitions

All the header structure definitions are located in the directory pointed to by logical name IOS_HEADER$INC (VMS) and in \\paciosfp2\osapshare\libraries\dvf (Win32). Structure definition include files are named SR_*.INC.

SR_HEADER.INC (listed below) contains the highest level definition of the header structure. It includes other sub-structure definitions: one for each header section. This continues down to the lowest level structures. For example SR_FULLTIME includes SR_DATE and SR_TIME, and anywhere a full-time definition is required, SR_FULLTIME is included.

It is recommended that these same include files be used to define structures in application programs dealing with the IOS Header.

```
C========================================================================
C
C  IOS HEADER DEFINITION
C
C  Abbreviations:
C         SR_     - prefix for structure names
C         CHAN    - CHANNELS  (in parameter names only)
C         COEFF   - COEFFICIENT
C         N       - number of elements or rows in an associated table
C                     e.g. REMARK.N is the total number of comments & remarks
C
C     The section structure definition names are <SR_section>,
C     where <section> is the unabbreviated section name.
C
C     However the section VARIABLE names ARE abbreviated, to their first
C     three characters.
C
C  Fortran Notes:
C     - this include file defines a header STRUCTURE only - no
C       structure-variables are declared.  Use a statement like:
C              RECORD /SR_HEADER/  HEADER
C       to define a structure-variable called HEADER
C  VB Notes:
C     - a SR_HEADER type is defined, but VB 4 & 5 blows up if you try
C       to declare a variable of this type.  It is too big for VB.
C     - the individual header sections can be declared, however.
```

```
C
C===============================================================================
C  Copyright 1992 Government of Canada, Department of Fisheries and Oceans,
C  Institute of Ocean Sciences, Sidney B.C.
C===============================================================================
C  1992 Aug 18  Written by Matthew Nicoll, Cypher Consulting
C  1993 Jul 14  MN: added GENERATION_DATE
C===============================================================================

              ! size parameters & basic structures
      INCLUDE 'header_basics.inc'

      STRUCTURE /SR_HEADER_ID/            ! of input header
          CHARACTER*8     TYPE            ! IOS, MLH etc.
          CHARACTER*4     VERSION         ! of header definition
          CHARACTER*10    VERSION_DATE    ! of header definition
          CHARACTER*10    GENERATION_DATE ! date generated from ITEM_DEF.TXT
      END STRUCTURE

      STRUCTURE /SR_ARRAY/
          CHARACTER*(LEN_LABEL)   NAME    ! table name
          CHARACTER*3             SECID   ! section ID read from
          INTEGER                 NROWS   ! number of rows
          INTEGER                 NCOLS   ! number of columns
          INTEGER                 PTR     ! start point in H.A
      END STRUCTURE

C     --------------------------------------------- Section Structures ---

      INCLUDE 'sr_comments.inc'
      INCLUDE 'sr_administration.inc'
      INCLUDE 'sr_location.inc'
      INCLUDE 'sr_deployment.inc'
      INCLUDE 'sr_recovery.inc'
      INCLUDE 'sr_instrument.inc'
      INCLUDE 'sr_history.inc'
      INCLUDE 'sr_raw.inc'
      INCLUDE 'sr_calibration.inc'
      INCLUDE 'sr_file.inc'

      INCLUDE 'sr_remarks.inc'
      INCLUDE 'sr_custom.inc'

C     --------------------------------------------- Header Structure ---

      STRUCTURE /SR_HEADER/
          CHARACTER*22                TIME_STAMP
          RECORD /SR_HEADER_ID/       HEADER_ID

          RECORD /SR_FILE/            FIL
          RECORD /SR_ADMINISTRATION/  ADM
          RECORD /SR_LOCATION/        LOC
          RECORD /SR_DEPLOYMENT/      DEP
          RECORD /SR_RECOVERY/        REC
          RECORD /SR_INSTRUMENT/      INS
          RECORD /SR_RAW/             RAW
          RECORD /SR_CALIBRATION/     CAL

          RECORD /SR_HISTORY/         HIS
          RECORD /SR_COMMENTS/        COM

          RECORD /SR_CUSTOM/          CUSTOM              ! custom items
          RECORD /SR_REMARKS/         REMARK              ! comments & remarks

          INTEGER                     SECTION_IN_ORDER(NUM_SECTIONS)
          INTEGER                     SECTION_OUT_ORDER(NUM_SECTIONS)

          INTEGER                     NUM_ARRAYS
          RECORD /SR_ARRAY/           ARRAY(MAX_NUM_ARRAYS)
          REAL*4                      A(MAX_ARRAY_DATA)
          LOGICAL*1                LAST_BYTE          ! for checking length
      END STRUCTURE

C===================== End of Header Definition ==============================
```

8.11 Array sizes and String lengths

Array sizes, such as the maximum number of remarks etc. are defined as parameters in file IOS_HEADER$INC:ARRAY_SIZES.INC.

The length of various strings in the header, such as channel names, are defined as parameters in file IOS_HEADER$INC:STRING_LENGTHS.

If you have a header structure already defined in a program, these parameters will already be available, because they are included in the header structure definition include files.

8.12 Access to Header Data in a FORTRAN program.

8.12.1 Introduction

All the data of a header is stored in a large structure, containing many sub-structures (e.g. one for each header section). See the definition of RECORDs and STRUCTUREs in the DEC or MS Fortran manuals, and the Structure Definitions section above.

Thus an entire header-full of data can be passed to a subroutine with a single parameter - the header structure name. This is how most of the header library routines work. It is also easy to define an array of header structures, if you are working with a number of files all at once.

There are two methods of handling the header in a user's program. One method is to define a header structure (or more than one) in the program, and pass the structure name to library routines named "HR_". The other method is to use the "HRC_" library routines, which work with a header structure in a common block called IOS_HEADER.

8.12.2 Opening a Header File for Output

The HRC_WRITE_HEADER and HR_WRITE_HEADER routines must be given the unit number of a header file which is already open. In VAX FORTRAN, include the CARRIAGECONTROL='LIST' option in the OPEN statement, (or use the IOSLIB OPENF routine) so that the first character of each record does not get stolen as a carriage control character.

Using the HR_OPEN_FOR_WRITE routine is another alternative.

8.12.3 Defining your own header structure

To define a header structure in your program:

```
    INCLUDE 'IOS_HEADER$INC:SR_HEADER.INC'
    RECORD /SR_HEADER/ MY_HEADER
```

(See the section titled Accessing_Header_Include_Files, for details on using IOS Header include files.)

After opening your header file, you could then read the header file on unit 8 as follows:

```
    CALL HR_READ_HEADER(MY_HEADER,8,STATUS)

STATUS is returned 0 - all OK
```

```
                   1 - warnings
                   2 - errors
```

The number of records in the data file could then be copied into local variable NRECS as follows:

```
    NRECS = MY_HEADER.FIL.NUMBER_OF_RECORDS
```

8.12.4 Using the header structure in common

To accomplish the same thing as in the example above, using the header common block:

```
    CALL HRC_READ_HEADER(8,STATUS)
    CALL HRC_GET_ITEM_I('FIL','NUMBER OF RECORDS', NRECS, TYPE,STATUS)
```

You could change items using the HRC_PUT_ITEM_<type> routines, but to access the header structure directly, or to access data in any of the header tables, you will have to include the header common in your program. You can do this as follows:

```
    INCLUDE 'IOS_HEADER$INC:HEADER_COMMON.INC'
```

This will declare common block IOS_HEADER, containing one header structure named simply H. This means that the number of records in the file would be in H.FIL.NUMBER_OF_RECORDS.

If you want to use the common block, but would prefer a structure name other than "H", you can code the following instead:

```
    INCLUDE 'IOS_HEADER$INC:SR_HEADER.INC'  ! define the structure
    RECORD /SR_HEADER/ MY_HEADER            ! declare the structure
    COMMON /IOS_HEADER/ MY_HEADER           ! put in common
```

8.12.5 Composite data type structures

LL, FT and TI are composite header data types, defined using structures. If you want to declare any of these types in your program, you can use the same structure definitions as in the header. They are located in the directory pointed to by logical name IOS_HEADER$INC:, and are called

```
  LL - SR_LATLON.INC
  FT - SR_FULLTIME.INC
  TI - SR_TIME_INCREMENT.INC
```

Note that SR_FULLTIME.INC includes date and time sub-structure files SR_DATE.INC and SR_TIME.INC.

8.12.6 Time increments

The time increment structure (defined in IOS_HEADER$INC:SR_TIME_INCREMENT) includes a REAL*8 element called COMPOSITE. When a time increment is read from a header, the five REAL*4 numbers of the time increment are combined into COMPOSITE, with units of DAYS. The COMPOSITE is never written to an output header.

```
Note the following:
    - routines HR_R8_TO_TI and HR_PUT_ITEM_TI also calculate the
      COMPOSITE.

    - if any of the time increment values are changed in a user's
      program the composite value WILL NO LONGER BE CORRECT!
```

```
If your program uses time increments in days, this feature saves
having to write code to combine the 5 time increment values.

See routines HR_R8_TO_TI and HR_TI_TO_R8, for working with time
increments.
```

8.12.7 Remark Handling

All the comments and remarks of the header are kept in a single array. Each header section has pointers into the array called REM1 and REM2 indicating that section's first and last remark.

If a section has no remarks, REM1 is set 0, and REM2 is set -1.

Along with each remark is kept its trimmed length, and the amount it should be indented relative to the other remarks when it is written out.

Remarks may be copied out of the header structure directly. To add or delete remarks, however, use the remark-handling routines in the IOS_HEADER library. (E.g. use the routine HRC_ADD_REMARK to add comments.)

If the header structure is named "H", you could print out remarks as follows:

```
    DO I=H.ADM.REM1,H.ADM.REM2              ! all ADMINISTRATION remarks
        WRITE(*,*)H.REMARK.REM(I).TEXT
    ENDDO

    DO I=H.COM.REM1,H.COM.REM2              ! all COMMENTS section
        WRITE(*,*)H.REMARK.REM(I).TEXT
    ENDDO

    DO I=1,H.REMARK.N                       ! all remarks, all sections
        WRITE(*,*)H.REMARK.REM(I).TEXT
    ENDDO
```

The remark-handling routines allow for remarks to be dealt with by section. For example the following code would also write out all the ADMINISTRATION remarks:

```
    CHARACTER*80 REM
    INTEGER      I,HRC_N_REMARKS
    DO I = 1,HRC_N_REMARKS('ADM')
        CALL HRC_GET_REMARK('ADM',I,REM)
        WRITE(*,*)REM
    ENDDO
```

8.13 Custom Items

If the read routine does not recognize an item label (ie if it is not a Standard Item) then a Custom Item is created. Custom items are stored in a special table in the header structure. The information is stored in character form.

If the following record was encountered in the ADMINISTRATION section of a header:

ACCOUNT NUMBER : 12345

and this was the first custom item found, then if the header structure is called H:

```
    H.CUSTOM.ITEM(1).SECTION would contain 'ADM'
    H.CUSTOM.ITEM(1).LABEL   would contain 'ACCOUNT NUMBER'
    H.CUSTOM.ITEM(1).INFO    would contain '12345'
```

Custom item information can be obtained directly from the header structure. See routine HR_LOOKUP_CUSTOM_ITEM.

The HRC_PUT_ITEM... and HRC_GET_ITEM... routines may be a more convenient way of accessing custom items.

For example, to retrieve the ACCOUNT NUMBER item into character variable ACNUM, and integer variable INT_ACNUM:

```
    CHARACTER   ACNUM*10, ITEMTYPE*2
    INTEGER     STAT,INT_ACNUM
    CALL HRC_GET_ITEM_C('ADM','ACCOUNT NUMBER', ACNUM    ,ITEMTYPE,STAT)
    CALL HRC_GET_ITEM_I('ADM','ACCOUNT NUMBER', INT_ACNUM,ITEMTYPE,STAT)
```

An advantage of using these routines is that if ACCOUNT NUMBER were to become a Standard Item in the definition of the header, the same code would work.

8.14 Time Stamp

The time stamp in the header (as of 25-Sep-2002) is set by the HRC_INIT_HEADER routine. However the time stamp on a header is **not** automatically re-set when a header is read or written. The routine HRC_SET_TIME_STAMP must be used to explicitly set a time stamp.

8.15 Section Inclusion flags

Each section of the header has LOGICAL*1 flags called INCLUDE_IN and INCLUDE_OUT. If a section was found during input, both these flags are set to TRUE. The header write routine only writes out sections whose INCLUDE_OUT flag is set.

So, if your program adds information to a section which was not present in the input header file, you will have to set the INCLUDE_OUT flag to get that section in the output header. For example, if the header structure name is "H", then to make sure the DEPLOYMENT section gets written out:

```
    H.DEP.INCLUDE_OUT=.TRUE.
```

If a 'PUT' subroutine is used to put item information into a header, the inclusion flag for the section is automatically set.

8.16 Channel Detail Inclusion Flag

The flag H.FIL.INCLUDE_CHANNEL_DETAIL must be set for the table to be included in the output header.

The flag is turned on automatically if the input header included the table. It is also turned on by any Header Library routines which put information into the detail table. The user's program must control the flag otherwise. If you put information into the table, set the flag to make sure the table appears in the output header:

```
    H.FIL.INCLUDE_CHANNEL_DETAIL = .TRUE.
```

8.17 Section Ordering

The 'Standard' order of the sections in the output header is the order in which sections are defined in the IOS_HEADER$GEN:ITEM_DEF.TXT file.

If a header is read in, then written out, the section ordering will not be changed. If the header is NOT read in, the sections will be written in Standard order.

The subroutine HR[C]_SET_SECTION_ORDER can be used to control the output section order.

8.18 Header Null Values

The IOS Header null values are defined in previous sections of this document.

The null values are defined as parameters in the following include file: IOS_HEADER$INC:NULVALS.INC. The values are defined separately for Numeric and Character types:

```
          Numeric        Character

absent : HN_ABSENT       HC_ABSENT
blank  : HN_BLANK        HC_BLANK
unknown: HN_UNKNOWN      HC_UNKNOWN
n/a    : HN_NA           HC_NA
```

Thus you don't need to know the actual values. To set the ICE THICKNESS to n/a, just code H.LOC.ICE_THICKNESS=HN_NA.

The following utility routines can save you from using the include file. The 'ISNULL' functions save you from having to do four comparisons to determine if a variable is null or not.

```
HR_SET_NULL         - sets a non-character variable to a
                      null value
HR_ISNULL_<type>    - determines if a variable of type <type>
                      is null.  <type> may be one of I I2 I4 R4 R8
                      FT TI LL C
HR_ISNULL           - similar: the type is included as a parameter
                      instead of part of the function name.
                    - does not handle type character.
HRC_ISNULL_ITEM     - determines if a named item is null.
```

8.19 Arrays

Header arrays must be accessed using the provided array handling routines. These routines are documented in the FORTRAN_LIBRARY_ROUTINES section, under ARRAY_HANDLING.

There are routines for extracting, replacing, deleting, and defining arrays. You can also extract individual rows, columns or elements.

To access an array, you must know the name of the array. You must also declare your own array large enough to receive the array from the header.

Use routine HRC_GET_ARRAY_SIZE to find out an array's size, then use HRC_GET_ARRAY to copy the array out of the header.

The following information is available directly from the header:

```
MAX_NUM_ARRAYS   - (fortran parameter) maximum number of arrays
                   allowed.
MAX_ARRAY_DATA   - maximum number of data elements, for all arrays,
                   which can be stored in a header structure.

H.NUM_ARRAYS     - number of arrays currently defined in the header
```

```
H.ARRAY(I).NAME  - name of the I'th array.
H.ARRAY(I).NROWS - size of the I'th array.
H.ARRAY(I).NCOLS
H.ARRAY(I).SECID - Section ID of the section where the array
                   was located in an input header, and where it
                   will be written in an output header.
```

To move an array from one section to another:

```
CALL HRC_GET_ARRAY_NUMBER(NAME, ANUM)
H.ARRAY(ANUM).SECID = 'FIL'
```

8.19.1 Example Array Program

```
   -------------------------------------------------------------------
C  The following program gets an array from the header, then prints it:

      INTEGER         MAXROW,     MAXCOL
      PARAMETER      (MAXROW=20),(MAXCOL=3)       ! size of YOUR array

      REAL*4          A(MAXROW,MAXCOL)
      CHARACTER*20    NAME /'COMPASS CORRECTIONS'/
      INTEGER         NROWS,NCOLS          ! array size in header

      LOGICAL*1       HRC_ARRAY_EXISTS     ! function

        ... code to open and read in the header ...

      IF (.NOT. HRC_ARRAY_EXISTS(NAME)) STOP '*** Array not there ***'

      CALL HRC_GET_ARRAY_SIZE(NAME, NROWS,NCOLS)
      CALL HRC_GET_ARRAY(NAME,MAXROW,MAXCOL, A)

      DO I=1,NROWS
        WRITE(*,*)(A(J),J=1,NCOLS)
      ENDDO

   -------------------------------------------------------------------
```

8.19.2 Array References

For each channel in each of the calibration section tables, there is
an element called ARRAY_NAME:

```
H.CAL.RAW_CHANNEL(I).ARRAY_NAME
H.CAL.CALC_CHANNEL(I).ARRAY_NAME
H.CAL.CORR_CHANNEL(I).ARRAY_NAME
```

This array name will be blank if there is no array reference.

If there is an array reference, a calibration program can use the array name to access the array in the header, using the array handling libraray routines.

Thus when a calibration requires more 'coefficients' than will fit in a coefficient list (e.g. a compass corrections lookup table), the 'coefficients' can be put into an array instead.

8.20 TIME SERIES Handling

There are a number of different ways the time of a record in a time series file can be stored or inferred:

- calculated as START_TIME + (recnum-1) * TIME_INCREMENT
- Day_of_Year and Offset methods, as described under USAGE_NOTES / TIME_CHANNEL_DEFINITION
- formatted date and/or time channels, as defined in the SPECIAL_CHANNEL_TYPES section.

There are Header Library routines to allow a program to be written to handle any of these types of time series files, in a generalised manner. The routines allow you to open a time-series file, read the header, read a data record, then get the time of that record, without knowing how the time is actually recorded in the file. The routines also allow a number of time-series files, possibly of different types, to be read and coordinated.

```
HRC_TSZ_PREP     - identifies time channels, either automatically or by
                   using specified channel numbers.
                 - defines the mapping from Z8 times to the R8 times of
                   a particular header.
                 - stores the results in common defined in HR_TSZ.INC
HR_TSZ_DUMP      - writes out the contents of common HR_TSZ
```

The purposes of the following routines can be determined from their names, if the routine name parts are interpreted as follows:

TS	depends upon contents of common HR_TSZ, uses a Real*4 data vector
TS8	depends upon contents of common HR_TSZ, uses a Real*8 data vector
R8	a Real*8 number of days from a header time zero, in the header zone
Z8	a Real*8 number of days from the Fixed Time Zero, UTC
FT	a full-time structure.
GET	obtains the time of the last record read with HR_READ_DATA_R?
PUT	stores the time ready for writing with the next call to HR_WRITE_DATA_R?

```
HR_TS_FT_TO_R8
HR_TS_R8_TO_FT

HR_TS_GET_FT
HR_TS_GET_R8
HR_TS_GET_Z8
HR_TS_PUT_R8
HR_TS_PUT_Z8

HR_TS8_GET_FT
HR_TS8_GET_R8
HR_TS8_GET_Z8
HR_TS8_PUT_R8
HR_TS8_PUT_Z8

HR_FT_TO_Z8.F
HR_Z8_STRING
HR_Z8_TO_FT
```

Other related routines:

```
HRC_FINDTIME         - examines a header to determine how time is recorded
HR_SET_ZERO_CONDAY        - sets the IOSLIB zero conday to the header
                            time zero if it exists, else the Fixed Time Zero
HR_SET_FIXED_ZERO_CONDAY - sets the IOSLIB zero conday to the FTZ
```

To determine what the Fixed Time Zero is, call HR_SET_FIXED_ZERO_CONDAY, then call the IOSLIB routine GET_ZERO_CONDAY. It is currently (2003-06-26) set as 1900-01-01.

The sequence when reading one file is:

```
HRC_READ_HEADER
HRC_TSZ_PREP
Loop
    HRC_READ_DATA_R4
    HR_TS_GET_R8/FT
End Loop
```

HR_TSZ_PREP sets up the mapping from one file/header to the file- independent time base. Thus when reading from multiple files, HR_TSZ_PREP must be re-called with the header of the file about to be read, before using the other HR_TS routines.

If more than one file is being read:

```
For each file f (and corresponding header Hf):
    HR_READ_HEADER(Hf,...)
End For

Loop
    For each file f
        HR_TSZ_PREP(Hf,...)
        HR_READ_DATA_R4(Hf,...)
        HR_TS_GET_R8/FT
    End For
End Loop
```

For dealing with files with multiple special date/time channels, it is recommended that you a) use REAL*8 data vectors, b) use the HR_TS8 routines, and c) carefully read the documentation of each HR_TS routine you use, to make sure you know what channels it is working with.

See the documentation on each routine for more details.

8.21 Data Column Headings

Programs linked after 28-April-1999 will produce IOS_Headers with column headings included as header comment records just before the *END OF HEADER record. The headings are produced by routine HRC_WRITE_COLUMN_HEADINGS, and are generated each time the header is written, using the current format information in the header.

The headings can be turned off, or customised, by calling routine HC_SET_COLUMN_HEADINGS any time before the call to HRC_WRITE_HEADER.

8.22 Mandatory Items and Sections

Header sections and items may be specified as "mandatory" in a file called the "Mandatory Item File". The routine HR_CHECK_HEADER must be called to verify that all mandatory items and sections are present in a header in memory.

This entire feature is optional: the HR_CHECK_HEADER routine need not be called. If it is called, and the mandatory item file does not exist, only one warning message will be produced.

Sections and items are specified independently. If an item is specified as mandatory, but its section is not, the item is mandatory only if the section is present (i.e. if the section's input or output flag is turned on.)

See the description of the UTILITY library routine HRC_CHECK_HEADER for full details, including Mandatory Item File coding and naming instructions.

8.23 Header Routine Libraries

There is a main library of IOS Header FORTRAN routines available for linking in VMS (VAX and AXP) and 32-bit Windows. The routines in this library are documented in the FORTRAN LIBRARY ROUTINES section of this document.

There are likely to be additional libraries of header routines (or libraries containing header-handling routines) maintained by groups within IOS. The latter libraries will contain routines written to handle or simplify header-handling tasks specific to the data-processing systems of each group.

If you have written a header routine which you think should be in the main library (ie you think it would be generally applicable):

- name it in a manner consistent with the existing routines,
- format the module header in the same manner as the existing routines (see the IOS Header System Manual for a definition of the format),
- submit it via Email to LinguantiJ@pac.dfo-mpo.gc.ca or menicoll@CypherConsulting.com

8.24 Feed-Back

Send all bug reports, suggestions, requests and complaints regarding the header and header routines via Email to menicoll@CypherConsulting.com and LinguantiJ@dfo-mpo.gc.ca

9 The Win32 IOS Header Dynamic Link Library

9.1 Introduction

The IOS Header DLL was created using Visual Fortran, for use in 32-bit Windows (95, 98, NT and XP) It was created primarily for use from Visual Basic, but it can be used from any language that can call DLL routines. It has been tested with Visual Basic and Fortran. IOS Header structure (type) definitions have been generated for Visual Basic. The DLL calling interface definitions have been generated for both Visual Basic and Fortran.

The entire IOS Header library is incorporated in the DLL, but not all routines have been made available to calling programs. Routines which involve passing an entire header structure are NOT available externally. This is because Visual Basic cannot handle a structure as large as the IOS Header structure. The DLL could be re-developed with the full interface if it becomes desirable to use the DLL from Fortran or C.

9.2 Using the DLL from Visual Basic

9.2.1 Required files

From `\\paciosfp2\osapshare\libraries\dlls` get the following files:

`hdriface.bas`	contains declare statements for all the routines callable in the IOS Header DLL. Add this file to your Visual Basic project.
`hdrtypes.bas`	contains type definitions of all the IOS Header structures, and constant definitions for all the array sizes, string lengths and null values. Add this file to your project if you need any of these definitions.
`headdll.dll`	must be somewhere on your Windows path.

9.2.2 Sample VB / Header Project

Copy all the files from the o:\libraries\ios_header\vbdemo directory into a directory on your PC. Make sure the headdll.dll file is the same directory, or somewhere in your Windows Path. Open project file DLLHEAD.VBP in Visual Basic.

Module HEADTEST.BAS contains subroutines for opening, modifying, writing and closing IOS Header files using the DLL - each step controlled from a button on the form.

9.2.3 Accessing Header Data

When using the DLL, header data is accessed in the same manner as when using the static library from Fortran, except that in Visual Basic, an entire header cannot be obtained in one structure, so individual header section structures are obtained and replaced using routines HR_GETSEC_xxx and HR_PUTSEC_xxx.

When the static library is used from Fortran, the header can be read into a common block (using the HRC_ routines) or into a provided header structure (using the HR_ routines). This makes it easy to manipulate two or more headers at once. When used as a DLL, only the header-in-common option is available. To manipulate multiple headers in a VB program, it would be necessary to declare multiple copies of each header section, and manage the reading, getting, putting and writing, through the DLL's common block.

9.2.4 IOS Header File Handling

The IOS Header library, prior to the development of the DLL, left all file opening and closing up to the main Fortran program. The Fortran program passed unit numbers to the library routines. When used as a DLL, the DLL must perform its own file handling. Therefor the following routines were added to the library. These routines must be used when using the DLL, and are optional otherwise.

HR_OPEN_MESSAGE_FILE	- opens a named file for header messages
HR_CLOSE_MESSAGE_FILE	- closes the message file
HR_OPEN_FOR_READ	- opens a data file for reading
HR_OPEN_FOR_WRITE	- opens a data file for writing
HR_CLOSE_FILE	- closes a data file

These routines are described in detail in the **Fortran Library Routines** section of this manual.

9.2.5 Passing Arguments between VB and the Fortran DLL

9.2.5.1 Type Matching

When preparing to call an IOS Header routine, look up the routine definition in the IOS Header Manual, then in VB declare and code matching arguments according to the following table:

Fortran	**Visual Basic**	* See notes
LOGICAL*1	Byte	*
INTEGER*2	Integer	
INTEGER*4	Long	
REAL*4	Single	
REAL*8	Double	
Structure	User-defined Type	*
CHARACTER*n	String * n + Length As Long	*

To confirm, look up the VB declaration of the routine in file HdrIface.bas. You needn't worry about the calling mechanism (by reference or value) since this is looked after in the declaraction in HdrIface.bas.

9.2.5.2 Character strings.

Every string argument must be immediately followed in the call sequence by a Long (4-byte) integer containing the length of the string. If a header routine returns a string of some fixed length, declare a fixed length string of at least that length, and put it in the call sequence followed by the declared length. If a header routine is receiving an un-determined length string, declared in the Fortran as CHARACTER*(*), you can either code a literal string, followed by its length as a literal integer, or you can send a variable length string, followed by its length. In both cases, a string length can be specified using the VB L*en* function. For fixed length strings, Len gives the declared length, and for variable length strings, it gives the current length.
Examples:

```
Dim variable_filename  As String
Dim fixed_filename     As String * 12
Dim status             As Long
Dim uin                As Long

variable_filename = "header_message_file.txt"
Call HR_OPEN_MESSAGE_FILE(variable_file_name, Len(variable_file_name), status)
fixed_filename = "input.dat"
Call HR_OPEN_FILE_FOR_READ(fixed_filename, Len(fixed_filename), uin, status)
Call HR_OPEN_FILE_FOR_WRITE("output.dat", 10, "F", 1, uout, status)
```

9.2.5.3 String arrays

To create a memory data layout in VB which is the same as a character array in Fortran, declare an array of fixed-length strings inside a VB user-defined type (structure). Send the VB structure (by reference) to correspond to the Fortran character array name In an argument list, followed by the length of one of the array elements, (by value).
Fortran:

```
SUBROUTINE FSUBSTRINGARRAY(C)
!DEC$ATTRIBUTES DLLEXPORT :: FSUBSTRINGARRAY
CHARACTER*6 C(4)
...
END
```

Visual Basic:

```
Type StringArray
      s(1 To 4) As String * 6
End Type

'--- For IOS Header routines, the following declaration would
'--- be in HdrIface.bas
Public Declare Sub FSUBSTRINGARRAY Lib "EXAMPLE.DLL" _
      Alias "_FSUBSTRINGARRAY@8" _
      (ByRef sa As StringArray, ByVal clen As Long)

Sub TestStringArray()
      Dim sa As StringArray
      Dim clen As Long
      clen = Len(sa.s(1))
      Call FSUBSTRINGARRAY(sa, clen)
End Sub
```

9.2.5.4 Non-Character Arrays

Declare the array in VB, and code the first element of the array in the argument list. The interface declaration in HdrIface.bas does not tell VB that the argument is an array, although the argument name in the declaration will end in "_array", to tell *you* that an array is required. Coding the first element sends the address of the start of the array, which is all Fortran wants. It is up to you to make sure the array is declared big enough.
Example:

```
Dim Data (1 to 20) As Single
...
Call HRC_READ_DATA_R4(uin, rn, Data(1), stat)
```

9.2.5.5 LOGICAL vs Byte and Boolean

Fortran LOGICAL variables may be declared size 1, 2 or 4. In all cases, only the first byte is used, and is set to 0 for false, and unsigned 1 for true (which is minus one in two's complement form).
A Visual Basic Boolean variable occupies 2 bytes. All 8 bits are set 1 for true, and all 0 for false. In the IOS Header library, LOGICAL*1 is used for logical variables.
The Visual Basic type *Byte* corresponds well to Fortran LOGICAL*1. A Byte can be used just like a Boolean in VB: it can be set to True (255) or False (0) and can be used in place of Boolean in If statements. When passed to Fortran, Fortran will set the byte to 1 for True, which also tests *true* in VB.

9.2.5.6 Structures

Structure arguments are declared in the IOS Header routines like this:
```
      RECORD /SR_FILE/  FIL
```
The corresponding declaration in VB is:
```
      Dim FIL As SR_FILE
```
The type SR_FILE, along with all other Header Structure-types, is defined in HdrTypes.Bas.

9.3 Using the DLL from Fortran

- Include the HDRIFACE.FOR file from the `\\paciosfp2\osapshare\libraries\dlls` directory in any Fortran program which is to call header routines in the DLL.
- Use the header library routines to open and close the message and data files. Your program and the DLL are like two separate programs - e.g. unit 3 in your program has nothing to do with unit 3 in the DLL. If you open a file in your program, and pass the unit number to the HRC_READ_HEADER routine, the read statement in the DLL will respond with an error 29 - file not found. Instead, use routine HR_OPEN_FOR_READ, which will open a data file and provide you with a unit number to pass to HRC_READ_HEADER.
- Link with the HEADDLL.LIB library instead of the HEADLIB.LIB library.
- At run-time, make sure HEADDLL.DLL is somewhere on your path.

Remember that at present the header routines which involve passing an entire header structure, and the low-level header routines, are not available in the DLL. Check the HDRIFACE.FOR file to see if a routine is available. (These limitations can be remedied, if use of the DLL instead of the static library from Fortran becomes desirable.)

10 Fortran Library Routines

10.1 Array Handling

Array handling routines allow the manipulation of IOS Header arrays: named arrays of numbers delimited in the header with **$ARRAY** and **$END**.

10.1.1 HRC_ARRAY_EXISTS

```
   LOGICAL*1 FUNCTION HRC_ARRAY_EXISTS(ANAME)

Returns a TRUE value if the named array exists in the header.

---Input---
   CHARACTER*(*)   ANAME       ! Array name (NOT case sensitive)


See also: HRC_GET_ARRAY_NUMBER
----------------------------------------------------------------------
```

10.1.2 HRC_DEFINE_ARRAY

```
   SUBROUTINE HRC_DEFINE_ARRAY(SECTION,ANAME,NROWS,NCOLS, ASTAT)

Define the section, name, and dimensions of a new array in the header.

---Input---

   CHARACTER*(*)   SECTION     ! Header section for output.
                               ! only first 3 characters needed,
                               ! not case-sensitive.
   CHARACTER*(*)   ANAME       ! Array name (any case)
   INTEGER         NROWS       ! first dimension     >= 1
   INTEGER         NCOLS       ! second dimension    >= 1

---Output---
   INTEGER         ASTAT       ! 0 - array definition successful
                               ! 1 - array defined, warnings produced
                               ! 2 - ERRORS - array NOT defined.


Use HRC_PUT_ARRAY to put the array data into the header, after it
has been defined.

Array names must be unique in the entire header.  I.e. you may not
defined an array if there is an array of the same name any where
else in the header.  The section specified to this routine just
indicates what section the array is to be written in.

See also: HRC_DELETE_ARRAY, HRC_PUT_ARRAY
----------------------------------------------------------------------
```

10.1.3 HRC_DELETE_ARRAY

```
   SUBROUTINE HRC_DELETE_ARRAY(ANAME)
```

```
Deletes the header array named in string ANAME.

---Input---

   CHARACTER*(*)   ANAME       ! Array name (NOT case sensitive)


Note: this routine decrements the array numbers of any arrays which
      have a higher array number than array ANAME

See also: HRC_DEFINE_ARRAY
----------------------------------------------------------------------
```

10.1.4 HRC_GET_ARRAY

```
   SUBROUTINE HRC_GET_ARRAY(ANAME,MAXROW,MAXCOL, B)

Returns the contents of header array ANAME in array B.
MAXROW and MAXCOL must be the declared dimensions of B.

---Input---

   CHARACTER*(*)   ANAME       ! Array name (NOT case sensitive)
   INTEGER         MAXROW      ! declared first dimension of B
   INTEGER         MAXCOL      ! declared second dimension of B

---Output---
   REAL*4          B(MAXROW,MAXCOL)  ! returned array.
                                     ! B(1,1) set to HN_ABSENT if
                                     ! ANAME not found in header.


Use HRC_GET_ARRAY_SIZE to make sure the array exists in the header,
and to get the actual size of the array.

If B is not declared large enough to hold the array, a warning
message will be produced, but as much of the array as possible
will be put into B.

If B is declared larger than the array in the header, the elements
of B outside the header array bounds will not be altered.

If there is no array called ANAME in the header, an error message
will be produced, and B(1,1) will be set to NULL-ABSENT (HN_ABSENT)

See also: HRC_GET_ARRAY_SIZE, HRC_ARRAY_EXISTS, HRC_PUT_ARRAY
----------------------------------------------------------------------
```

10.1.5 HRC_GET_ARRAY_COLUMN

```
   SUBROUTINE HRC_GET_ARRAY_COLUMN(ANAME,COLNUM, X)

Returns column COLNUM of header array ANAME, in vector X.

---Input---

   CHARACTER*(*)   ANAME       ! Array name (NOT case sensitive)
   INTEGER         COLNUM      ! number of column wanted

---Output---
   REAL*4          X(*)        ! returned array column
```

```
                               ! X(1) set to HN_ABSENT if ANAME not
                               ! found in header, or if COLNUM is bad.


 Use HRC_GET_ARRAY_SIZE to make sure the array exists in the header,
 and to get the actual size of the array.

 It is up to the programmer make sure that X is declared large
 enough.  If it is not, other data in memory will be over-written
 with array data.

 If there is no array called ANAME in the header, an error message
 will be produced, and X(1) will be set to NULL-ABSENT (HN_ABSENT)

 ----------------------------------------------------------------------
```

10.1.6 HRC_GET_ARRAY_ELEMENT

```
    SUBROUTINE HRC_GET_ARRAY_ELEMENT(ANAME,ROWNUM,COLNUM, Z)

 Returns element (ROWNUM,COLNUM) of header array ANAME in Z.

 ---Input---

    CHARACTER*(*)   ANAME       ! Array name (NOT case sensitive)
    INTEGER         ROWNUM      ! row number
    INTEGER         COLNUM      ! column number

 ---Output---
    REAL*4          Z           ! set to HN_ABSENT if ANAME not
                                ! found in header, or if ROWNUM or
                                ! COLNUM is invalid


 Use HRC_GET_ARRAY_SIZE to make sure the array exists in the header,
 and to get the actual size of the array.

 If there is no array called ANAME in the header, an error message
 will be produced, and Z will be set to NULL-ABSENT (HN_ABSENT)

 ----------------------------------------------------------------------
```

10.1.7 HRC_GET_ARRAY_NUMBER

```
    SUBROUTINE HRC_GET_ARRAY_NUMBER(ANAME, ANUM)

 Looks up array name ANAME in the header, and returns the array number
 in ANUM.  ANUM will be returned zero if ANAME could not be found.

 ---Input---

    CHARACTER*(*)   ANAME       ! Array name (NOT case sensitive)

 ---Output---
    INTEGER         ANUM        ! array number - index into H.ARRAY

 Note: An array's number may change after routine HRC_DELETE_ARRAY
       is used.

 See also: function HRC_ARRAY_EXISTS
 ----------------------------------------------------------------------
```

10.1.8 HRC_GET_ARRAY_ROW

```
   SUBROUTINE HRC_GET_ARRAY_ROW(ANAME,ROWNUM, X)

Returns row ROWNUM of header array ANAME, in vector X.

---Input---

   CHARACTER*(*)   ANAME       ! Array name (NOT case sensitive)
   INTEGER         ROWNUM      ! number of row wanted

---Output---
   REAL*4          X(*)        ! returned array row
                               ! X(1) set to HN_ABSENT if ANAME not
                               ! found in header, or if ROWNUM is bad.


Use HRC_GET_ARRAY_SIZE to make sure the array exists in the header,
and to get the actual size of the array.

It is up to the programmer make sure that X is declared large
enough.  If it is not, other data in memory will be over-written
with array data.

If there is no array called ANAME in the header, an error message
will be produced, and X(1) will be set to NULL-ABSENT (HN_ABSENT)

----------------------------------------------------------------------
```

10.1.9 HRC_GET_ARRAY_SIZE

```
   SUBROUTINE HRC_GET_ARRAY_SIZE(ANAME, NROWS,NCOLS)

Looks up array name ANAME in the header, and returns the array size.
NROWS and NCOLS will be returned zero if ANAME could not be
found (i.e. does not exist in the header).

---Input---

   CHARACTER*(*)   ANAME       ! Array name (NOT case sensitive)

---Output---
   INTEGER         NROWS       ! number of rows
   INTEGER         NCOLS       ! number of columns

----------------------------------------------------------------------
```

10.1.10 HRC_PUT_ARRAY

```
   SUBROUTINE HRC_PUT_ARRAY(ANAME,MAXROW,MAXCOL, B)

Copies the contents of array B into the header array named in
string ANAME.  THE ARRAY MUST ALREADY BE DEFINED IN THE HEADER.
MAXROW and MAXCOL must be the declared dimensions of B.
The number of rows and columns copied out of B depends upon
the defined size of the array in the header.

---Input---

   CHARACTER*(*)   ANAME       ! Array name (NOT case sensitive)
   INTEGER         MAXROW      ! declared first dimension of B
   INTEGER         MAXCOL      ! declared second dimension of B

   REAL*4          B(MAXROW,MAXCOL)  ! array.


If array ANAME is not defined in the header, an error message
will be produced, and no array data will be copied into the header.

If B is declared smaller, in either dimension, than the array
as defined in the header, an error message will be produced,
and no array data will be copied into the header.

If B is declared larger than the array in the header, the elements
of B outside the header array bounds will be ignored.

See also: HRC_GET_ARRAY_SIZE, HRC_DELETE_ARRAY, HRC_DEFINE_ARRAY,
          HRC_ARRAY_EXISTS
----------------------------------------------------------------------
```

10.1.11 HRC_PUT_ARRAY_COLUMN

```
   SUBROUTINE HRC_PUT_ARRAY_COLUMN(ANAME,COLNUM, X)

Replaces column COL of header array ANAME, with vector X.

---Input---

   CHARACTER*(*)   ANAME       ! Array name (NOT case sensitive)
   INTEGER         COLNUM      ! number of column to replace
   REAL*4          X(*)        ! array column


It is up to the programmer make sure that X is declared large
enough, and contains enough data to fill a column of the array.

Use HRC_GET_ARRAY_SIZE to make sure the array exists in the header,
and to get the actual size of the array.

If there is no array called ANAME in the header, an error message
will be produced.

----------------------------------------------------------------------
```

10.1.12 HRC_PUT_ARRAY_ELEMENT

```
   SUBROUTINE HRC_PUT_ARRAY_ELEMENT(ANAME,ROWNUM,COLNUM, Z)

Replaces element (ROWNUM,COLNUM) of header array ANAME with Z.

---Input---

   CHARACTER*(*)   ANAME       ! Array name (NOT case sensitive)
   INTEGER         ROWNUM      ! row number
   INTEGER         COLNUM      ! column number
   REAL*4          Z           ! value to insert.


Use HRC_GET_ARRAY_SIZE to make sure the array exists in the header,
and to get the actual size of the array.
```

```
----------------------------------------------------------------------
```

10.1.13 HRC_PUT_ARRAY_ROW

```
   SUBROUTINE HRC_PUT_ARRAY_ROW(ANAME,ROWNUM, X)

Replaces row ROWNUM of header array ANAME, with vector X.

---Input---

   CHARACTER*(*)   ANAME       ! Array name (NOT case sensitive)
   INTEGER         ROWNUM      ! number of row to replace
   REAL*4          X(*)        ! data to replace row with.


It is up to the programmer make sure that X is declared large
enough, and contains enough data to fill a row of the array.

Use HRC_GET_ARRAY_SIZE to make sure the array exists in the header,
and to get the actual size of the array.

If there is no array called ANAME in the header, an error message
will be produced.

----------------------------------------------------------------------
```

10.2 Configure

10.2.1 HR_CLOSE_MESSAGE_FILE

```
   SUBROUTINE HR_CLOSE_MESSAGE_FILE()

Writes a time stamp to the current message file, then closes it.

See also: HR_OPEN_MESSAGE_FILE
----------------------------------------------------------------------
```

10.2.2 HR_OPEN_MESSAGE_FILE

```
   SUBROUTINE HR_OPEN_MESSAGE_FILE(MESFN,STAT)

Opens for output file MESFN, using the first available unit
number > 10, and assigns that unit number to UERR (in common
HR_UNITS), so that all subsequent header messages will be written
to file MESFN. A time stamp is written as the first record of the file.

---Input---
   CHARACTER*(*)   MESFN       ! name of file to open
                               ! if blank, HEADERR.TXT will be used.
---Output---
   INTEGER         STAT        ! 0 if OK, else see Fortran OPEN codes

Use this routine instead of HR_SET_ERROR_UNITS if:
 - using the IOS Header library as a DLL,
 - using the static header library, but don't want to open or write
   other information to the header error message file.

Note that (default) unit 1 will be used by the header library for
reading error messages from the MESSAGES.TXT file.
```

```
See also: HR_CLOSE_MESSAGE_FILE, HR_SET_ERROR_UNITS
----------------------------------------------------------------------
```

10.2.3 HR_SET_ERROR_UNITS

```
   SUBROUTINE HR_SET_ERROR_UNITS(UIN,UOUT)

Sets Fortran unit numbers to use for error message input and output.

---Input---
   INTEGER     UIN     ! unit number to read IOS header library error
                       ! messages from.
                       ! (The file/logical name is set and opened in
                       ! routine HR_LOOKUP_MESSAGE

   INTEGER     UOUT    ! unit number for output of warning and error
                       ! messages from header library routines.  This
                       ! file is NOT opened in header library routines.

If this routine is not called, unit 1 is used for input, and 0 for
output.  If both hr_units are set the same, the input unit is incremented
before use.  (See routine HR_LOOKUP_MESSAGE.)

NOTE: Use HR_SET_UNITS instead of this routine if you want to control
      standard output, e.g. for Windows programs.

See also : HR_OPEN_MESSAGE_FILE, HR_SPLAT, HR_SET_UNITS
----------------------------------------------------------------------
```

10.2.4 HR_SET_UNITS

```
   SUBROUTINE HR_SET_UNITS(UIN,UOUT,USTD)

Sets Fortran unit numbers to use for error message input and output.

---Input---
   INTEGER     UIN     ! unit number to read IOS header library error
                       ! messages from.
                       ! (The file/logical name is set and opened in
                       ! routine HR_LOOKUP_MESSAGE
                       ! Default: 1

   INTEGER     UOUT    ! unit number for output of warning and error
                       ! messages from header library routines.  This
                       ! file is NOT opened in header library routines.
                       ! Default: 0

   INTEGER     USTD    ! unit number for "standard output".
                       ! 6 (or 0) - for console (window) output
                       ! -1       - for none
                       ! -2       - for Windows message box
                       ! Default: 0
                       ! For Windows programs, do not use 0 or 6.

If both UIN and UOUT are set the same, the input unit is incremented
before use.  (See routine HR_LOOKUP_MESSAGE.)

If UOUT and USTD are set the same, messages are written only once.
(See routine HR_SPLAT)
```

```
This routine is the same as HR_SET_ERROR_UNITS, but with the USTD
argument added.

See also : HR_OPEN_MESSAGE_FILE, HR_SET_ERROR_UNITS
----------------------------------------------------------------------
```

10.2.5 HRC_SET_SECTION_ORDER

```
   SUBROUTINE HRC_SET_SECTION_ORDER(COMMAND)

Defines the order in which sections are written to the output header.
(Sets the SECTION_OUT_ORDER array in the header structure.)

---Parameters---

   CHARACTER*(*)   COMMAND     ! a command string as defined below.

STANDARD - order as defined in the ITEM_DEF.TXT file
         - this is the default order if this header was not read in
INPUT    - same as in input header
         - this is the default order if this header was read in.

<section names> - 3 character section names, separated by single spaces
                - e.g.:  'FIL ADM COM REC'
                - any sections not specified are automatically added at
                  the end in default order
                - e.g.: 'COM' will move comments to the top, and the
                  other sections will come out in standard order.

----------------------------------------------------------------------
```

10.3 Data IO

The Data IO routines read and write data records, automatically utilizing information about the format of the data records which is stored in the header. They are especially useful when working with data files which have special channels: formatted latitudes, longitudes, dates or times.

Note that the HRC_READ_DATA_R4 and HRC_WRITE_DATA_R4 routines contain _R8 and _I4 entries.

10.3.1 HR_CLOSE_FILE

```
   SUBROUTINE HR_CLOSE_FILE(U)

Closes the file currently open with the unit U.

---Input---
   INTEGER            U           ! unit number

See also: HR_OPEN_FOR_READ, HR_OPEN_FOR_WRITE.

Fortran programs which use the Header library normally look after their
own file opening and closing.  However when the DLL library is used
from a non-Fortran main program, this routine is required to close the
file.
----------------------------------------------------------------------
```

10.3.2 HR_OPEN_FOR_READ

```
   SUBROUTINE HR_OPEN_FOR_READ(FILENAME, U, STAT)

Opens file FILENAME for input, using the first free unit number
greater than 10, and returns that unit number in U.

U must be used in calls to subsequent header routines which
do file handling.

---Input---
   CHARACTER*(*)       FILENAME    ! name of file to open

---Output ---
   INTEGER             U           ! unit number
   INTEGER             STAT        ! Fortran open status
                                   ! 0 if open was successful
                                   ! -1 if error occured determining
                                   !    file format (ioslib FILEFORM)

The file may be FORMATTED or UNFORMATTED.
If you use the HR_READ_DATA_?? header library routine for data input,
you don't need to know whether the file is FORMATTED or not.
If you need to know, use the Fortran INQUIRE statement with the FORM
or FORMATTED option to find out, after using this routine to open the
file.

Fortran programs which use the Header library normally look after their
own file opening and closing.  However when the DLL library is used
from a non-Fortran main program, this routine is required to open the
file.
----------------------------------------------------------------------
```

10.3.3 HR_OPEN_FOR_WRITE

```
   SUBROUTINE HR_OPEN_FOR_WRITE(FILENAME, FORMCHAR, U, STAT)

Opens file FILENAME for output, using the first free unit number
greater than 10, and returns that unit number in U

U must be used in calls to subsequent header routines which
do file handling.

---Input---
   CHARACTER*(*)       FILENAME    ! name of file to open
   CHARACTER*(*)       FORMCHAR    ! Formatted/Unformatted indicator:
                                   ! - only the first character is used
                                   ! - 'F' or 'f' means FORMATTED,
                                   !   else UNFORMATTED is assumed
---Output ---
   INTEGER             U           ! unit number
   INTEGER             STAT        ! Fortran open status
                                   ! 0 if open was successful

Fortran programs which use the Header library normally look after their
own file opening and closing.  However when the DLL library is used
from a non-Fortran main program, this routine is required to open the
file.
----------------------------------------------------------------------
```

10.3.4 HRC_ENCODE_1CHAN_R4

```
   SUBROUTINE HRC_ENCODE_1CHAN_R4(I,DATA4, OUTSTR, S,FW, STAT)
   ENTRY      HRC_ENCODE_1CHAN_R8(I,DATA8, OUTSTR, S,FW, STAT)

Encodes data value DATA4(I) or DATA8(I) into OUTSTR, ready for output accordi
to the header format specifications for channel I.

This routine requires that the TYPE column in the header Channel
Details table be defined for Special Channels.

---Input---
   INTEGER       I         ! channel number

   REAL*4        DATA4(*)  ! data4(i) is the  value to be encoded
   REAL*8        DATA8(*)  ! same, (for R8 entry)
                           ! If type(i) is D, and type(i+1) is T, then
                           ! data?(i+1) may also be used.

---Output---

   CHARACTER*(*) OUTSTR    ! output string.  OUTSTR(:FW) is ready to go
                           !   into an IOS header file output record.
                           !   - data may extend beyond FW
                           !   - leading spaces may be included.
                           !     (at least one for special channels)
   INTEGER       S         ! start column, if coded in the channel detail
                           ! (else zero)
   INTEGER       FW        ! field width - width of column where data
                           !   would be written in an ios header file.
   INTEGER       STAT      ! status. zero if no errors.


START and WIDTH (in the channel details table) are used if they
are specified, but they are optional.

Special channel fields (except Integers) always include a leading space.

If FullTime is to be encoded from a real number, the time zone from
H.FIL.TIME_ZERO then H.FIL.START_TIME will be looked for and used.

Time format for FT, DT and T special channels:
 - if no format or decimal places are coded, time format is HH:MM:SS.SSS
 - if a time format is coded, it is used (on output only).
   Rounding and Normalization will be done as required to the nearest
   hour, minute, second or fraction of a second.
 - if there is no format, decimal places (range 0 to 3) will be used
   on the seconds field.

D followed by T channels:
 - when type(i) is D, and type(i+1) is T, then data(i)+data(i+1) is
   used (or date and time from the DATA_FT structure) to encode date
   and time together, so that time rounding can carry over to the date.
   The time is saved and returned next call (if the channel number matches.)

Character data:
 - must be stored using routine HR_PUT_CDATA, (or put as null-terminated
   strings into common hr_cdata) before this routine is called.
 - The character string written for channel i will be the string
   starting at position NINT(data?) in hr_cdata,  and extending to
   (but not including) the next null.
 - the string will be put in quotes if it has any embedded blanks
   or tabs.  (The quote character comes from ioslib.)
 - if a width is specified for a character channel, the string will be
   truncated to fit, if necessary.
----------------------------------------------------------------------
```

10.3.5 HRC_GET_CDATA

```
   SUBROUTINE HRC_GET_CDATA (iPTR, C, L)

Retrieves from the character channel data buffer hr_cdata the character
string starting at position iPTR.
Use after calling HRC_READ_DATA_?? to retrieve character channel data.

---Input---

   INTEGER         iPTR        ! starting position in hr_cdata

---Output---

   CHARACTER*(*)   C           ! character data returned
                               ! blank if L = 0 or L = -1
   INTEGER         L           ! > 0: length of string put into C
                               ! = 0: null string (null found at iPTR)
                               ! -1 : iPTR invalid, or no null found

If your pointer is stored in a real array, use the NINT function, e.g:
  CALL HRC_GET_CDATA(NINT(data(i)), C, L)

Use HR_GET_CDATA to extract from a specified character buffer instead
of the one in common hr_cdata.

See also: HR_PUT_CDATA, HRC_READ_DATA_SPECIAL_R4
----------------------------------------------------------------------
```

10.3.6 HRC_PUT_CDATA

```
   SUBROUTINE HRC_PUT_CDATA   (C,  iPTR)
   ENTRY HRC_PUT_CDATA_R4(C, r4PTR)
   ENTRY HRC_PUT_CDATA_R8(C, r8PTR)

Stores the character string C into a buffer in common hr_cdata,
in preparation for writing a data record with character channel(s).

---Input---

   CHARACTER*(*)   C           ! character data to be stored
                               ! (null-terminate to preserve trailing blanks)

---Output--- One of:

   INTEGER         iPTR        ! starting position in hr_cdata
   REAL*4          r4PTR
   REAL*8          r8PTR

If channel i is a character channel, and your data is primarily
REAL*4, then:
   REAL*4 data(40)
   CALL HRC_PUT_CDATA_R4('my string',data(i))
   CALL HRC_WRITE_DATA_R4(u,data)

The HR_WRITE_DATA_R4 routine, when it sees that channel i is type 'C'
will convert data(i) to the nearest integer, and extract the null-terminated
```

```
character string starting at that location in the hr_cdata buffer.

The R4 and R8 entries are there to save you having to copy iPTR into
your data array after calling HR_PUT_CDATA.

If you want to store the data in your own character buffer, instead of
the one in common, call HR_*(yourbuf, C, ptr), instead of HRC_.

The trimmed length of C will be stored, unless C is null-terminated,
in which case all characters up to the null will be stored.

Note that the hr_cdata buffer is automatically cleared when a record is
written with a HRC_WRITE_DATA... call.

See also: HRC_GET_CDATA, HRC_WRITE_DATA_SPECIAL_R4
----------------------------------------------------------------------
```

10.3.7 HRC_READ_DATA_R4

```
   SUBROUTINE HRC_READ_DATA_R4(U,RN, DATAR4OUT,RSTAT)
   ENTRY HRC_READ_DATA_R8(U,RN, DATAR8OUT,RSTAT)
   ENTRY HRC_READ_DATA_I4(U,RN, DATAI4OUT,RSTAT)

Reads a data record from unit U, and returns H.FIL.NUMBER_OF_CHANNELS
real numbers in the DATA? array.  The type of data returned (REAL*4,
REAL*8 or INTEGER*4) depends upon the entry point used.
If there are any special channels, the HRC_READ_DATA_SPECIAL_R8
routine is used, and the results converted to the type of the entry
called.

---Input---
   INTEGER     U           ! unit number to read from

---Input/Output---
   INTEGER     RN          ! record number - must be set to zero for
                           ! the first call for this unit number.
                           ! - incremented for each read except EOF.

---Output---                !One of:
   REAL*4      DATAR4OUT(*)     ! data from unit U
   REAL*8      DATAR8OUT(*)     ! data from unit U
   INTEGER*4   DATAI4OUT(*)     ! data from unit U

   INTEGER     RSTAT       ! Fortran Read IO Status:
                           !   0 - all OK
                           !  -1 - end of file
                           !  -2 - insufficient data on record for a
                           !       free-form read
                           !  other values: error: see Fortran manual.

The file attached to unit U must already be open.

The data file may be Formatted or Unformatted (Binary).
If the data file is Formatted, the format is obtained from the header:
  - from the FIL.FORMAT item if it is not null
  - else from the Channel Detail table
    (see routine HRC_DETAIL_TO_FORMAT for details)
  - the data may contain special data types, if they are defined
    in the TYPE column of the Channel Detail table of the header:
       FT - full time    DT - date & time   D - date  T - time
      LAT - latitude    LON - longitude
      (see HRC_READ_DATA_SPECIAL_R4, or IOS Header documentation
       for details.)
  - if the FIL.FORMAT item is 'FREE', or if there are any special
    channel types, the data is read in a non-column-dependent manner.
  - if the FIL.FORMAT item is 'DELIMITED d', the data is extracted from
    delimited fields.  (See HR_DELIM_SCAN for details on
    interpretation of the delimiter indicator d.)
    If a field for a numeric channel is blank or null, the pad value
    for that channel is assumed.

If a blank record is encountered in a formatted file, a warning is
produced, it is counted, but is otherwise ignored.  Reading continues
until EOF or a non-blank records is found.

Note:
   The file type (formatted or unformatted), and the record format, are
   determined or re-determined only under the following conditions:
       - on the first time this routine is called
       - if RN is Zero on input,
       - if the unit number U is different from the last call.

   Reading alternately from two or more data files will be slow,
   because the file type INQUIRE statement, and the format string
   creation will be repeated on every call.

   The type of the data file (formatted or unformatted) is independent
   of the type of the header file.
----------------------------------------------------------------------
```

10.3.8 HRC_READ_DATA_SPECIAL_R4

```
     SUBROUTINE HRC_READ_DATA_SPECIAL_R4(U, RN, DATA4,RSTAT)
     ENTRY      HRC_READ_DATA_SPECIAL_R8(U, RN, DATA8,RSTAT)

  Reads a data record from unit U, and returns one REAL value per
  channel in DATA4 or DATA8.
  - can be used to read data with special channels coded in non-real
    form.
  - can be used to read data in a non-column-dependent manner
  - can be used to read data when there is no format information in
    the header.

  ---Input---
     INTEGER     U           ! unit number to read from

  ---Input/Output---
     INTEGER     RN          ! record number, incremented

  ---Output---
     REAL*4      DATA4(*)    ! data from unit U
     REAL*8      DATA8(*)    ! data from unit U (for R8 entry)

     INTEGER     RSTAT       ! Fortran Read IO Status:
                             !   0  - all OK
                             !  -1  - end of file
                             !  777 - decode error

  The file attached to unit U must already be open.
  The data file must be Formatted.

  START and WIDTH may be omitted, for non-column-dependent reading.
  If a Date channel (type D) has embedded blanks, then WIDTH must be
  specified.
```

```
For Real and Integer channels, a width is considered to be
"explicitly specified" if the WIDTH is specified OR if there is a width
specified in the FORMAT.  Eg. if WIDTH is null, but the format is
I7 or F7.3, then the width is considered "specified" as 7.
(See routine HRC_CHANNEL_FORMAT for which is used if both are specified.)

For non-integer special channels, a width (including a leading space)
may be obtained from a format (eg 6 from 'HH:MM').  A width so-obtained
will maintain fixed format, but has a side-effect: see
"Column-dependent reading" below.

The Type column of the Channel Details table in the header
should be defined for each channel.  The Format column is optional.
The Start and Width information will be used if it is specified.

Format        Type       Dec_pl
-------      ------     --------
 n/a           FT         n/a     Full time: ZON YYYY/MM/DD HH:MM:SS.SSS
 n/a           DT         n/a     date & time:   YYYY/MM/DD YY:MM:SS.SSS
YYYY/MM/DD     D          n/a     date, any IOSLIB date format
HH:MM:SS.SSS   T          0-7     time: format: any mask using H,M,S
                                                default is shown
                                  dec_pl: controls seconds output if
                                          format is null
DMH or DMSH    LAT        0-7     dec_pl controls output accuracy of ...
 "  "   "      LON         "      ... last field - Minutes or Seconds
F, E, or full R4          1-7     real*4, dec_pl used if single char fmt
F, E, or full R8          1-16    real*8,   "    "    "    "     "    "
null, Q or NQ C           n/a     Q, null: quotes removed on input,
                                  NQ: - no quotes looked for or removed.
                                      - if freeform, embedded blanks will
                                        cause errors.
             <null>               R4 or R8 is assumed


Reading may be either column-dependent or free-form.  If the width
field of channel 1 is specified, reading starts column-dependent.
As soon as a null width (*) is encountered, reading becomes free-form.
If a Start column is specified, free-form scanning moves to that
column.  Reading only becomes completely column-dependent again if
both a start column AND a width are specified.  (This is because
a width doesn't mean much if it is not clear where that width starts!)
With the first channel a start column is not required because the
width is assumed to start at column 1.

* T, DT and FT special channels, the (time) format is ignored -
  whatever is coded is read: to the specified width, if specified.

Column-dependent reading:
  If a special channel has an explicit width, it is assumed that it
  occupies that entire width, and a following channel will be scanned
  for after that width.
  If a non-integer special channel's width is not explicit, i.e. is taken
  from the format or the default for the type, then the channel MAY occupy
  less space, and the following channel will be scanned for immediately
  following the ACTUAL space occupied.  This means that if following
  channels are in consistent columns, that the special channel must
  have a consistent actual width.
  E.g. if a file has channels:
    Time  - with no width coded in the header,
          - format is either HH:MM:SS, or longer, or is null (so that the
            default format of HH:MM:SS.SSS is used)
    X     - Real, format F7.2

  then '-------' shows where X is looked for in the following records:
       12:30:30 -99.99         - will read OK
               -------
       12:30 -99.99            - will read OK
            -------
       12:30    -99.99         - will will NOT read OK (will read X as -99)
            -------

  If the Time channel had a width of 9 specified in the header
  (8 + 1 for a leading space), then the 1st & 3rd data record above
  would read OK, and the 2nd would not.
  (This "feature" was designed for reading a data file encountered
  in September 1998.  If you don't like this effect, code the Width!)

  Real and Integer channels, on the other hand, are assumed to
  occupy their entire stated width, so given 2 channels X and Y with
  F7.2 formats:
       -99.99 -88.88           - will read OK
       -99.   -88.88           - will read OK
       -99. -88.88             - will NOT read OK.

For Character data, the data value returned is a pointer to a
null-terminated string in character string hr_cdata (in common hr_cdata).
The character data can be retrieved using routine HR_GET_CDATA.
For non-freeform data, quotes (single or double) are optional in all
cases, and will be removed.  For freeform data, quotes (single only)
are mandatory if there are embedded blanks in the character data, but
otherwise optional.  Quotes will be removed.

If a blank record is encountered, a warning is
produced, it is counted, but is otherwise ignored.  Reading continues
until EOF or a non-blank records is found.

Real Date and Time values returned:
 - if the header has no time zero, the fixed time zero is used.
 FT, DT   - days since header time zero, (if no header time zero, then
            fixed time zero.)
 D        - days since time 00:00 of date of header time zero.
 T        - time as absolute fraction of a day

 DELIMITED data:
 - indicated by FORMAT item = "DELIMITED d"
   where d is the delimiter (default=comma), or COMMA or TAB
 - non-character data is scanned for within each delimited field
   as in FREEFORM mode.
 - character data may be unquoted, or quoted with single or double quotes
 - null fields result in pad values for real and integer channel types,
   a blank for character, and an ERROR for others.
----------------------------------------------------------------------
```

10.3.9 HRC_WRITE_DATA_R4

```
   SUBROUTINE HRC_WRITE_DATA_R4(U,RN, DATAR4IN,WSTAT)
   ENTRY HRC_WRITE_DATA_R8(U,RN, DATAR8IN,WSTAT)
   ENTRY HRC_WRITE_DATA_I4(U,RN, DATAI4IN,WSTAT)

Writes a record containing H.FIL.NUMBER_OF_CHANNELS data values
to unit U.  The type of data provided may be REAL*4, REAL*8 or
INTEGER*4, depending upon the entry point (subroutine name) used.
The data will be converted, if necessary, to the data type specified
```

```
in the header.
If there are any special channels, the HRC_WRITE_DATA_SPECIAL_R8
routine is used.

---Input---
   INTEGER     U           ! unit number to write to

---Input/Output---
   INTEGER     RN          ! record number - must be set to zero for
                           ! the first call for this unit number.
                           ! - incremented for each write

---Input---   (one of:)
   REAL*4      DATAR4IN(*) ! data to be written - number of elements
                           ! to write must already be in
                           ! H.FIL.NUMBER_OF_CHANNELS
   REAL*8      DATAR8IN(*)
   INTEGER*4   DATAI4IN(*)

---Output---
   INTEGER     WSTAT       ! Fortran Write IO Status:
                           !   0 - all OK
                           !  other values: error: see Fortran manual.

The file attached to unit U must already be open.

The data file may be Formatted or Unformatted (Binary).
If the data file is Formatted, the format is obtained from the header:
  - from the FIL.FORMAT item if it is not null
  - else from the Channel Detail table
    (see routine HRC_DETAIL_TO_FORMAT for details)

Note:
   The file type (formatted or unformatted), and the record format, are
   determined or re-determined only under the following conditions:
       - on the first time this routine is called
       - if RN is Zero on input,
       - if the unit number U is different from the last call.

   Writing alternately to two or more data files will be slow, because
   the file type INQUIRE statement, and the format string creation will
   be repeated on every call.

   The type of the data file (formatted or unformatted) is independent
   of the type of the header file.
----------------------------------------------------------------------
```

10.3.10 HRC_WRITE_DATA_SPECIAL_R4

```
   SUBROUTINE HRC_WRITE_DATA_SPECIAL_R4(U,DATA4, WSTAT)
   ENTRY      HRC_WRITE_DATA_SPECIAL_R8(U,DATA8, WSTAT)

Writes a data record to unit U, taking one REAL value per
channel from DATA4 or DATA8.

Can be used to:
- write data with special channels in non-real form,
  such as Character, Full Time, Date, Time, Latitude and Longitude.
- write data in a non-column-dependent manner
- write data when there is no format information in
  the header.
- write data in delimited form, using comma or other delimiter.
```

```
This routine requires that the TYPE column in the header Channel
Details table be defined for Special Channels.

---Input---
   INTEGER     U           ! unit number to write to

   REAL*4      DATA4(*)    ! data to be written to unit U
   REAL*8      DATA8(*)    ! same, (for R8 entry)

---Output---

   INTEGER     WSTAT       ! write status. zero if no errors.

The file attached to unit U must already be open.
The data file must be Formatted.

For more details, see the notes in routine HRC_ENCODE_1CHAN_R4.FOR

----------------------------------------------------------------------
```

10.4 Encode

10.4.1 HR_ENCODE

```
   SUBROUTINE HR_ENCODE(TYPE,D, STR,STAT)

Encodes data of TYPE, in structure D, into string STR.

---Input---
   CHARACTER*(*)       TYPE        ! data type of the input variable

   INCLUDE 'ios_header$inc:header_basics.inc'
   INCLUDE 'ios_header$inc:alltypes.inc'
   RECORD /ALLTYPES/ D             ! overlaid storage of all types

---Output---

   CHARACTER*(*)       STR         ! output string
   INTEGER             STAT        ! 0 - all OK
                                   ! 3 - encoding error (ie STR to short
                                   !     for a numeric type.
                                   ! 4 - invalid type

Note: Output will be left_justified in STR.
----------------------------------------------------------------------
```

10.4.2 HR_ENCODE_F4

```
   SUBROUTINE HR_ENCODE_F4(IN,STR,Q)

Encodes REAL*4 IN into string STR, left justified.
Numeric null values are are translated into character null values.
If Q='Q' and STR would be blank on return, STR is set to a blank in quotes.

---Parameters---
   REAL*4          IN
   CHARACTER*(*)   STR
   CHARACTER*1     Q
----------------------------------------------------------------------
```

10.4.3 HR_ENCODE_F8

```
   SUBROUTINE HR_ENCODE_F8(IN,STR,Q)

Encodes REAL*8 IN into string STR.
Numeric null values are are translated into character null values.
If Q='Q' and STR would be blank on return, STR is set to a blank in quotes.

---Parameters---
   REAL*8          IN
   CHARACTER*(*)   STR
   CHARACTER*1     Q

----------------------------------------------------------------------
```

10.4.4 HR_ENCODE_FT

```
   SUBROUTINE HR_ENCODE_FT(FT, OUTSTR,MES)

Encodes a Full Time into string OUTSTR.
Header null values are translated to character.

---Parameters---
---Input---
   INCLUDE 'ios_header$inc:sr_fulltime.inc'
   RECORD /SR_FULLTIME/FT          ! full time

---Output---
   CHARACTER*(*)       OUTSTR      ! output full-time string
   CHARACTER*(*)       MES         ! status message: all OK if blank

If OUTSTR is shorter than an encoded full-time (27 characters),
it will be truncated without complaint (ie MES will still be blank).

See also: function HR_FT_STRING
----------------------------------------------------------------------
```

10.4.5 HR_ENCODE_I

```
   SUBROUTINE HR_ENCODE_I(IN,STR,Q)

Encodes default size Integer IN into string STR.
Numeric null values are are translated into character null values.
If Q='Q' and STR would be blank on return, STR is set to a blank in quotes.

---Parameters---
   INTEGER         IN
   CHARACTER*(*)   STR
   CHARACTER*1     Q
----------------------------------------------------------------------
```

10.4.6 HR_ENCODE_I2

```
   SUBROUTINE HR_ENCODE_I2(IN,STR,Q)

Encodes Integer*2 IN into string STR.
Numeric null values are are translated into character null values.

---Parameters---
```

```
   INTEGER*2       IN
   CHARACTER*(*)   STR
   CHARACTER*1     Q

----------------------------------------------------------------------
```

10.4.7 HR_ENCODE_I4

```
   SUBROUTINE HR_ENCODE_I4(IN,STR,Q)

Encodes Integer*4 IN into string STR.
Numeric null values are are translated into character null values.
If Q='Q' and STR would be blank on return, STR is set to a blank in quotes.

---Parameters---
   INTEGER*4       IN
   CHARACTER*(*)   STR
   CHARACTER*1     Q
----------------------------------------------------------------------
```

10.4.8 HR_ENCODE_LL

```
   SUBROUTINE HR_ENCODE_LL(LL,STR)

Encodes a Latitude or Longitude into string STR.
Header null values are translated to character.

---Parameters---

   INCLUDE 'ios_header$inc:sr_latlon.inc'
   RECORD /SR_LATLON/LL            ! Latitude or Longitude

   CHARACTER*(*)       STR         ! output string

----------------------------------------------------------------------
```

10.4.9 HR_ENCODE_LLS

```
   SUBROUTINE HR_ENCODE_LLS(LL,NDEC,LJUST, STR)

Encodes a Latitude or Longitude into string STR.
Header null values are translated to character.
This routine is suitable for non-header output of a lat. or long.

---Parameters---

   INCLUDE 'ios_header$inc:sr_latlon.inc'
   RECORD /SR_LATLON/  LL          ! Latitude or Longitude

   INTEGER             NDEC        ! number of decimal places on minutes
   LOGICAL*1           LJUST       ! deg & min are left justified
   CHARACTER*(*)       STR         ! output string

This routine differs from HR_ENCODE_LL as follows:

  - a blank string is returned if LL is null-absent
  - only one space is left between the degrees and minutes numbers
  - the number of decimal places of minutes can be controlled
  - left-justification is an option (ie no leading spaces)

Call with LJUST:
  True - if putting lat & lon beside one another, to make each string
         as short as possible.
  False- if putting lat & lon underneath each other, to make dec pt
         and hemisphere characters line up.
----------------------------------------------------------------------
```

10.4.10 HR_ENCODE_S

```
   SUBROUTINE HR_ENCODE_S(IN,STR)

Copies string IN into STR, adding quotes if IN has leading or
embedded blanks, or is entirely blank.

---Parameters---
   CHARACTER*(*)   IN
   CHARACTER*(*)   STR
----------------------------------------------------------------------
```

10.4.11 HR_ENCODE_TI

```
   SUBROUTINE HR_ENCODE_TI(TI,STR)

Encodes a Time Increment into STR
Header null values are translated to character.

---Parameters---

   INCLUDE 'ios_header$inc:sr_time_increment.inc'
   RECORD /SR_TIME_INCREMENT/TI

   CHARACTER*(*)       STR         ! output string

----------------------------------------------------------------------
```

10.4.12 HR_FT_STRING

```
   CHARACTER*(*) FUNCTION HR_FT_STRING(FT)

Returns full time FT as a character string.

---Input---

   INCLUDE 'ios_header$inc:sr_fulltime.inc'
   RECORD /sr_fulltime/ FT

Use subroutine HR_ENCODE_FT if you need diagnostics in case of
encoding problems.

----------------------------------------------------------------------
```

10.4.13 HR_Z8_STRING

```
   CHARACTER*(*) FUNCTION HR_Z8_STRING(Z8,zone)

Returns Z8 (days from fixed time zero) as a full time character string.
Declare length 27 to get milliseconds.
If declared shorter, the string will be trunctated, NOT ROUNDED.

---Input---
```

```
   REAL*8          Z8
   CHARACTER*3     zone        ! time zone to convert to.
                               ! 'hdr' to use ts_hdr_zone from hr_tsz.inc


Use subroutines HR_Z8_TO_FT and HR_ENCODE_FT if you need diagnostics
in case of encoding problems.

----------------------------------------------------------------------
```

10.5 Get INfo Out

10.5.1 HRC_CHANNEL_ID

```
   CHARACTER*(*) FUNCTION HRC_CHANNEL_ID(I,OPT)

Returns the channel ID "name [units]" for channel I of the header.

---Input---

   INTEGER         I           ! channel number
   CHARACTER*(*)   OPT         ! option characters (case sensitive).

OPT: - option characters may occur anywhere in OPT
     - case sensitive
     N - suppress null units
       - i.e. you'll get "name" instead of "name []" if units is
         blank or any of the other null values.
     A - include units only if required to make the ID unique.
       - i.e. you'll get just "name" if "name" is unique in the header.
     S - if no units are required to get a unique ID, then the name
         is returned, truncated to the minimum length >= 4 which
         is unique to that length.
       - e.g. if there is just one channel named "Temperature",
         the result will be "Temp".

Note that name uniqueness checks are NOT case sensitive.

Note to programmers: when adding option characters, make sure they
not conflict with options of function HR_MAKE_CHANNEL_ID, so that
OPT can be passed unmodified to that function.

See also function HR_MAKE_CHANNEL_ID, HR_MOD_CHANNEL_ID

----------------------------------------------------------------------
```

10.5.2 HRC_CHANNEL_POS

```
   SUBROUTINE HRC_CHANNEL_POS( CHAN_ID, PMAX, POS, N)

Returns a list of channel positions (numbers), of channels in the
header which have names and units which match channel ID CHAN_ID.

---Input---

   CHARACTER*(*)   CHAN_ID     ! channel ID to search for: eg "Name [Units]"
                               ! - [Units] is optional.  If present,
                               !   a case sensitive units match is done.
                               ! - name and units may start and/or end
                               !   with * for wild card search.
   INTEGER         PMAX        ! maximum number to find (size of POS)

---Output---
   INTEGER         POS(PMAX)   ! channel positions (numbers)
   INTEGER         N           ! number of positions returned in POS.
                               ! - zero if no channels names were found
                               !   matching CHAN_ID

The channel name search is NOT case sensitive.
The (optional) units search IS case sensitive.

Searches the channel table in the FILE section only.

Examples:  CHAN_ID        matches Name: 'Temperature' Units: 'Celsius' ?
          'Temperature'                  yes
          'Temp*'                        yes
          'temp'                         no
          '*ture'                        yes
          '*tur'                         no
          'Temperature [Kelvin]'         no
          'Temperature [Celsius]'        yes
          'Temperature[Celsius]'         yes
          'Temperature   [Celsius]'      yes
          'Temp* [C*]'                   yes
          '*ture [*]'                    yes

See also function HRC_CHANNEL_POS1.

----------------------------------------------------------------------
```

10.5.3 HRC_CHANNEL_POS1

```
   INTEGER FUNCTION HRC_CHANNEL_POS1(CHAN_ID, OPT)

Returns the position (number) of the first channel in the header
whose name matches CHAN_ID.  Zero is returned if there is no match.

---Input---

   CHARACTER*(*)   CHAN_ID     ! channel ID to search for: eg "Name [Units]"
                               ! - [Units] is optional.  If present,
                               !   a case sensitive units match is done.
                               ! - name and units may start and/or end
                               !   with * for wild card search.

   CHARACTER*1     OPT         ! if '*', same as if the NAME part of
                               ! CHAN_ID ended with '*'.


Calling with CHAN_ID='temp   ' and OPT = '*' has the same effect as
calling with CHAN_ID='temp*  ' and OPT = ' '.

The channel name search is NOT case sensitive.
The (optional) units search IS case sensitive.

Searches the channel table in the FILE section only.

Examples:  CHAN_ID        matches Name: 'Temperature' Units: 'Celsius' ?
          'Temperature'                  yes
          'Temp*'                        yes
          'temp'                         no
          '*ture'                        yes
          '*tur'                        no
          'Temperature [Kelvin]'        no
          'Temperature [Celsius]'       yes
          'Temperature[Celsius]'        yes
          'Temperature   [Celsius]'     yes
          'Temp* [C*]'                  yes
          '*ture [*]'                   yes
---

See also subroutine HRC_CHANNEL_POS

----------------------------------------------------------------------
```

10.5.4 HRC_CHARACTER_CHANNELS

```
   LOGICAL*1 FUNCTION HRC_CHARACTER_CHANNELS()

Returns a TRUE value if there are any character channel types
specified in the channel detail table.

----------------------------------------------------------------------
```

10.5.5 HRC_DETAIL_TO_FORMAT

```
    SUBROUTINE HRC_DETAIL_TO_FORMAT(FMT,LSTAT)

Creates a Fortran Format string from the format information in the
FILE section Channel table detail.

---Output---
   CHARACTER*(*)       FMT     ! format string returned
                               ! includes parentheses
   INTEGER             LSTAT   ! > 0: success!
                               !      LSTAT is trimmed length of FMT
                               ! -1 : 'A' formats inserted for special
                               !      channels
                               ! -2 : error messages produced

Rules Used when interpreting the detail information:

- if FORMAT is only 1 character:
    - it is assumed it is E, F, I or A,
    - the WIDTH and DECIMAL_PLACES fields will be used to construct
      the format specifier.
    - if DECIMAL_PLACES are omitted, 0 is assumed
    - if WIDTH is zero or null, 15 is assumed

- if FORMAT is longer than 1 character:
    - it is assumed that it contains a complete format specifier
    - DECIMAL_PLACES will be ignored
    - if WIDTH is specified, and is greater than the width specified
      within FORMAT, an 'nX' format item will be added, after FORMAT,
      where n is the extra number of spaces
    - if WIDTH is specified, and <= the width within FORMAT, WIDTH
      is ignored

- if START is specified:
    - if START = the sum of the preceding field widths, nothing
      is done.
    - if START is NOT = the sum of the preceding field widths, a
      T (Tab) format item is inserted to ensure that the field starts
      in the specified column.

- if a non-integer special channel type is encountered:
    - an A format is inserted.
    - if the width is specified, it is put after the 'A'.


----------------------------------------------------------------------
```

10.5.6 HRC_FINDTIME

```
    SUBROUTINE HRC_FINDTIME(timetype,chnum1,chnum2)

 Analyses the header in common, and determines if and how the time
 of each data record is defined.

 ---Output---

    INTEGER     timetype    ! 0 - no record time available
                            ! 1 - Special channel(s) (ASCII yyyy/mm/dd etc.)
                            !   - one of Channel Detail Types
                            !     D, FT, DT or D and T found.
                            !     (T alone does not count.)
                            ! 2 - OFFSET from TIME ZERO: a single real number
                            !     time channel, in TIME_UNITS
                            ! 3 - DAY_OF_YEAR (DOY), a single real number
                            !     time channel.
                            ! 4 - Calculate: No time channel: record time
                            !     can be calculated from START_TIME and
                            !     TIME_INCREMENT.

    INTEGER     chnum1      ! if timetype = 1, 2 or 3, channel number of
                            ! time channel.  If there are separate date
                            ! and time channels, chnum1 will be date, and
    INTEGER     chnum2      ! chnum2 will be time.
                            ! If a special channel of type T is found but
                            ! no type D, timetype and chnum1 will be 0,
                            ! but chnum2 will be the number of the
                            ! time channel.
                            ! If there are multiple special date/time chans,
                            ! chnum1 & chnum2 will indicate the first ones.

 The timetype is determined as follows:

 1. Check for Special:

     - check the Type column of the channel details table for any of
        FT, DT, D or T.
     - if there are multiple special time channels, the first one(s)
       found will be returned in chnum1 (and chnum2).
     - don't even need to know what the channel numbers they are,
       or what type of special time channels because
       HRC_READ_DATA_R4 will read whatever is there, and put the full
       time of the data record
       into fulltime structure DATA_FT in common FTLATLON.

 2. Else Check for OFFSET or 3. DAY_OF_YEAR

     - if TIME_UNITS is defined, search the Channel table for
       a channel with UNITS = TIME_UNITS and = one of the
       IOS Header time units:
          DAY_OF_YEAR, DAYS, HOURS, MINUTES, SECONDS
     - if not found, look for a channel name containing TIME
       (a bit flakey)
     - if found, then we know what the channel is, and how to
       interpret the time.  For OFFSET, a time zero is also required.

 4. Else Check for Calculate:
     - if header items TIME_INCREMENT and START_TIME are both
       defined.

 Else: no time data!

 Note: this routine will NOT find the situation where there are two
       separate REAL channels: 1 for date and 1 for time.
 ----------------------------------------------------------------------
```

10.5.7 HRC_GET_DATATYPE_AND_FORMAT

```
   SUBROUTINE HRC_GET_DATATYPE_AND_FORMAT(DATATYPE,FORMAT)

Returns the data type and format from the header in common.

---Output---
   CHARACTER*(*)   DATATYPE    ! R4 R8 I2 I4 - all channels same type
                               ! SP          - mixed or special types
                               ! **          - error
   CHARACTER*(*)   FORMAT      ! a Fortran format string, including brackets
                               ! blank - unformatted, according to the
                               !         FILE TYPE item.
                               ! FREE  - header format is FREE
                               ! DELIMITED [delimiter]
                               !       - as coded in header, but upper case

For Unformatted files:
  - all channels must have the same data type.
  - if the DATA_TYPE item is not specified, R4 is assumed.


----------------------------------------------------------------------
```

10.5.8 HRC_GET_FORMAT

```
   SUBROUTINE HRC_GET_FORMAT(FMT,FLEN)

Returns H.FIL.FORMAT if it is non-null.  Otherwise a format string is
constructed from the Channel Detail table,

---Output---
   CHARACTER*(*)       FMT     ! format string returned

   INTEGER             FLEN    ! > 0: success - trimmed length of FMT
                               ! -1 : warning messages produced
                               ! -2 : error messages produced

See HRC_DETAIL_TO_FORMAT for details of how a format is created from
the Channel Details table.
----------------------------------------------------------------------
```

10.5.9 HRC_GET_ITEM_I

```
   SUBROUTINE HRC_GET_ITEM_I(SECTION,LABEL, IOUT,ITEMTYPE,STAT)
```

```
Subroutines and entries HRC_GET_ITEM_<type> find the item identified
by SECTION and LABEL in the header structure, and return the value
of that item in variable <type>OUT.

Underscores instead of blanks are acceptable in multi-word labels.
E.g. "NUMBER OF RECORDS" is equivalent to "NUMBER_OF_RECORDS".

If <type> is not the same as the type of the item, conversion is done
if possible.

The type of the item found is returned in ITEMTYPE.

---Input---
   CHARACTER*(*)       SECTION     ! header section
                                   ! (1st 3 chars needed only)
   CHARACTER*(*)       LABEL       ! item label

   ! (the following statement is located here only so that the
   !  structure definitions are available for the output variables.)
   INCLUDE 'ios_header$inc:header_common.inc'

---Output---
   ! one of the following OUT variables, depending upon <type>
       INTEGER                 IOUT
       INTEGER*2               I2OUT         ! - set if STAT <= 1
       INTEGER*4               I4OUT         ! - may be one of the
       REAL*4                  R4OUT         !   header null values.
       REAL*8                  R8OUT
       RECORD/SR_FULLTIME/     FTOUT
       RECORD/SR_TIME_INCREMENT/ TIOUT
       RECORD/SR_LATLON/       LLOUT
       CHARACTER*(*)           COUT

   CHARACTER*2         ITEMTYPE  ! data type of item
                                 ! (C I I2 I4 R4 R8 FT LL TI)
   INTEGER             STAT      ! see details below


     Valid   Found in
STAT Label    Header    Notes
-----------------------------------------------------------------------
 0 -  Yes      Yes      Value: a non-null data value is returned
 1 -  Yes      Yes      Value: null-blank, null-n/a or null-unknown
 2 -  Yes      No       Value: null-absent
 3 -  Yes      Yes      ERROR: trying to decode a value from character
 4 -  Yes      Yes      ERROR: incompatible types - could not convert
 5 -  No       No       ERROR: invalid section name or item label

Type Conversions:
----------------
   - if the type of the item is not the same as the type of the entry
     called, a conversion is done if possible.
   - if no conversion is possible, STAT is returned 3, and the output
     variable is returned null-blank.
   - default Fortran conversions are done between all I and R types.
   - conversion to and from CHARACTER are done using the HR_ENCODE and
     HR_SCAN routines.  Thus the same results as for writing and
     reading a header are obtained.
   - LL (Lat/Lon) can be converted to/from R4 and R8 - the real number
     is assumed to be degrees, with negative values used for South and
     West hemispheres.
   - TI can be converted to/from R4 and R8 - the real number is assumed
     to be in Days.
```

```
GET_ITEM routine calling structure:

                         HRC_GET_ITEM_<type>     \     both in
                               |                   >   HRC_GET_ITEM_I
                         HR_GET_ITEM_<type>      /               .FOR
                           /              \
                          /                \
                     HR_GET_ITEM          HR_CONVERT
                     /        \                 \
             HR_LOOKUP_ITEM  HR_FETCH            \
               /       \         |             HR_ENCODE
              /         \    HR_COPY_BYTES     HR_SCAN
             /           \                     HR_SET_NULL
HR_LOOKUP_ITEM_INDEX  HR_LOOKUP_CUSTOM_ITEM    HR_TI_TO_R8
                                               HR_R8_TO_TI

----------------------------------------------------------------------
```

10.5.10 HRC_GET_LAT_DEG8

```
   SUBROUTINE HRC_GET_LAT_DEG8(DEGREES8)

Returns H.LOC.LATITUDE from the header, as a single real*8 number,
in degrees.  The result will be negative for South latitudes.

---Output---
   REAL*8            DEGREES8      ! latitude in degrees

See also: HRC_GET_ITEM_I, entries _R8, _C, and _LL.
----------------------------------------------------------------------
```

10.5.11 HRC_GET_LON_DEG8

```
   SUBROUTINE HRC_GET_LON_DEG8(DEGREES8)

Returns H.LOC.LONGITUDE from the header, as a single real*8 number,
in degrees.  The result will be negative for West longitudes.

---Output---
   REAL*8            DEGREES8      ! longitude in degrees

See also: HRC_GET_ITEM_I, entries _R8, _C, and _LL.
----------------------------------------------------------------------
```

10.5.12 HRC_GET_PAD_VECTOR

```
   SUBROUTINE HRC_GET_PAD_VECTOR(PADV)

Subroutines and entries HRC_GET_PAD_VECTOR returns a pad value in
PADV for each channel.

---Output---

   REAL*4  PADV(*)


For each channel, the pad value from the details table is inserted if
there is one there, else the pad value from the PAD item is used.

NOTE:
  If there is no pad value available, the header null value HN_BLANK
  will be inserted.  (I.e. null values HN_ABSENT, HN_NA, HN_UNKNOWN
  will NOT be returned, even if the pad value is specified as such
  in the header.)

----------------------------------------------------------------------
```

10.5.13 HRC_GET_TIME_INCREMENT

```
   SUBROUTINE HRC_GET_TIME_INCREMENT(UNITS,OUT8)

Returns H.FIL.TIME_INCREMENT from the header, as a single real*8
number, in the units specified in string UNITS.

---Input---
   CHARACTER*(*)     UNITS         ! DAY, HOUR, MIN, SEC, MS
                                   ! any case, only 1st 2 chars used,
                                   ! HR = HOUR
---Output---
   REAL*8            OUT8          ! time increment

See also: HR_TI_T0_R8, HR_R8_TO_TI
----------------------------------------------------------------------
```

10.5.14 HRC_GETSEC_FIL

```
   SUBROUTINE HRC_GETSEC_FIL(FIL)

   INCLUDE 'ios_header$inc:header_common.inc'
   RECORD /SR_FILE/            FIL     ! out

Copies the FILE section from the header-in-common into
structure FIL.

This source file contains GET and PUT SEC routines for all header
sections.
These routines were created to allow a Visual Basic program to
get a section of the header, since the entire header structure is
too big for Visual Basic.

              ---- Name ----          --- Arg Type ---

   SUBROUTINE HRC_GETSEC_FIL (FIL)     SR_FILE
   SUBROUTINE HRC_PUTSEC_FIL (FIL)
   SUBROUTINE HRC_GETSEC_ADM (ADM)     SR_ADMINISTRATION
   SUBROUTINE HRC_PUTSEC_ADM (ADM)
   SUBROUTINE HRC_GETSEC_LOC (LOC)     SR_LOCATION
   SUBROUTINE HRC_PUTSEC_LOC (LOC)
   SUBROUTINE HRC_GETSEC_DEP (DEP)     SR_DEPLOYMENT
   SUBROUTINE HRC_PUTSEC_DEP (DEP)
   SUBROUTINE HRC_GETSEC_REC (REC)     SR_RECOVERY
   SUBROUTINE HRC_PUTSEC_REC (REC)
   SUBROUTINE HRC_GETSEC_INS (INS)     SR_INSTRUMENT
   SUBROUTINE HRC_PUTSEC_INS (INS)
   SUBROUTINE HRC_GETSEC_HIS (HIS)     SR_HISTORY
   SUBROUTINE HRC_PUTSEC_HIS (HIS)
   SUBROUTINE HRC_GETSEC_RAW (RAW)     SR_RAY
   SUBROUTINE HRC_PUTSEC_RAW (RAW)
   SUBROUTINE HRC_GETSEC_CAL (CAL)     SR_CALIBRATION
   SUBROUTINE HRC_PUTSEC_CAL (CAL)
```

```
----------------------------------------------------------------------
```

10.5.15 HRC_SPECIAL_CHANNELS

```
   LOGICAL*1 FUNCTION HRC_SPECIAL_CHANNELS()

Returns a TRUE value if there are any special channel types
(non-real data such as dates, times, lats, lons) specified in the
channel detail table.

----------------------------------------------------------------------
```

10.6 Put INfo IN

10.6.1 HR_SET_FORMAT

```
   SUBROUTINE HR_SET_FORMAT(channel, new_format)

Given the header channel structure, and a new_format string, the
detail format fields are set, and in some cases the Type is set.

Note that for special channels, if the width and decimal places
are not (or can not) be specified, they are left unchanged.

Channel Type is changed ONLY in the following cases:
   - If an I format is specified, the channel type will be set to I.
   - If a non-special, non-I format is specified for a channel which
     is of type I, then the type is changed to blank (real).

For character channels, new_format may be coded as follows:
    ffww   or  ff ww
  where ff is NQ, Q, C or A which will cause the format to be set
  as follows:
      NQ - NQ
      A  - NQ
      Q  - Q
      C  - ' '
  Both ff and ww are optional.
  E.g. Q12   will set the format to Q, and the width to 12
       NQ 10 will set the format to NQ, and the width to 10.
       A10   will set the format to NQ, and the width to 10.
       8     will leave format unchanged, and the width to 8
       Q     will set the format to Q, and leave the width unchanged.
       C9    will set the format to blank, and the width to 9.
  (not case dependent).

For Lat and Lon channels, new_format may be coded as:
       fd or f d
  where f is DMH, DMSH or omitted
    and d is a number of decimal places (optional)
  If d is omitted, the decimal places will not be changed.

For other special channels, new_format is simply copied into
channel.format.  (For Time channels, decimal places is set blank.)

For non-special channels, the new_format is separated into the
channel format, width and decimal places.

----------------------------------------------------------------------
```

10.6.2 HRC_ADD_FILE_CHANNEL

```
   SUBROUTINE HRC_ADD_FILE_CHANNEL(NAME,UNITS,MINVAL,MAXVAL, CHN)

Adds a channel to the CHANNEL table of the FILE section of the
header.  Item H.FIL.NUMBER_OF_CHANNELS is incremented.

---Input---
   CHARACTER*(*)   NAME        ! channel name
   CHARACTER*(*)   UNITS
   REAL*4          MINVAL      ! use header null values to leave out
   REAL*4          MAXVAL

---Output---
   INTEGER         CHN         ! channel number of the new channel
                               ! = H.FIL.NUMBER_OF_CHANNELS
                               ! zero if channel table is full

The returned channel number (CHN) is useful if you want to call
HRC_SET_CHANNEL_DETAIL immediately after calling this routine.

If this routine is unsuccessful (i.e. if the channel table is full)
a warning message is produced, and CHN is returned zero.

-----------------------------------------------------------------------
```

10.6.3 HRC_FORMAT_TO_DETAIL

```
   SUBROUTINE HRC_FORMAT_TO_DETAIL(FMT,STATUS)

Takes a Fortran Format string and copies the information into the
FILE section Channel table detail.

---Input---
   CHARACTER*(*)       FMT     ! format string
                               ! parentheses optional
---Output---
   INTEGER             STATUS  ! 0 - no trouble
                               ! 1 - warning(s) printed
                               ! 2 - error(s) printed

The surrounding brackets on FMT are optional.
Embedded blanks are ignored.

FMT may contain X and T format specifiers.  These will be used to set
the Start field in the Detail table.

TYPE column: not changed except for:
  - if the format type is I, the TYPE is set to I
  - if the format type is A, the TYPE is set to C, and FORMAT to NQ.

START column:
  If there are no X or T format specifiers, the START column will
  be set null-blank.  (Prior to Feb 14, 2001, START was always set.)
  If you NEVER want the START column set, use HRC_FORMAT_TO_DETAIL2

A format item may contain a repetition factor (e.g. '4F6.2'), in
which case the format will be applied to the next 4 channels.

Repetition factors for multiple items, with parenthases, can
NOT be handled (e.g. '4(F5.2,1X)' )

If there is any trouble parsing a format specifier, the specifier
```

```
will not be split up into the Format, Width and Decimal_Places fields.
Instead, the whole specifier will be put into the Format field,
and the other fields will be set to null-blank.
(And a warning will be printed.)

The H.FIL.INCLUDE_CHANNEL_DETAIL flag is turned on.
----------------------------------------------------------------------
```

10.6.4 HRC_FORMAT_TO_DETAIL2

```
   SUBROUTINE HRC_FORMAT_TO_DETAIL2(FMT,STATUS)

Takes a Fortran Format string and copies the information into the
FILE section Channel table detail, then blanks out the Start
column.

---Input---
   CHARACTER*(*)       FMT     ! format string
                               ! parentheses optional
---Output---
   INTEGER             STATUS  ! 0 - no trouble
                               ! 1 - warning(s) printed
                               ! 2 - error(s) printed


See the documentation of routine HRC_FORMAT_TO_DETAIL for details.

The Start column information is redundant if the field Widths sum
to be the record length (i.e. if there are no extra spaces between
fields not included in the widths).
The start column information complicates reordering channels, and
mixing channels from multiple input files.  Thus for data processing
systems which always have no extra spaces between field widths,
use this routine (..._DETAIL2) instead of (..._DETAIL).

Note Feb 13, 2001: ..._DETAIL now only sets START if there are
                   X or T items in the input format string.

----------------------------------------------------------------------
```

10.6.5 HRC_PROGRAM_HISTORY

```
   SUBROUTINE HRC_PROGRAM_HISTORY(NAME,VERSION,TIME_STAMP,
  +                                RECS_IN,RECS_OUT)

Adds a row to the PROGRAMS table of the HISTORY section
of the IOS Header in common IOS_HEADER,
and turns on the History section output flag.

---Input---
   CHARACTER*(*)  NAME                 ! program name
   CHARACTER*(*)  VERSION              ! program version
   CHARACTER*(*)  TIME_STAMP           ! date & time in format:
                                       ! 'yyyy/mm/dd hh:mm:ss...'
   INTEGER        RECS_IN              ! record counters
   INTEGER        RECS_OUT

Routine HRC_SET_TIME_STAMP creates a time stamp in the required format.
(Only the first 19 bytes of the string are used.)

----------------------------------------------------------------------
```

10.6.6 HRC_PUT_ITEM_I

```
   SUBROUTINE HRC_PUT_ITEM_I(SECTION,LABEL,IIN, ITEMTYPE,STAT)

Subroutines and entries HRC_PUT_ITEM_<type> find the item identified
by SECTION and LABEL in the header structure, and inserts the data
from variable <type>IN.

Underscores instead of blanks are acceptable in multi-word labels.
E.g. "NUMBER OF RECORDS" is equivalent to "NUMBER_OF_RECORDS".

If LABEL is not a recognised standard item label, or an existing
custom item, a warning will be produced, and a custom item will be
created.

If <type> is not the same as the type of the item, conversion is
done if possible.

The type of the item found is returned in ITEMTYPE.

---Input---
   CHARACTER*(*)       SECTION     ! header section
                                   ! (1st 3 chars needed only)
   CHARACTER*(*)       LABEL       ! item label

   ! (the following statement is located here only so that the
   !  structure definitions are available for the output variables.)
   INCLUDE 'ios_header$inc:header_common.inc'

   ! one of the following IN variables, depending upon <type>
       INTEGER                     IIN
       INTEGER*2                   I2IN
       INTEGER*4                   I4IN
       REAL*4                      R4IN
       REAL*8                      R8IN
       RECORD/SR_FULLTIME/         FTIN
       RECORD/SR_TIME_INCREMENT/   TIIN
       RECORD/SR_LATLON/           LLIN
       CHARACTER*(*)               CIN
---Output---

   CHARACTER*2         ITEMTYPE    ! data type of item (C I I2 I4...)
   INTEGER             STAT        ! 0 - all OK
                                   ! 1 - all OK - type conversion done
                                   ! 2 - error: invalid section
                                   ! 3 - error: incompatible types
                                   ! 4 - error decoding from character

Conversions:
-----------
   - If the type of the item is not the same as the type of the entry
     called, a conversion is done if possible.
   - If no conversion is possible, STAT is returned 3, and the item
     is set to null-blank.
   - Default Fortran conversions are done between all I and R types
   - Conversions to and from CHARACTER are done using the HR_ENCODE and
     HR_SCAN routines.  Thus the same results as for writing and
     reading a header are obtained.
   - LL (Lat/Lon) can be converted to/from R4 and R8 - the real number
     is assumed to be degrees, with negative values used for South and
     West hemispheres.
   - TI can be converted to/from R4 and R8 - the real number is assumed
     to be in Days.

PUT_ITEM routine calling structure:

                         HRC_PUT_ITEM_<type>     \     both in
                               |                   >   HRC_PUT_ITEM_I
                         HR_PUT_ITEM_<type>      /               .FOR
                           /              \
                          /                \
                     HR_PUT_ITEM          HR_CONVERT
                     /        \                 \
             HR_LOOKUP_ITEM  HR_FETCH            \
               /       \         |             HR_ENCODE
              /         \    HR_COPY_BYTES     HR_SCAN
             /           \                     HR_SET_NULL
HR_LOOKUP_ITEM_INDEX   HR_LOOKUP_CUSTOM_ITEM   HR_TI_TO_R8
                                               HR_R8_TO_TI

----------------------------------------------------------------------
```

10.6.7 HRC_SET_CHANNEL_DETAIL

```
   SUBROUTINE HRC_SET_CHANNEL_DETAIL(CHN,PAD,START,WIDTH,FORMAT,
  +                                  TYPE,DEC)

Inserts channel detail information into the FILE section,
for channel number CHN.

---Input---
   INTEGER         CHN         ! channel number
   REAL*4          PAD         ! pad value for the channel
   INTEGER         START       ! ...col of this channel in data file
   INTEGER         WIDTH       ! ...of this channel in the data file
   CHARACTER*(*)   FORMAT      ! e.g. 'E14.7'
   CHARACTER*(*)   TYPE        ! e.g. 'REAL*4'
   INTEGER         DEC         ! # decimal places

Use HRC_CHANNEL_POS1 to get a channel number from a channel name.
Routine HRC_ADD_FILE_CHANNEL also provides a channel number.

If this routine is unsuccessful (i.e. if CHN is out of range)
a warning message is produced.

----------------------------------------------------------------------
```

10.6.8 HRC_SET_TIME_STAMP

```
   SUBROUTINE HRC_SET_TIME_STAMP(TIME_STAMP)

Creates a header time stamp using the system date and time, and inserts
it into the header in common IOS_HEADER (HRC) or in the provided header
(HR).
The same time stamp is returned in string TIME_STAMP (without a leading
asterisk)

---Output---
   CHARACTER*(*)    TIME_STAMP      ! yyyy/mm/dd hh:mm:ss.ss
                                    ! use CHARACTER*22 to get it all.

----------------------------------------------------------------------
```

10.7 Read Write

10.7.1 HR_SET_COLUMN_HEADINGS

```
   SUBROUTINE HR_SET_COLUMN_HEADINGS(WRITE_COLUMN_HEADINGS,
  +                                  MAX_LINE_LENGTH,
  +                                  NLINES,
  +                                  CUTSTR,
  +                                  WRITE_CHANNEL_NUMBERS)

Sets the parameters in common HR_COLHEAD which control the
writing of column headings as comments at the end of the IOS header.

---Input---
   LOGICAL*1   WRITE_COLUMN_HEADINGS
                            ! True:  write column headings
                            ! False: do NOT write column headings:
                            !        subsequent calls to
                            !        HR_WRITE_COLUMN_HEADINGS
                            !        will have no effect.

   INTEGER     MAX_LINE_LENGTH ! Maximum length of column heading records.
                            ! -1: do not change from current value

   INTEGER     NLINES       ! Maximum number of lines over which a
                            ! Channel name may be extended.
                            !  0: only the '--1-- --2-- ...' line will
                            !     be produced.
                            ! -1: do not change from current value

   CHARACTER*(*) CUTSTR     ! String to insert to indicate characters
                            ! have been cut out of the middle of a
                            ! channel name, e.g. '~' =>   Temp~ure1
                            !                    '...' => Tem...re1
                            ! - will be truncated if longer then 4.
                            ! - 'same' (lower case): no change from
                            !           current string.
                            ! - ' ' for no cut indicators.

   LOGICAL*1   WRITE_CHANNEL_NUMBERS
                            ! .True. to have the channel number
                            ! inserted above the names.

These parameters are used in routine HRC_WRITE_COLUMN_HEADINGS.
----------------------------------------------------------------------
```

10.7.2 HRC_READ_HEADER

```
   SUBROUTINE HRC_READ_HEADER(U,HSTAT)

Reads an IOS header from unit U into common IOS_HEADER.
(The HR_ version reads into the named header structure.)
The file must already be open, and positioned at the first record.

---Input---
   INTEGER                  U           ! unit number to read from

---Output---
   INTEGER                  HSTAT       ! 0 - all OK
```

```
                                            ! 1 - warnings
                                            ! 2 - errors - do not use.

Note:
- HSTAT is set in the error and warning routines.
- Item labels, table names and section names are made upper-case in
  routine HR_GET_HEADER_RECORD
- routine HRC_CHECK_HEADER can be used to check that all mandatory
  items are present in the header, after reading the header.
----------------------------------------------------------------------
```

10.7.3 HRC_WRITE_COLUMN_HEADINGS

```
   SUBROUTINE HRC_WRITE_COLUMN_HEADINGS(U,HSTAT)

Writes data column headings as header comment records, according
to the parameters in common HR_COLHEAD, if the COLHEAD_WRITE flag
in that common is .TRUE.

---Input---
   INTEGER     U           ! unit number to write to

---I/O-----
   INTEGER     HSTAT       ! header write status: 0:OK, 1:warn 2:error

Route HR_SET_COLUMN_HEADINGS may be used to set the parameters
in common HR_COLHEAD
----------------------------------------------------------------------
```

10.7.4 HRC_WRITE_HEADER

```
   SUBROUTINE HRC_WRITE_HEADER(U,HSTAT)

Writes an IOS header to unit U from common IOS_HEADER.
(The HR_ version writes from the named header structure.)
The file must already be open.

---Input---
   INTEGER                   U           ! unit number to write to

---Output---
   INTEGER                   HSTAT       ! status: 0: OK, 1: warnings
                                         !                2: errors

On the VAX, the output file should be opened with the
CARRIAGE_CONTROL='LIST' option.  (See IOSLIB routine OPENF).

----------------------------------------------------------------------
```

10.8 Remark Handling

10.8.1 HRC_ADD_REMARK

```
   SUBROUTINE HRC_ADD_REMARK(SECTION,NEWREM)

Adds (appends) remark NEWREM into section SECTION of the header.

---Parameters---
```

```
   CHARACTER*(*)   SECTION     ! section name (only 1st 3 chars used)
   CHARACTER*(*)   NEWREM      ! remark to append

----------------------------------------------------------------------
```

10.8.2 HRC_CLEAR_REMARKS

```
   SUBROUTINE HRC_CLEAR_REMARKS(SECTION)

Deletes all remarks in section SECTION of the header.
If SECTION is 'ALL', all remarks of the entire header will be deleted.

---Parameters---

   CHARACTER*(*)   SECTION     ! section name (only 1st 3 chars used)

Note: this routine just re-sets pointers, it does not actually blank
      out the remarks in memory.

See also: HRC_DELETE_REMARK to delete a single remark.

----------------------------------------------------------------------
```

10.8.3 HRC_DELETE_REMARK

```
   SUBROUTINE HRC_DELETE_REMARK(SECTION,POSITION)

Deletes remark at position POSITION in section SECTION of the header.

---Parameters---

   CHARACTER*(*)   SECTION     ! section name (only 1st 3 chars used)
   INTEGER         POSITION    ! remark position within the section

If there is no remark at position POSITION, an error message will be
produced, and nothing else will be done.

Remarks following the deleted one will be shifted down in the remark
array, and all following remark pointers will be reduced accordingly.

See also: HRC_CLEAR_REMARKS to delete all remarks of a section.
          HRC_REPLACE_REMARK to replace a remark.

----------------------------------------------------------------------
```

10.8.4 HRC_GET_REMARK

```
   SUBROUTINE HRC_GET_REMARK(SECTION,POSITION,REM)

Returns the POSITION'th remark of section SECTION in REM.

---Input---

   CHARACTER*(*)   SECTION     ! section name (only 1st 3 chars used)
   INTEGER         POSITION    ! the position within the section of
                               ! the remark.
---Output---
   CHARACTER*(*)   REM         ! remark returned.
                               ! will have leading blanks if indented
                               ! more than section minimum indentation
```

```
If POSITION is greater than the number of existing remarks for the
section, or if it is less than one, REM is returned HC_ABSENT
(defined as 'o' in IOS_HEADER include file NULVALS.INC).

Thus this routine can be called in a loop with POSITION increasing
until HC_ABSENT ('o') is returned.

-----------------------------------------------------------------------
```

10.8.5 HRC_INSERT_REMARK

```
   SUBROUTINE HRC_INSERT_REMARK(SECTION,POSITION,NEWREM)

Inserts remark NEWREM into section SECTION of the header in
common IOS_HEADER.

---Parameters---

   CHARACTER*(*)   SECTION     ! section name (only 1st 3 chars used)
   INTEGER         POSITION    ! the position within the section of the
                               ! new remark.
   CHARACTER*(*)   NEWREM      ! remark to insert.

Existing remarks in the section from POSITION up will be shifted up one
position.  If POSITION is greater than the number of remarks already in
the section, NEWREM will simply be appended to those already there.
If POSITION is less than 1, 1 will be used.

See also: HRC_ADD_REMARK to simply append a remark to a section
          HRC_REPLACE_REMARK to replace a remark.

-----------------------------------------------------------------------
```

10.8.6 HRC_N_REMARKS

```
   INTEGER FUNCTION HRC_N_REMARKS(SECTION)

Returns the number of remarks in section SECTION.
If SECTION is 'ALL', total number of remarks for the header is
returned.

---Parameters---

   CHARACTER*(*)   SECTION     ! section name (only 1st 3 chars used)

If SECTION is invalid, a value of 0 will be returned, and an error
message will be produced.
-----------------------------------------------------------------------
```

10.8.7 HRC_REPLACE_REMARK

```
   SUBROUTINE HRC_REPLACE_REMARK(SECTION,POSITION,NEWREM)

Replaces the POSITION'th remark in section SECTION with NEWREM.

---Parameters---

   CHARACTER*(*)   SECTION     ! section name (only 1st 3 chars used)
   INTEGER         POSITION    ! the position within the section of the
                               ! new remark.
   CHARACTER*(*)   NEWREM      ! new remark
```

```
If POSITION is greater than the number of existing remarks for the
section, or if it is less than one, an error message is produced,
and nothing is done.

See also: HRC_ADD_REMARK to simply append a remark to a section,
          HRC_INSERT_REMARK to insert a remark.

----------------------------------------------------------------------
```

10.9 Time Series

10.9.1 HR_SET_FIXED_ZERO_CONDAY

```
   SUBROUTINE HR_SET_FIXED_ZERO_CONDAY

Sets the IOSLIB zero conday to the IOS Header standard reference date.

To obtain the IOS Header reference date, call this routine, then call
IOSLIB routine GET_ZERO_CONDAY.
----------------------------------------------------------------------
```

10.9.2 HR_TS_FT_TO_R8

```
   SUBROUTINE HR_TS_FT_TO_R8(FT, R8)

Converts a Full Time to a REAL*8 number of days from the
header time zero, as determined by the last call to HR_TSZ_PREP.
If the FT time zone differs from that of the header time_zero,
the result is converted to the header time_zero time zone.

   INCLUDE 'ios_header$inc:header_basics.inc'
---Input---
   RECORD /SR_FULLTIME/    FT      ! a full time
---Output---
   REAL*8                  R8      ! days since header time zero.
                                   ! returned HN_UNKNOWN if FT is invalid

See also: HR_FT_TO_Z8, HRC_FT_TO_R8
----------------------------------------------------------------------
```

10.9.3 HR_TS_GET_FT

```
   SUBROUTINE HR_TS_GET_FT(rn,data,zone, ft)

Returns the time of the last record read (with HR_READ_DATA_R4)
as a full time in ft.  If necessary, the time will be
converted to the specified time zone.

---Input---

   INTEGER      rn           ! record number (returned from HR_READ_DATA_R4)
                             ! (only used if record times are calculated)
   REAL*4       data(*)      ! data returned from HR_READ_DATA_R4.
                             ! (not used for SPECIAL time channels)
   CHARACTER*(*) zone        ! desired time zone for FT (blank = UTC)
                             ! - 'hdr' to use ts_hdr_zone from hr_tsz.inc

---Output---
```

```
   INCLUDE 'ios_header$inc:header_basics.inc'
   RECORD/SR_FULLTIME/ ft  ! the full time of the last record read.
                           ! - will be null/absent if not available/

See notes in routine HR_TS_GET_Z8 on required use of HR_TSZ_PREP.

See also: HR_TS_GET_Z8, HRC_TSZ_PREP
-----------------------------------------------------------------------
```

10.9.4 HR_TS_GET_R8

```
    SUBROUTINE HR_TS_GET_R8(rn,data, r8)

Returns the time of the last record read (with HR_READ_DATA_R4)
as a REAL*8 offset from the HEADER time zero.

---Input---

   INTEGER     rn          ! record number (returned from HR_READ_DATA_R4)
                           ! (only used if record times are calculated)
   REAL*4      data(*)     ! data returned from HR_READ_DATA_R4.
                           ! (not used for SPECIAL time channels)
---Output---

   REAL*8      r8          ! time of the last record read, as an offset
                           ! from the fixed time zero (FTZ), UTC.

Routine HR_TSZ_PREP must be called after the header for the data
file was read, and before any calls to this routine.  If multiple
files are being read simultaneously, call HR_TSZ_PREP with the header of
the last file read, before calling this routine.

See also: HRC_TSZ_PREP, HR_TS_GET_FT, HR_TS_GET_Z8
-----------------------------------------------------------------------
```

10.9.5 HR_TS_GET_Z8

```
    SUBROUTINE HR_TS_GET_Z8(rn,data, z8)

Returns the time of the last record read (with HR_READ_DATA_R4)
as a REAL*8 offset from the fixed time zero, UTC.

---Input---

   INTEGER     rn          ! record number (returned from HR_READ_DATA_R4)
                           ! (only used if record times are calculated)
   REAL*4      data(*)     ! data returned from HR_READ_DATA_R4.
                           ! (not used for SPECIAL time channels)
---Output---

   REAL*8      z8          ! time of the last record read, as an offset
                           ! from the fixed time zero (FTZ), UTC.

Routine HR_TSZ_PREP must be called after the header for the data
file was read, and before any calls to this routine.  If multiple
files are being read simultaneously, call HR_TSZ_PREP with the header of
the last file read, before calling this routine.

See also: HRC_TSZ_PREP, HR_TS_GET_FT, HR_TS8_GET_Z8
-----------------------------------------------------------------------
```

10.9.6 HR_TS_PUT_R8

```
   SUBROUTINE HR_TS_PUT_R8(r8, data,rn)

This routine performs the same function as HR_TS_PUT_Z8, except that
it receives the time, r8, relative to the HEADER time zero, instead
of the fixed time zero.  See HR_TS_PUT_Z8 for details.

---Input---

   REAL*8      r8          ! time of the record about to be written,
                           ! as an offset from the header time zero.
                           ! (If the header has no time zero, the
                           ! fixed time zero is used.)
---Output---
   REAL*4      data(*)     ! data ready to send to  HR_WRITE_DATA_R4.
   INTEGER     rn          ! record number
                           ! (only set if record times are calculated)


----------------------------------------------------------------------
```

10.9.7 HR_TS_PUT_Z8

```
   SUBROUTINE HR_TS_PUT_Z8(z8, data,rn)

Receives z8, the time of a record about to be written (with routine
HR_WRITE_DATA_R4).   The time is put into the correct channel(s) in
'data', or into the time structure in common ftlatlon, (as specified
in the header sent to the last call to HR_TSZ_PREP), so that in the
next call to HRC_WRITE_DATA_Rn, the time is written correctly.
If the record times are calculated (i.e. not in each record), then
then 'data' is not changed, but the record number corresponding to
time z8 is returned in 'rn'.

z8 must be a REAL*8 offset from the fixed time zero.
The data will be converted to the header time zone, if the header
time zone is not UTC or null.

---Input---

   REAL*8      z8          ! time of the record about to be written,
                           ! as an offset from fixed time zero, UTC.
---Output---
   REAL*4      data(*)     ! data ready to send to  HR_WRITE_DATA_R4.
   INTEGER     rn          ! record number
                           ! (only set if record times are calculated)


Routine HR_TSZ_PREP must be called after the header for the data
file is prepared, and before any calls to this routine.  If multiple
files are being written simultaneously, call HR_TSZ_PREP with the header
of the next file to write before calling this routine.

If used in conjunction with HR_TS_GET_Z8, and the output header has
all the same time information as the input header, then HR_TSZ_PREP
only has to be called once - before input.  If any of the time
characteristics are different in the output header, HR_TSZ_PREP should
be re-called before output.
```

```
See also: HR_TS_GET_FT, HRC_TSZ_PREP
-----------------------------------------------------------------------
```

10.9.8 HR_TS_R8_TO_FT

```
   SUBROUTINE HR_TS_R8_TO_FT(R8, FT)

Converts REAL*8 R8 to a Full Time in FT, using the header time_zero
determined in the last call to HR_TSZ_PREP.
R8 is the number of days since that TIME ZERO.

---Input---
   REAL*8                 R8      ! days since header time zero.

---Output---
   INCLUDE 'ios_header$inc:header_basics.inc'
   RECORD /SR_FULLTIME/    FT      ! a full time
                                   ! zone will be the header zone

See also: HRC_FT_TO_R8, HR_TS_FT_TO_R8, HR_Z8_TO_FT

Note: Who'd have thought, when a mammal first raised itself up onto
      two legs, that this is what it would lead to?
-----------------------------------------------------------------------
```

10.9.9 HR_TS8_GET_FT

```
   SUBROUTINE HR_TS8_GET_FT(rn,data, zone, ft)

Returns the time of the last record read (with HR_READ_DATA_R8)
as a full time in ft.  The time will be converted to the specified
time zone if necessary.

---Input---

   INTEGER     rn          ! record number (returned from HR_READ_DATA_R8)
                           ! (only used if record times are calculated)
   REAL*8      data(*)     ! data returned from HR_READ_DATA_R8.
                           ! (not used for SPECIAL time channels)
   CHARACTER*(*) zone      ! desired time zone for ft (blank = UTC)
---Output---

   INCLUDE 'ios_header$inc:header_basics.inc'
   RECORD/SR_FULLTIME/ ft  ! the full time of the last record read.
                           !      - will be null/absent if not available/

Routine HR_TSZ_PREP must be called after the header for the data
file was read, and before any calls to this routine.  If multiple
files are being read simultaneously, call HR_TSZ_PREP with the header
of the file being read every time that file is switched to.

See also: HR_TS_GET_Z8, HR_TS_GET_FT,  HRC_TSZ_PREP
-----------------------------------------------------------------------
```

10.9.10 HR_TS8_GET_R8

```
    SUBROUTINE HR_TS8_GET_R8(rn,data, r8)

Returns the time of the last record read (with HR_READ_DATA_R8)
as a REAL*8 offset from the HEADER time zero.
```

```
---Input---

   INTEGER     rn          ! record number (returned from HR_READ_DATA_R4)
                           ! (only used if record times are calculated)
   REAL*8      data(*)     ! data returned from HR_READ_DATA_R8.
                           ! (not used for SPECIAL time channels)
---Output---

   REAL*8      r8          ! time of the last record read, as an offset
                           ! from the fixed time zero (FTZ), UTC.

Routine HR_TSZ_PREP must be called after the header for the data
file was read, and before any calls to this routine.  If multiple
files are being read simultaneously, call HR_TSZ_PREP with the header of
the last file read, before calling this routine.

See also: HRC_TSZ_PREP, HR_TS8_GET_FT, HR_TS8_GET_Z8
-----------------------------------------------------------------------
```

10.9.11 HR_TS8_GET_Z8

```
   SUBROUTINE HR_TS8_GET_Z8(rn,data, z8)

Returns the time of the last record read (with HR_READ_DATA_R8)
as a REAL*8 UTC offset from the IOS Header library fixed time zero.

---Input---

   INTEGER     rn          ! record number (returned from HR_READ_DATA_R8)
                           ! (only used if record times are calculated)
   REAL*8      data(*)     ! data returned from HR_READ_DATA_R8.

---Output---

   REAL*8      z8          ! UTC time of the last record read, in days
                           ! from the fixed time zero.

Routine HR_TSZ_PREP must be called after the header for the data
file was read, and before any calls to this routine.  If multiple
files are being read simultaneously, call HR_TSZ_PREP with the header of
the last file read, before calling this routine.

See also: HR_TS_GET_Z8, HR_TS8_PUT_FT, HRC_TSZ_PREP
-----------------------------------------------------------------------
```

10.9.12 HR_TS8_PUT_R8

```
   SUBROUTINE HR_TS8_PUT_R8(r8, data,rn)

This routine performs the same function as HR_TS8_PUT_Z8, except that
it receives the time, r8, relative to the HEADER time zero, instead
of the fixed time zero.  See HR_TS8_PUT_Z8 for details.

---Input---

   REAL*8      r8          ! time of the record about to be written,
                           ! as an offset from the header time zero.
                           ! (If the header has no time zero, the
                           ! fixed time zero is used.)
---Output---
```

```
   REAL*8      data(*)     ! data ready to send to  HR_WRITE_DATA_R4.
   INTEGER     rn          ! record number
                           ! (only set if record times are calculated)


----------------------------------------------------------------------
```

10.9.13 HR_TS8_PUT_Z8

```
   SUBROUTINE HR_TS8_PUT_Z8(z8, data,rn)

Receives z8, the time of a record about to be written (with routine
HR_WRITE_DATA_R8).   The time is put into the correct channel(s) in
'data', according to the ts_hdr_timetype stored in common hr_tsz.

Thus in the next call to HRC_WRITE_DATA_R8, the time is written correctly.
If the record times are calculated (i.e. not in each record), then
then 'data' is not changed, but the record number corresponding to
time r8 is returned in 'rn'.

z8 must be a REAL*8 offset from the fixed time, UTC.
Conversion to the header time zone (as stored in common hr_tsz)
wil be done if requried.

---Input---

   REAL*8      z8          ! time of the record about to be written,
                           ! as an offset from the fixed time_zero, UTC

---Output---
   REAL*8      data(*)     ! data ready to send to  HR_WRITE_DATA_R8.
   INTEGER     rn          ! record number
                           ! (only set if record times are calculated)


Routine HR_TSZ_PREP must be called after the header for the data
file is prepared, and before any calls to this routine.  If multiple
files are being written simultaneously, call HR_TSZ_PREP with the header
of the next file to write before calling this routine.

If used in conjunction with HRC_TS_GET_Z8, and the output header has
all the same time information as the input header, then HR_TS_PREP
only has to be called once - before input.  If any of the time
characteristics are different in the output header, HR_TS_PREP should
be re-called before output.

Note:  Prior to 2003-03-14, if there were separate special date/time
  channels, HN_ABSENT was put into DATA, and the date/time in r8
  was put into the the time structure in common ftlatlon.
  This caused trouble if there were more than one date/time channel pairs!
  So, since in Real*8 there is no loss of accuracy, it was changed to
  put the correct values into data:

Special channel values:  (values inserted in the Header time zone.)
  Separate date/time channels:
    data(ts_timepos1) : number of days from the header time zero day.
    data(ts_timepos2) : absolute time of day, in days.
  One combined channel:
    data(ts_timepos1) : days from header time zero.


See also: HR_TS_PUT_Z8, HR_TS8_GET_FT, HRC_TSZ_PREP
```

```
----------------------------------------------------------------------
```

10.9.14 HR_TSZ_DUMP

```
   SUBROUTINE HR_TSZ_DUMP(u)

Write the contents of common hr_tsz to unit u

   INTEGER u       ! unit number

This is largely a debugging tool.

----------------------------------------------------------------------
```

10.9.15 HRC_TSZ_PREP

```
   SUBROUTINE HRC_TSZ_PREP (option, datepos, timepos, tstatus)


Set up for obtaining the time for each input data record, using the
HR_TS* routines, and using the ios header fixed time zero.

---Input---
   INTEGER             option  ! 0 - use HR_FINDTIME to determine where
                               !     record times are to come from.
                               ! 1 - use datepos and timepos channels
                               ! 2 - use start time and time increment

                               !-- Following used only if option = 1 --
   INTEGER             datepos ! date (and time) channel number
   INTEGER             timepos ! time channel number, if datepos contains
                               ! only the date.
                               ! May be > 0 only if datepos > 0

---Output---
   INTEGER             tstatus ! 0 - no recognised time definition in
                               !     the header.  (Failure)
                               ! 1 and up: timetype: see HRC_FINDTIME

This routine must be called after the header is read, and before any
HR_TS_* routines are called.
----------------------------------------------------------------------
```

10.10 Utility

10.10.1 HR_CHANGE_TIME_ZONE

```
   SUBROUTINE HR_CHANGE_TIME_ZONE(FT1,ZONE,FT2)

Change full time FT1 to time zone ZONE, returning the result in FT2.

   INCLUDE 'ios_header$inc:sr_fulltime.inc'    ! structure definitions

---Input---
   RECORD /SR_FULLTIME/    FT1
   CHARACTER*3             ZONE

---Output---
   RECORD /SR_FULLTIME/    FT2
```

```
FT2 may not be the same (address) as FT1
----------------------------------------------------------------------
```

10.10.2 HR_CHANNEL_MATCH

```
   LOGICAL*1 FUNCTION HR_CHANNEL_MATCH(C1, C2)

Checks to see if string C1 matches string C2.
C1 may start and/or end with a wild card (*).  C2 may not.
C1 may start (and optionally end) with a square bracket.  C2 may not.


---Input---

   CHARACTER*(*)   C1          ! channel name or template.  Begin and/
                               ! or end with * for wild card searching.
                               ! Start with [ for case-sensitive match.
   CHARACTER*(*)   C2          ! complete channel name to match to.

By default, the matching is case-insensitive.
Case-sensitive matching is used if C1 starts with a left square bracket.
In this case the left bracket (and a right bracket, if present) is
removed before case-sensitive matching is done.

This routine (as indicated by the name) was designed for matching
channel names, but may be used for comparing any two strings up to
the header-defined channel name length.  The case-insensitive option
was added in April 2002 to handle units comparisons.

 Examples:
   HR_CHANNEL_MATCH('Temperature','TEMPERATURE')        TRUE
   HR_CHANNEL_MATCH('Temp*'      ,'TEMPERATURE')        TRUE
   HR_CHANNEL_MATCH('Temp'       ,'TEMPERATURE')        FALSE
   HR_CHANNEL_MATCH('*:water','Temperature:Water')      TRUE
   HR_CHANNEL_MATCH('*:wat'  ,'Temperature:Water')      FALSE
   HR_CHANNEL_MATCH('*:wat*' ,'Temperature:Water')      TRUE
   HR_CHANNEL_MATCH('*PERAT*','Temperature:Water')      TRUE
   HR_CHANNEL_MATCH('*'      ,'Temperature:Water')      TRUE

   HR_CHANNEL_MATCH('metres'  ,'Metres')                TRUE
   HR_CHANNEL_MATCH('[metres' ,'Metres')                FALSE
   HR_CHANNEL_MATCH('[metres]','Metres')                FALSE
   HR_CHANNEL_MATCH('[Metres' ,'Metres')                TRUE
   HR_CHANNEL_MATCH('[Metres]','Metres')                TRUE
   HR_CHANNEL_MATCH('[*]'     ,'Metres')                TRUE
   HR_CHANNEL_MATCH('[M*]'    ,'Metres')                TRUE

----------------------------------------------------------------------
```

10.10.3 HR_CHANNEL_NAME_FROM_ID

```
   CHARACTER*(*) FUNCTION HR_CHANNEL_NAME_FROM_ID(CHAN_ID)

If CHAN_ID contains no left square bracket, CHAN_ID is returned.
If CHAN_ID contains a left square bracket, the string returned
is CHANID with the left square bracket and all following characters
removed.

---Input---
```

```
   CHARACTER*(*)   CHAN_ID

----------------------------------------------------------------------
```

10.10.4 HR_CHECK_SIZE

```
   SUBROUTINE HR_CHECK_SIZE(HIN,LAST_BYTE)

Checks that the header structure in the calling routine is the same
length as in the Header library.  This will catch any programs which
have not been recompiled and linked after a header structure
definition change.

---Parameters---
   LOGICAL*1       HIN                  ! user's header structure
   INTEGER         LAST_BYTE            ! user's H.LAST_BYTE

----------------------------------------------------------------------
```

10.10.5 HR_COPY_TABLES

```
   SUBROUTINE HR_COPY_TABLES(SECTION,H1,H2)

All non-empty tables in section SECTION of header H1 are copied into
header H2, completely replacing those tables in H2.

---Parameters---
   CHARACTER*3         SECTION     ! current section ID

   INCLUDE 'ios_header$inc:sr_header.inc'
   RECORD /SR_HEADER/  H1          ! FROM and TO header structures
   RECORD /SR_HEADER/  H2

If a table is NOT in H1, that table in H2 is left un-changed.

See also routine HR_EDIT_HEADER.
----------------------------------------------------------------------
```

10.10.6 HR_DATE_STRUCTURE

```
   SUBROUTINE HR_DATE_STRUCTURE(GETPUT, YEAR,MONTH,DAY, DATE)

Copies date information to or from a date structure, depending upon
whether GETPUT is 'GET' or 'PUT'.

---Input---
   CHARACTER*3   GETPUT    ! instruction: 'GET' from structure, or'
                           !              'PUT' into structure
                           ! (not case sensitive)
---Input or Output---

   INTEGER       YEAR      ! output: 4 digit e.g. 1993
                           ! input : 2 or 4 digit.  If 2, will be
                           !         converted to 4, in range 1941-2040
   INTEGER       MONTH
   INTEGER       DAY

   INCLUDE 'ios_header$inc:sr_date.inc'
   RECORD        /SR_DATE/ DATE         ! a date structure

Note: This routine does NOT put data in, or copy data out of a header,
      unless the DATE specified is in a header structure.

See also: HR_TIME_STRUCTURE, HR_FT_STRUCTURE
----------------------------------------------------------------------
```

10.10.7 HR_DELIM_SCAN

```
   SUBROUTINE HR_DELIM_SCAN(STR, DELIMITED, DELIMITER, Q)

Scans STR for delimited format information.

---Input---
   CHARACTER*(*)   STR         ! input string, case independent

---Output---
   LOGICAL*1       DELIMITED   ! true if 1st word in STR is DELIMITED
   CHARACTER*1     DELIMITER   ! delimiter
   CHARACTER*1     Q           ! quote character (for writing)


If the first word in STR is 'DELIMITED', then:
  - DELIMITED is returned TRUE (else FALSE)
  - delimiter is set according to the second word in STR:
     - TAB or COMMA         - tab or comma is set.
     - a single character   - that character is set
     - no second word:      - comma is set
  - the quote character is set according to the third word in STR:
     - no third word        - double quote <">
     - a single character   - that character is set.
     - SINGLEQUOTE          - single quote is set
     - DOUBLEQUOTE          - double quote is set

----------------------------------------------------------------------
```

10.10.8 HR_EDIT_HEADER

```
   SUBROUTINE HR_EDIT_HEADER(E,H)

This routine adds the information of header E (the 'Edit' header)
to header H (the 'Target' header).

---Input---
   INCLUDE 'ios_header$inc:sr_header.inc'
   RECORD /SR_HEADER/ E        ! edit header

---Input/Output---
   RECORD /SR_HEADER/ H        ! target header (header to edit)

If the Target header contains items, tables or arrays which are also in
the Edit header, those items and tables will be REPLACED with the
information from the Edit header.

Custom items can be removed, replaced or added in the same manner as
standard items.

To remove an item, add a custom item called "%REMOVE" in the ADMIN
section of the edit header, and give it an otherwise unused null value,
e.g. 'N/A'.  Then put that null value on all items which are to be
removed.  E.g. to remove NUMBER OF RECORDS:
    *ADMINISTRATION
          %REMOVE : N/A
    *FILE
          NUMBER OF RECORDS : N/A
Note: a non-null 'remove' value can be used if only one header type
of item is to be removed: e.g. you could use '-9' for numeric items,
or 'REMOVE' for character items.  Null values will work for any type.

If the Edit header contains remarks, then if the first remark in a
section is:
    APPEND  - the remarks will be appended to those in the Target header
    PREPEND - the remarks will go before all those in the target header
    REPLACE - the remarks will replace those in the Target header
    AFTER * - the remarks will be inserted - see notes below.
    BEFORE * - the remarks will be inserted - see notes below
    else    - the remarks will be appended to those in the Target header

The APPEND, PREPEND and REPLACE keyword may be anywhere in the first remark l
of the section, but must be upper case, and must be the only characters
on the line.

The AFTER and BEFORE keywords must be the first word on the line, and
must be followed by:
    - at least one blank,
    - up to 4 strings, delimited with '|'.  The strings must match
      exactly strings somewhere in consecutive remarks in the target header
      to indicate the insertion point.

    Insert Examples:
     1.    BEFORE More remarks to precede.
           New remark to insert.

        If a remark in the target header and section contain the string
        'More remarks to precede', then the new remark will be inserted
        before that remark.


     2.    AFTER Analysis methods|----------------| |
           Shoe size was analysed in comparison to thumb length,
           using methods explored by Dr. Small Bodybits.

        The 2 new remarks will be inserted after 3 consecutive remarks
        containing 'Analysis methods', '--------------', and a blank record.

     If the string(s) are not found, the remarks will be appended.
     The string matching is case sensitive.

Note: September 22, 2004; work was started on a version of this routine
      which optionally logs actual changes made.
      See save\hr_edit_header2.for
----------------------------------------------------------------------
```

10.10.9 HR_FT_STRUCTURE

```
   SUBROUTINE HR_FT_STRUCTURE(GETPUT, ZONE,YEAR,MONTH,DAY,HOUR,MINUTE,SECOND,

Copies information to or from a full-time structure, depending upon
whether GETPUT is 'PUT' or 'GET'.

---Input---
   CHARACTER*3   GETPUT    ! instruction: 'GET' from structure, or
                           !              'PUT' into structure
                           ! (not case sensitive)
---Input or Output---
```

```
   CHARACTER*3   ZONE
   INTEGER       YEAR
   INTEGER       MONTH
   INTEGER       DAY
   INTEGER       HOUR
   INTEGER       MINUTE
   REAL          SECOND

   INCLUDE  'ios_header$inc:sr_fulltime.inc'
   RECORD        /SR_FULLTIME/ FT            ! a full-time structure

Note: This routine does NOT put data in, or copy data out of a header,
      unless the FT specified is in a header structure.

      SECOND is type REAL, as in the structure,
      not INTEGER as in routine HR_TIME_STRUCTURE.

See also: HR_DATE_STRUCTURE, HR_TIME_STRUCTURE
----------------------------------------------------------------------
```

10.10.10 HR_FT_TO_Z8

```
   SUBROUTINE HR_FT_TO_Z8(FT, Z8)

Converts a Full Time to a REAL*8 number of days from the
IOS Header library fixed time Zero, UTC.

   INCLUDE 'ios_header$inc:sr_fulltime.inc'
---Input---
   RECORD /SR_FULLTIME/    FT      ! a full time
---Output---
   REAL*8                  Z8      ! days since fixed time zero, UTC

If FT.ZONE is null (e.g. blank), no time zone conversion is done.
   i.e. it is assumed that the full time is already UTC.
If FT.DATE.YEAR is a header null value, Z8 will be set to that value.
If FT.TIME.HOUR is a header null value, Z8 will be set to that value.

See also: HR_Z8_TO_FT
----------------------------------------------------------------------
```

10.10.11 HR_INTEGER_TI

```
   SUBROUTINE HR_INTEGER_TI(TI,ITI)

Converts time interval TI into all-integers.  ITI may either
be a structure or an array of 5 integers.

---Input---
   INCLUDE 'ios_header$inc:header_basics.inc'
   RECORD /SR_TIME_INCREMENT/  TI

---Output---
   INCLUDE 'ios_header$inc:sr_integer_ti.inc'
   RECORD /SR_INTEGER_TI/ ITI

For example  0. 1.5 0. 1. 0.  ==>  0 1 30 1 0

----------------------------------------------------------------------
```

10.10.12 HR_IS_FORTRAN_FORMAT

```
   LOGICAL*1 FUNCTION HR_IS_FORTRAN_FORMAT(FMT)

Returns TRUE if FMT is not null, FREE or DELIMITED.
I.e. if it can be assumed to be a Fortran format.

---Input---
   CHARACTER*(*)     FMT       ! header format item

----------------------------------------------------------------------
```

10.10.13 HR_ISNULL

```
   LOGICAL*1 FUNCTION HR_ISNULL(TYPE,D)

Returns TRUE if variable D of type TYPE contains one of the IOS header
null values.  Handles all ios header types except C (character).

---Input---
   CHARACTER*(*)     TYPE      ! type of variable D.  One of:
                               ! I,I2,I4,R4,R8,FT,TI,LL    (upper case)
                               ! NOTE: type Character not allowed.

   INCLUDE 'ios_header$inc:header_basics.inc'
   INCLUDE 'ios_header$inc:alltypes.inc'
   RECORD /ALLTYPES/ D         ! a variable of any header type except C


Character types cannot be handled because they are passed by descriptor
instead of by address.  Use HR_ISNULL_C

See also HR_ISNULL_<type>, where <type> may be any header type
including C (Character).

----------------------------------------------------------------------
```

10.10.14 HR_ISNULL_I

```
   LOGICAL*1 FUNCTION HR_ISNULL_I(I)

Functions HR_ISNULL_<type> determine if the supplied variable
of type <type> contains one of the four header null values.

   ! (the following statement is located here only so that the
   !  structure definitions are available for the input variables.)
   INCLUDE 'ios_header$inc:header_basics.inc'

---Input---
   ! one of the following, depending upon <type>
       INTEGER                    I
       INTEGER*2                  I2
       INTEGER*4                  I4
       REAL*4                     R4
       REAL*8                     R8
       RECORD/SR_FULLTIME/        FT
       RECORD/SR_TIME_INCREMENT/  TI
       RECORD/SR_LATLON/          LL
       CHARACTER*(*)              C

----------------------------------------------------------------------
```

10.10.15 HR_LATLON_STRUCTURE

```
   SUBROUTINE HR_LATLON_STRUCTURE(GETPUT, DEGREES,MINUTES,HEMISPHERE,
  +                                       LATLON)

Copies Latitude or Longitude information to or from a LatLon structure,
depending upon whether GETPUT is 'GET' or 'PUT'.

---Input---
   CHARACTER*3   GETPUT        ! instruction: 'GET' from structure, or'
                               !              'PUT' into structure
                               ! (not case sensitive)
---Input or Output---
   INTEGER       DEGREES
   REAL*4        MINUTES
   CHARACTER*1   HEMISPHERE

   INCLUDE  'ios_header$inc:sr_latlon.inc'
   RECORD        /SR_LATLON/ LATLON        ! a latlon structure

Note: This routine does NOT put data in, or copy data out of a header,
      unless the LATLON specified is in a header structure.

----------------------------------------------------------------------
```

10.10.16 HR_LOOKUP_CUSTOM_ITEM

```
   SUBROUTINE HR_LOOKUP_CUSTOM_ITEM(CUSTOM,SECTION,LABEL, P)

Searches for a custom item in the CUSTOM table of a header.

   INCLUDE 'ios_header$inc:string_lengths.inc'
   INCLUDE 'ios_header$inc:array_sizes.inc'
   INCLUDE 'ios_header$inc:sr_custom.inc'

---Input---
   RECORD /SR_CUSTOM/  CUSTOM          ! custom table of a header

   CHARACTER*(*)       SECTION         ! section name
                                       ! (only 1st 3 characters used)
   CHARACTER*(*)       LABEL           ! item label

---Output---

   INTEGER             P               ! pointer to the item in the
                                       ! custom table.  0 if not found.

For example, to extract custom item ACCOUNT NUMBER from the ADMIN
section, into character variable ACNUM:
    CALL HR_LOOKUP_CUSTOM_ITEM(H.CUSTOM,'ADM','ACCOUNT NUMBER',P)
    IF (P.GT.0) THEN
        ACNUM = H.CUSTOM.ITEM(P).INFO
    ELSE
        WRITE(6,*)'ACCOUNT NUMBER not found'
    ENDIF
----------------------------------------------------------------------
```

10.10.17 HR_MAKE_CHANNEL_ID

```
   CHARACTER*(*) FUNCTION HR_MAKE_CHANNEL_ID(NAME,UNITS, OPT)

Returns a channel ID "name [units]" for the given NAME and UNITS.
```

```
---Input---

   CHARACTER*(*)   NAME        ! channel name
   CHARACTER*(*)   UNITS       ! channel units
   CHARACTER*(*)   OPT         ! options, defined below, case sensitive

Options:  (option character may occur anywhere in OPT.)

  - N    : - units are omitted if null, so if units is null,
             you'll get "name" instead of "name []" or "name [?]"

Note to programmers: when adding option characters, make sure they
do not conflict with options in HRC_CHANNEL_ID, so that HRC_CHANNEL_ID
can pass its OPT argument unmodified to this function.

See also function HRC_CHANNEL_ID, HR_MOD_CHANNEL_ID

----------------------------------------------------------------------
```

10.10.18 HR_MOD_CHANNEL_ID

```
   CHARACTER*(*) FUNCTION HR_MOD_CHANNEL_ID(CHAN_ID, OPT)

Modifies a channel ID, depending upon OPT.

---Input---

   CHARACTER*(*)   CHAN_ID     ! channel name
   CHARACTER*(*)   OPT         ! options, defined below, case sensitive

Options:  * Case Sensitive *

  - N    : - units are removed if null, so if units is null,
             you'll get "name" instead of "name []" or "name [?]"
  - name : - name only is returned. (units, if present, are removed)
  - units: - units only is returned. (string inside [ ])

If OPT is not recognized, CHAN_ID is returned unchanged.

See also function HRC_CHANNEL_ID, HR_MAKE_CHANNEL_ID

----------------------------------------------------------------------
```

10.10.19 HR_MULTIPLE_TI

```
   SUBROUTINE HR_MULTIPLE_TI(FT1,TI,N,FT2)

Adds N times time increment TI to FT1, yielding FT2.

---Input---
   INCLUDE 'ios_header$inc:header_basics.inc'
   RECORD /SR_FULLTIME/        FT1
   RECORD /SR_TIME_INCREMENT/  TI
   INTEGER                     N

---Output---
   RECORD /SR_FULLTIME/        FT2


----------------------------------------------------------------------
```

10.10.20 HR_NORMALIZE_FT

```
   SUBROUTINE HR_NORMALIZE_FT(FT)

Given a Full-Time structure, the seconds are rounded to three
decimal places, then the entire full time is normalized so that
all fields are within their proper ranges.

On input the date must be a valid date, but the time fields may
be any number, positive or negative.  If the numbers in the time
add up to more than a day, the date will be correctly adjusted.

---Parameter---
   INCLUDE 'ios_header$inc:sr_fulltime.inc'
   RECORD /SR_FULLTIME/ FT     ! full time structure

Example:   In                         Out
      1994/11/30 23:59:59.99999    1994/12/01 00:00:00.000

This routine is called by routine HR_ENCODE_FT so that full-times
written to output headers are always normalized.

See also: HR_NORMALIZE_FTS, HR_NORMALIZE_FTF

Uses IOSLIB routine NORMALIZE_YMDHMS.
----------------------------------------------------------------------
```

10.10.21 HR_NORMALIZE_FTF

```
   SUBROUTINE HR_NORMALIZE_FTF(FT,TIMEFMT)

Given a Full-Time structure, and a time format string, the
time is rounded to the accuracy indicated in the format string,
then the entire full time is normalized so that all fields are
within their proper ranges.

---Parameter---
   INCLUDE 'ios_header$inc:sr_fulltime.inc'
   RECORD /SR_FULLTIME/ FT        ! full time structure

   CHARACTER*(*)        TIMEFMT   ! e.g. HH:MM
                                  ! if blank, round to nearest day
                                  ! not case dependent


For hours, minutes and integral seconds, it is simply the existance
of an H, M or S in the format which indicates the rounding level:
i.e.  given the format HH:M, the time is rounded to the nearest
minute, not the nearest ten minutes.  Likewise HH:MM:S would cause
rounding to the nearest second, not 10 seconds.

If the format contains an S and a decimal point, the number of
characters to the right of the decimal point determines the
number of decimal places the seconds are rounded to.

See also: HR_NORMALIZE_FT, HR_NORMALIZE_FTS

----------------------------------------------------------------------
```

10.10.22 HR_NORMALIZE_FTS

```
   SUBROUTINE HR_NORMALIZE_FTS(FT,NDEC)

Given a Full-Time structure, the time is rounded as specified with NDEC,
and the entire full time is normalized so that all fields are within
their proper ranges.

On input the date must be a valid date, but the time fields may
be any number, positive or negative.  If the numbers in the time
add up to more than a day, the date will be correctly adjusted.

---Parameter---
   INCLUDE 'ios_header$inc:sr_fulltime.inc'
   RECORD /SR_FULLTIME/ FT     ! full time structure

   INTEGER     NDEC            ! number of decimal places of seconds, or:
                               !  0 - nearest second
                               ! -1 - nearest 10 seconds
                               ! -2 - nearest minute
                               ! -3 - nearest 10 minutes
                               ! -4 - nearest hour.

Example:   In                          Out
   1994/11/30 23:59:59.99999 3    1994/12/01 00:00:00.000
   1994/11/30 23:59:59.99900 0    1994/12/01 00:00:00.000
   1994/11/30 23:59:59.90000 0    1994/12/01 00:00:00.000
   1994/11/30 23:59:49.90000 -1   1994/12/01 23:59:50.000
   1994/11/30 23:33:49.90000 -2   1994/12/01 23:34:00.000
   1994/11/30 23:33:49.90000 -4   1994/12/01 24:00:00.000

See also: HR_NORMALIZE_FT, HR_NORMALIZE_FTF

Uses IOSLIB routine NORMALIZE_YMDHMS.

-----------------------------------------------------------------------
```

10.10.23 HR_R8_TO_TI

```
   SUBROUTINE HR_R8_TO_TI(R8,UNITS, TI)

Converts to a Time Increment structure from a single REAL*8 number.
The units of R8 is specified in UNITS.

---Input---

   REAL*8              R8          ! input value

   CHARACTER*(*)       UNITS       ! DAY, HOUR, MIN, SEC, MS
                                   ! any case, only 1st 2 chars used,
                                   ! HR = HOUR
---Output---
   INCLUDE 'ios_header$inc:sr_time_increment.inc'
   RECORD /SR_TIME_INCREMENT/  TI  ! time increment in 5 real*4's
                                   ! and composite(days) in 1 real*8

If the TI.COMPOSITE is not needed, parameter TI may be declared in
the calling routine as a simple array of 7 real*4s, instead of the
time_increment structure.  The first 5 real numbers will be
days, hours, minutes, seconds & milliseconds,
and the last 2 will be rubbish (unless equivalenced to a real*8).

-----------------------------------------------------------------------
```

10.10.24 HR_SET_NULL

```
   SUBROUTINE HR_SET_NULL(TYPE,D,NULLTYPE)

Sets variable D, of type TYPE to the null value described in string
NULLTYPE.  Handles all ios header types except C (character).

---Input---
   CHARACTER*(*)     TYPE      ! type of variable D.  One of:
                               ! I,I2,I4,R4,R8,FT,TI,LL    (upper case)
                               ! NOTE: type Character not allowed.

   INCLUDE 'ios_header$inc:header_basics.inc'
   INCLUDE 'ios_header$inc:alltypes.inc'
   RECORD /ALLTYPES/ D         ! variable to be set: any header type
                               ! except C (character).

   CHARACTER*(*)     NULLTYPE  ! one of the following strings
                               ! (low case): 'blank', 'na', 'n/a',
                               !             'unknown', 'absent'
                               ! OR one of the character null values
                               ! defined in the HC_* parameters.

The HC_* parameters are HC_ABSENT, HC_NA, HC_UNKNOWN, and HC_BLANK,
defined in the NULVALS.INC ios header include file.

Character types cannot be handled because they are passed by descriptor
instead of by address.  To set character strings null, just assign one
of the HC_* values directly.

Calling this routine with Integer or Real types is equivalent to
assigning one of the HN_* null parameters to the variable directly.
----------------------------------------------------------------------
```

10.10.25 HR_SPECIAL_CHANTYPE

```
   LOGICAL*1 FUNCTION HR_SPECIAL_CHANTYPE(CHANTYPE)

Returns a TRUE value if channel type CHANTYPE is a special
channel type (e.g. I, D, DT, FT, LAT, LON, C)

---Input---
   CHARACTER*(*) CHANTYPE      ! channel type

See also: HRC_SPECIAL_CHANNELS
----------------------------------------------------------------------
```

10.10.26 HR_TI_TO_R8

```
   SUBROUTINE HR_TI_TO_R8(TI,UNITS, R8)

Converts from a Time Increment structure to a single REAL*8 number.
The desired units of the result is specified in UNITS.

---Input---
   INCLUDE 'ios_header$inc:sr_time_increment.inc'
   RECORD /SR_TIME_INCREMENT/  TI  ! time increment in 5 real*4's

   CHARACTER*(*)       UNITS       ! DAY, HOUR, MIN, SEC, MS
                                   ! any case, only 1st 2 chars used,
                                   ! HR = HOUR

---Output---
   REAL*8              R8          ! result

Parameter TI may be a declared as simple array of 5 real*4s,
instead of a structure.
----------------------------------------------------------------------
```

10.10.27 HR_TIME_STRUCTURE

```
   SUBROUTINE HR_TIME_STRUCTURE(GETPUT, HOUR,MINUTE,SECOND, TIME)

Copies time information to or from a time structure, depending upon
whether GETPUT is 'PUT' or 'GET'.

*** WARNING: SECOND is type INTEGER, even though seconds are stored
             as REAL in the time structure.

---Input---
   CHARACTER*3   GETPUT    ! instruction: 'GET' from structure, or'
                           !              'PUT' into structure
                           ! (not case sensitive)
---Input or Output---
   INTEGER       HOUR      ! time in separate variables
   INTEGER       MINUTE
   INTEGER       SECOND

   INCLUDE  'ios_header$inc:sr_time.inc'
   RECORD        /SR_TIME/ TIME            ! a time structure

Note: This routine does NOT put data in, or copy data out of a header,
      unless the TIME specified is in a header structure.

      TIME is a TIME structure, not a FULLTIME structure.

      SECOND is type INTEGER, even though seconds are stored
      as REAL in the time structure.

See also: HR_DATE_STRUCTURE, HR_FT_STRUCTURE
----------------------------------------------------------------------
```

10.10.28 HR_VALID_FT

```
   LOGICAL*1 FUNCTION HR_VALID_FT(CHECK_ZONE,FT)

Checks that Full Time FT contains a valid date and time.
If CHECK_ZONE is True, it also checks that the zone is valid
(I.e. is on the IOSLIB list of standard or daylight-savings
time zone abbreviations.)

---Parameters---
---Input---
   LOGICAL*1           CHECK_ZONE  ! True to have the zone checked.

   INCLUDE 'ios_header$inc:sr_fulltime.inc'
   RECORD /SR_FULLTIME/FT          ! full time

----------------------------------------------------------------------
```

10.10.29 HR_Z8_TO_FT

```
   SUBROUTINE HR_Z8_TO_FT(Z8, ZONE, FT)

Converts REAL*8 Z8 to a Full Time in FT.
Z8 is the number of days since the IOS Header library fixed TIME ZERO, UTC.

   INCLUDE 'ios_header$inc:sr_fulltime.inc'
---Input---
   REAL*8                  Z8      ! days since fixed time zero, UTC.
   CHARACTER*(*)           ZONE    ! desired time zone for output FT
                                   ! - if null, no conversion is done, and
                                   !   that same null value is put in ft.zone
                                   ! - if 'hdr', ts_hdr_zone from hr_tsz.inc
                                   !   is used.
---Output---
   RECORD /SR_FULLTIME/    FT      ! a full time, in the specified zone

If zone is 'hdr', HRC_TSZ_PREP must have been called, because
ts_hdr_zone and ts_days_to_utc are used from common hr_tsz.

See also: HR_FT_TO_Z8
----------------------------------------------------------------------
```

10.10.30 HRC_BLANK_HEADER

```
   SUBROUTINE HRC_BLANK_HEADER()

Initializes the header structure by setting all items and table
entries to the BLANK null value.
Items which indicate table sizes (e.g. FIL.NUMBER_OF_CHANNELS) are
set to one.
All table and remark table counters are set to 1, so that one blank
line will be produced for each table.
The Channel Detail Table inclusion flag is set .TRUE.
The Time Stamp is set to the current system date.

---Parameters: NONE

----------------------------------------------------------------------
```

10.10.31 HRC_CHECK_HEADER

```
   SUBROUTINE HRC_CHECK_HEADER(MODE,STAT)

Checks that mandatory sections and items are present in the header.

---Input---
   CHARACTER*(*)        MODE    ! 'IN' or 'OUT', controls whether
                                ! section INCLUDE_IN or OUT flags
                                ! are checked.

---Output---
   INTEGER              STAT    ! > 0 if any sections or items missing
                                ! 1 warnings only produced
                                ! 2 error messages produced

The Mandatory Item File defines:
  - which sections and items are mandatory,
  - what null values make and item 'missing',
  - whether a warning or an error message is produced if it is missing.
```

```
The Mandatory Item File is found as follows:
-------------------------------------------
  VAX: - pointed to by logical name IOS_HEADER$MANDITEM
  DOS: - named completely in environment variable MANDITEM
       - if MANDITEM is not set, MANDITEM.TXT in the current directory
         is looked for.

The Mandatory Item File is coded as follows:
-------------------------------------------

Sections and items are specified independently.
If an item is specified as mandatory, but its section is not, the
item is mandatory only if the section is present.
(I.e. if the section's input or output flag is turned on.)

There are two types of Mandatory Item File records:

SECTION <section_id>           [ <severity> ]
ITEM <section_id> <item_label> [ <severity> [ <missing_spec> ] ]

where:
  [ ]
    - indicate optional parameters

  <section_id>
    - must be in upper case
    - the first three characters of this string must match the first
      three characters of a section name.

  <item_label>
    - must be in upper case
    - must be a valid header section item label
    - if it has embedded blanks, it must be in single quotes
    - custom items can not be made mandatory.

  <severity>
    - must be one of:
      0 - not mandatory, same as if omitted from the mandatory item
          file.
      1 - warning
      2 - error (default)
      This is the status which will be returned from the header read
      and write routines if the item or section is missing.
    - if <severity> is omitted, 2 is assumed
    - <severity> must be coded, if <missing_spec> is to be coded.

  <missing_spec>
      - indicates what is to be considered a 'missing' item
      - must be a single string containing 1 to 4 of the following
        characters:
            a - absent, n - n/a, ? - unknown, b - blank
      - e.g. 'ab' means the item will be considered 'missing' if it is
        null-absent, or null-blank.
      - omitting <missing_spec> is equivalent to specifying 'a'.

NOTES:

    - the records may be coded in any order.
    - because ITEM records include the section_id, they are independent
      of SECTION records.
    - everything EXCEPT <missing_spec> characters must be upper case.
    - <missing_spec> characters must be lower case
    - there is no column dependence.
    - single quotes around strings are optional, except in the case of
      item labels with embedded blanks.
    - blank lines may be included
    - comments may be added after an '!' on any line.

Examples:

Example 1:

    SECTION LOCATION 1
    ITEM LOC LATITUDE   2    ab          ! comment
    ITEM LOC LONGITUDE  2    ab
    ITEM FIL 'NUMBER OF RECORDS'

  - a Warning will be produced if the LOCATION section is missing.
  - if the LOCATION section is present, then error messages will be
    produced if the latitude or longitude are absent or blank
  - if the FILE section is present, then an error message will be
    produced if the 'NUMBER OF RECORDS' item is absent.
    (It may be blank.)

Example 2:

    SECTION ADM
    ITEM ADM CRUISE

  - An error message will be produced if the ADMINISTRATION section
    missing, or if the CRUISE item in the ADMINISTRATION section is
    absent.

----------------------------------------------------------------------
```

10.10.32 HRC_DEFAULT_FORMAT

```
   SUBROUTINE HRC_DEFAULT_FORMAT()

Sets the FORMAT item blank, and sets the format information in the
channel details table to default values.  The TYPE column in the
detail table is used.  If a type is missing, or not recognized,
the type is assumed to be R4 (REAL*4) unless the DATA TYPE item
contains an 8, in which case the type is assumed to be R8 (REAL*8)

----------------------------------------------------------------------
```

10.10.33 HRC_FT_TO_R8

```
   SUBROUTINE HRC_FT_TO_R8(FT, R8)

Converts a Full Time to a REAL*8 number of days from the TIME ZERO
specified in the FILE section of the header.  If the TIME ZERO item
is not specified, the header library Fixed Time Zero used as time zero.
If the FT time zone differs from that of time zero, the result is
converted to the TIME ZERO time zone.

   INCLUDE 'ios_header$inc:header_common.inc'
---Input---
   RECORD /SR_FULLTIME/    FT      ! a full time
---Output---
   REAL*8                  R8      ! days since time zero.

See also: HRC_R8_TO_FT, HR_Z8_TO_FT
```

```
----------------------------------------------------------------------
```

10.10.34 HRC_INIT_HEADER

```
   SUBROUTINE HRC_INIT_HEADER()

Initializes the header structure in common block IOS_HEADER by setting
the time stamp to the current date and time, all items to the ABSENT
null value, and table entries to either the BLANK or ABSENT null value.

---Parameters: NONE

----------------------------------------------------------------------
```

10.10.35 HRC_ISNULL_ITEM

```
   LOGICAL*1 FUNCTION HRC_ISNULL_ITEM(SECTION,LABEL)

Returns TRUE if the item LABEL in section SECTION is any of the
IOS Header null values, or if the item does not exist in the header.

---Input---
   CHARACTER*(*)       SECTION     ! header section
                                   ! (1st 3 chars needed only)
   CHARACTER*(*)       LABEL       ! item label


Works with standard and custom items.

If the item is not found, a value of TRUE is returned.
----------------------------------------------------------------------
```

10.10.36 HRC_R8_TO_FT

```
   SUBROUTINE HRC_R8_TO_FT(R8, FT)

Converts REAL*8 R8 to a Full Time in FT.
R8 is the number of days since the TIME ZERO specified in the
FILE section of the header.  If the TIME ZERO item
is not specified, the header library fixed time zero is used.
The TIME ZERO time zone is assumed.  If TIME ZERO is null, then the
START TIME item time zone is inserted.  If that is null, then FT will
have a null zone.

   INCLUDE 'ios_header$inc:header_common.inc'
---Input---
   REAL*8                  R8      ! days since time zero.
---Output---
   RECORD /SR_FULLTIME/    FT      ! a full time

See also: HRC_FT_TO_R8, HR_FT_TO_Z8
----------------------------------------------------------------------
```

10.10.37 HRC_REMOVE_SECTION

```
   SUBROUTINE HRC_REMOVE_SECTION(SECTION)

Sets the INCLUDE_OUT flag to FALSE for one or ALL sections of the
header.
```

```
---Parameters---

   CHARACTER*(*)   SECTION     ! section name, or 'ALL'
                               ! (only 1st 3 chars used)

Note: this routine just sets flags, it does not actually blank
      out data in memory.

If you have the header structure declared, the following two
statements are equivalent:
   CALL HRC_REMOVE_SECTION('FILE')
   H.FIL.INCLUDE_OUT = .FALSE.

----------------------------------------------------------------------
```

10.10.38 HRC_SET_ZERO_CONDAY

```
   SUBROUTINE HRC_SET_ZERO_CONDAY(ZTIME,ZZONE)

Sets the IOSLIB zero conday to the header time_zero day,
or to the header library fixed time zero.


---Output---
   REAL*8              ZTIME      ! time of day, in days, of time zero set
   CHARACTER*3         ZZONE      ! header zone

If the header time_zero is non-null, the day of that full time is set,
and if it has a non-zero time, that time is returned in ZTIME.

If the header time_zero is null, the header library fixed time zero is set,
and zero is returned in ZTIME.

If the header time_zero is used, its zone is returned in ZZONE.
Else the zone of the header's start_time is returned in ZZONE, even if
it is null.

See also: HR_SET_FIXED_ZERO_CONDAY
----------------------------------------------------------------------
```