UNLIMITED DISTRIBUTION

National Defence
Research and Development Branch

Défense nationale
Bureau de recherche et développement

TECHNICAL COMMUNICATION 89/305
June 1989

OFFSRF:
A SYSTEM FOR REPRESENTING SHIP OFFSET DATA

David Hally

Defence Research Establishment Atlantic

Centre de Recherches pour la Défense Atlantique

Canada

DEFENCE RESEARCH ESTABLISHMENT ATLANTIC

9 GROVE STREET
P O BOX 1012
DARTMOUTH, N.S.
B2Y 3Z7

TELEPHONE
(902) 426-3100

CENTRE DE RECHERCHES POUR LA DÉFENSE ATLANTIQUE

9 GROVE STREET
C.P. 1012
DARTMOUTH, N.É.
B2Y 3Z7

DISTRIBUTION LIMITED

National Defence
Research and Development Branch

Défense nationale
Bureau de recherche et développement

OFFSRF:
A SYSTEM FOR REPRESENTING SHIP OFFSET DATA

David Hally

June 1989

Approved by W.C.E. Nethercotte
H/Hydronautics Section

Distribution Approved by

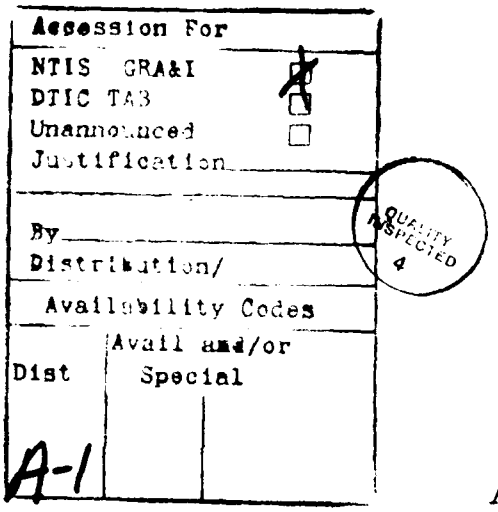
D/TD

TECHNICAL COMMUNICATION 89/305

Defence Research Establishment Atlantic

Centre de Recherches pour la Défense Atlantique

Canada



Abstract

It is argued that a standard format for representation of offset data in computer files would save time for both users and programmers of hydrodynamic programs. A system suitable for adoption as a standard is described. It includes standard formats for offset data on stations, on bow and stern profiles, and on transoms, as well as formats for common hull features such as knuckles, bilge keels, the design waterline, the deck edge, and the half-siding. The format is extensible so that additional data may also be included in the offset files.

To ease the burden on the programmer, Fortran sub-programs to read the standard data are provided; they provide extensive error checking and accurate pin-pointing of the location of errors in the file. Programmer's manuals for the sub-programs are included.

Certains chercheurs ont émis l'hypoyhèse que le recours à un format standard de représentation de donné des devis de tracés dans les ficiers informatisés permettrait de réduire le temps consacré par les utilisateurs et les programmeurs de logiciels d'hydrodynamique. Le texte décrit un système qui pourrait éventuellement être adopté comme norme. Il comprend des formats standard pour les données des devis de tracé relatives aux stations, aux profils de proue et de poupe et aux arcasses, ainsi que des formats liés aux caractéristiques générales des coques comme, par exemple, les coudes, les quilles de roulis, la ligne de flottaison théorique, l'abord du pont et le demi-échantillon. Le format n'est pas limité, de sorte qu'il sera possible d'adjouter ..es données aux fichiers des devis.

Afin de faciliter la tâche des programmeurs, on fournit des sous-programmes en Fortran pour la lecture des données standard; ils assurent l'ensemble des fonctions de vérification des erreurs et de repérage exact des erreurs dans le fichier. On trouvera aussi des manuels du programmeur sur les sous-programmes.

Contents

List of Figures

Notation

$\mathbf{n}$: A *normal vector to the transom*: $\mathbf{n} = (n_X, 0, n_Z)$.

n_X: X coordinate of a normal to the transom.

n_Z: Z coordinate of a normal to the transom.

$\mathbf{X}$: Cartesian coordinate vector: $\mathbf{X} = (X, Y, Z)$.

X: Cartesian coordinate equal to the distance from the forward perpendicular.

X_t: X value of a point on the transom.

Y: Cartesian coordinate equal to the distance from the centreline.

Z: Cartesian coordinate equal to the distance above the baseline.

Z_t: Z value of a point on the transom.

1 Introduction

Prediction of the hydrodynamic characteristics of ships using computer programs is now commonplace. The most common method for representing the hull shape in the computer is to specify hull offsets in a data file read by the program used; however, different programs commonly require the offsets to be presented in different formats. Hence, to determine the hydrodynamic characteristics of a particular hull it is sometimes necessary to prepare many different offset files. Adoption of a standard format would save a great deal of time when preparing these files either by text editor or by writing programs which convert from one standard to another.

In this memorandum a proposal for a standard is presented; it is called OFFSRF. The following considerations influenced the design of OFFSRF.

1. Any standard must balance between generality and ease of use; that is, the more general a standard is, the more complex it will be, and hence, the less likely is it to be used. The standard must be sufficiently general to allow use in a wide variety of programs, but sufficiently simple in structure that it will find favour amongst the users and programmers for whom it is intended.

2. Since the whole purpose of a standard is to save time in offset file preparation, the standard must be designed so that the offset files are as easily edited as possible.

3. A standard must allow inclusion of data describing varied hull features including the hull dimensions, offset points at stations and on bow and stern profiles, the location of knuckles, the deck edge, the design waterline, bilge keels, and the half-siding.

4. A standard should be extensible so that additional data required by some programs can be incorporated into the data file.

It is important to distinguish between OFFSRF *users* and OFFSRF *programmers*. Users are people who prepare the offset files and run programs which use OFFSRF; they only need to understand the correct formatting of OFFSRF data files. Programmers are people who write programs which use OFFSRF; they need also to understand how to call the sub-programs provided with OFFSRF, and how to use the standard data structures which OFFSRF uses to store the data describing the hull. In the descriptions in the following sections, a clear distinction is made between the knowledge required of the users and the programmers.

2 The Record Structure of the Offset File

The cornerstone of OFFSRF is the sub-division of the offset file into labelled records; that is, the file is sub-divided into groups of data each of which is identified by a label. For

example, all the data necessary to describe a single hull station is grouped into a record called STATION. A record may contain sub-records: thus, the STATION record may have the sub-records NUMBER, containing the station number, and POINTS, containing the offset points. The division of the file into labelled records has a number of advantages.

1. Since the label describes the type of data which it contains, it is easy to identify each datum in the file even if one is unfamiliar with the program for which the data were prepared. Most offset data files currently in use identify the data only by their positions in the file; this makes it very difficult for an inexperienced user to determine the use of a particular datum. In the proposed standard, the order of the records within the file is of no importance; this greatly facilitates the preparation and editing of data files.

2. The records can be organized such that the inclusion or removal of a whole record will not destroy the integrity of the rest of the data: for example, a whole station of offset points could be removed and the remaining data would still be a valid input file. This feature greatly facilitates the editing of the data files. In many offset files, removal of a station of offsets requires adjustments to other data in the file.

3. Data specific to a single program can be grouped into a single record. If a different program uses the same file, this record can simply be ignored. Thus, the record labels are useful for distinguishing data which are required by, or are superfluous to, a particular program.

Although the use of a record structure greatly enhances the ease of use of the offset files for the user, it has the drawback that it increases the work load of the programmer. Every program which uses the standard must be capable of interpreting records and sub-records. This is probably already sufficiently great a task that the standard would seldom be used. To avoid the problem, it is necessary that subroutines be provided for reading data from an offset file. The use of standard subroutines has other advantages.

1. Since subroutines are provided to read data describing many standard hull features, the programmer is free to concentrate on those data which are unique to his program: this is often a small subset of the full set of data required.

2. The subroutines can provide extensive error checking, removing much of the burden from the programmer, and ensuring high performance for the user. Because of the record structure of the file, the location of errors in the file can be pinpointed accurately, thus allowing the user to make corrections quickly.

However, the inclusion of subroutines implies a standard for the data structures used to represent the data in programs using OFFSRF; that is, any program which uses OFFSRF must retrieve the data read from the file and to do so it must use the same data structures as are used by OFFSRF. It is the incompatibility of data structures between existing programs which will likely cause the most trouble for converting those programs so that they may use OFFSRF.

To further enhance the ease of interpretation of data, OFFSRF allows comments to be added to the file.

2.1 The Syntax of the Record Structure

The record structure used by OFFSRF is quite general and can be used to organize data files other than ship offset files. The syntax of the structure is defined in this section.

The start of each record is a line in the file whose first character is {; the characters following the { specify the record label. The end of the record is a line which begins with } and is followed by the record label. Alternatively, the record label can be omitted after the closing }; however, errors are easier to detect if the record label is used. For example, the following two records are equivalent.

{EXAMPLE RECORD #1
data
}EXAMPLE RECORD #1

{EXAMPLE RECORD #1
data
}

When there is very little data in a record, it is sometimes convenient to be able to write the data on the same line as the record label. When this is allowed (one should check the descriptions of the format for each record specified by the programmer), the record label may be followed by a colon, then by the data required. The record may be terminated by a } at the end of the line, or further data may be included on following lines. The following two records are equivalent.

{EXAMPLE RECORD #2: *one line of data* }

{EXAMPLE RECORD #2
one line of data
}EXAMPLE RECORD #2

Similarly, the following are equivalent.

{EXAMPLE RECORD #3: *one line of data*
more data
}EXAMPLE RECORD #3

{EXAMPLE RECORD #3
one line of data
more data
}EXAMPLE RECORD #3

The record labels are case dependent and blanks are significant. The record label begins at the first character following the {; it ends at the last non-space character on the line or at

the character preceding the optional colon. The records beginning with the following lines all have different record labels.

Beginning of record	Record label
{EXAMPLE LABEL	'EXAMPLE LABEL'
{EXAMPLE LABEL	'EXAMPLE LABEL'
{ EXAMPLE LABEL	' EXAMPLE LABEL'
{EXAMPLE LABEL : *data*	'EXAMPLE LABEL '
{Example Label	'Example Label'

A line beginning with ! is a comment line and is ignored when the file is read. Comment lines may only be placed before a record label, or after all the data required by the record: a comment line should not appear in the midst of the data or between the record label and the data; this facilitates the reading of the data without having to check whether each line begins with !.

Blank lines between records are ignored; however, they may be included as data: for example, the name of a hull might be a blank string which must appear in the offset file as a blank line. The contents of any line which is neither blank nor begins with {, } or !, are interpreted as data.

Each record can contain data and/or sub-records. Any explicit data (i.e. data which does not appear in the sub-records) must appear before the first sub-record. The format of the sub-records is exactly the same as for the records. Up to ten levels of sub-records are allowed.

A sub-record may not be specified to the right of the colon on the same line as the record label. The following is *not* allowed.

```
{INCORRECT RECORD: {SUB-RECORD LABEL: sub-record data }
   more data
}INCORRECT RECORD
```

The format of the data within each record or sub-record is not specified by the record structure; it is specified by the programmer who must provide Fortran code to read the data for all records which are recognized by his program. The means by which this is done is described in Section 2.2. The correct format for any record must be provided by the programmer in the documentation of his program.

Figure 1 is an example data file using the record structure; it lists a number of common foods. The file is broken into four records: COMMENT, MEAT, CARBOHYDRATES, and VEGETABLES. The MEAT record contains only data: the four words `Beef`, `Pork`, `Lamb`, and `Chicken`. The CARBOHYDRATES record contains no explicit data, only the sub-records BREAD, PASTA, and CEREAL. The PASTA sub-record contains both data (`Spaghetti` and `Linguine`) and a sub-sub-record, WITH HOLES.

The COMMENT record allows the user to include comments in the file which will be written to the terminal when the file is read. These differ from the comment lines beginning with a ! since those are ignored completely when reading the input file. Typically several

```
!This file lists some common foods
{COMMENT
This is an example file suitable for reading by RDFLRC.

}COMMENT

{MEAT
  Beef
  Pork
  Lamb
  Chicken
}MEAT

{CARBOHYDRATES
  {BREAD: Whole-wheat Rye White}
  {PASTA
     Spaghetti Linguine
     {WITH HOLES: Macaroni Cannelloni Manicotti}
  }PASTA
  {CEREAL: Shreddies Granola}
}

{VEGETABLES
  Eggplant
  {GREEN: Broccoli Cucumber}
  {YELLOW: Turnips Squash}
}VEGETABLES
```

Figure 1: An example file using the record structure

COMMENT records will appear in a file; the message in each is transmitted to the terminal as soon as it is encountered in the file. Blank lines are significant in the COMMENT record; in the example above, the blank line following the line '`This is an example ...`' is also sent to the terminal.

2.2 Programming Using the Record Structure

The Fortran sub-program RDFLRC (ReaD FiLe ReCords) is provided to allow the programmer to interpret the record structure of an input file. A complete programmer's guide for this sub-program is given in Appendix A.1.

RDFLRC allows the programmer to specify which records in the input file are required by his program and which records are optional. If a required record is not present in the file, an error message is sent to the terminal to inform the user that necessary data was missing. If an optional record is not present in the file, the data structures which it governs are given default values; there is no error. With one exception, any record which is not contained in the list of required and optional records is ignored. When a record is ignored there is no error but a message is written to the terminal to inform the user. The exception is the COMMENT record which is never ignored. This structure allows maximum flexibility when the same data file is to be used as input for more than one program. Each program can identify exactly what data it needs but will not be bothered if data which it does not need appears.

2.2.1 Reading the Data in Records

Since the record structure described above contains no knowledge of the data contained in records, it is incumbent upon the programmer to provide Fortran code to read the data from every required or optional record or sub-record. This is done in the programmer-supplied sub-program RDREC (ReaD RECords) which is called with the record label as an argument (the character variable RECLAB). The body of RDREC for the above example might be of the following form.

```
SUB-PROGRAM RDREC(RECLAB,other arguments)
<RDREC variable declarations>

IF (RECLAB.EQ.'MEAT') THEN
   <Read data for the MEAT record>
ELSE IF (RECLAB.EQ.'CARBOHYDRATES') THEN
   CONTINUE
ELSE IF (RECLAB.EQ.'BREAD') THEN
   <Read data for the BREAD record>
ELSE IF (RECLAB.EQ.'PASTA') THEN
   <Read data for the PASTA record>
ELSE IF (RECLAB.EQ.'CEREAL') THEN
   <Read data for the CEREAL record>
ELSE IF (RECLAB.EQ.'VEGETABLES') THEN
   <Read data for the VEGETABLES record>
ELSE IF (RECLAB.EQ.'GREEN') THEN
   <Read data for the GREEN record>
ELSE IF (RECLAB.EQ.'YELLOW') THEN
   <Read data for the YELLOW record>
```

```
ELSE IF (RECLAB.EQ.'DRINK') THEN
   <Read data for the DRINK record>
ELSE
   <Write error message if this record contains data:
    Sub-program CHKREC>
END IF
RETURN
END
```

The descriptor <Read data for the ... record> stands for Fortran code which reads only the data explicitly contained in a record. For example, for the VEGETABLES record this would only read the datum Eggplant; the remaining data in the sub-records GREEN and YELLOW are read by the Fortran code <Read data for the GREEN record> and <Read data for the YELLOW record>. Notice that for the CARBOHYDRATES record, nothing need be done since this record contains only sub-records.

Notice that code for the DRINK record is included although there was no such record in the data file. If this is a required record, its absence will be detected by RDFLRC and an error message will be written. On the other hand, if it is optional, there is no error. In either case, one must provide a means for reading data from the DRINK record for those files in which it is included.

Since the record label does not indicate the level of nesting of a record (i.e. whether it is a record, a sub-record, a sub-sub-record, etc.), it follows that the same record can be used at any level of nesting. For example, each of the three records MEAT, CARBOHYDRATES, and VEGETABLES could be grouped as sub-records to a record called FOOD. No change would need to be made to the sub-program RDREC, except perhaps to add code to read in any data explicitly contained in the FOOD record.

If no Fortran code is found to read the data for a record (i.e. if the final ELSE condition is executed), a check should be made to determine whether the record contains data explicitly: if it does, an error message must be written since that data will be lost; if it does not, there is no error and nothing need be done. A sub-program CHKREC (CHecK RECord) is provided to perform this check and to write the error message: see Appendix A.2. This check is an aid to the programmer in case he forgets to supply a conditional statement to read data from one of the records. If RDREC is correctly programmed, the error message should not be caused by any user error.

Two other sub-programs are provided for use when reading the data from a record. The logical function GNXTLN (Get NeXT LiNe) reads the next non-blank, non-comment line from an input file and returns it in a character variable via its argument list. GNXTLN is provided as a convenient means for the programmer to read the input file while ignoring blank lines and comment lines. A complete programmer's guide for GNXTLN is given in Appendix A.3.

The logical function GNXTNM (Get NeXT NuMber) reads the next series of non-space digits from a character string and interprets them as a number; if there are no non-space digits in the string, the next non-blank, non-comment line in the input file is read first using

GNXTLN. GNXTNM is provided as a convenient way to read numbers from the input file while discarding blank lines and comment lines. A complete programmer's guide for GNXTNM is given in Appendix A.4.

2.2.2 Setting Defaults for Optional Records

If an optional record is not found in the file, default values must be assigned to all the data structures which it governs. It is not sufficient to do this only at the start of the program (for example, using Fortran DATA statements) since, if more than one file is read, the default values may be changed when the first file is read, but will not then be reassigned when the second file is read. Hence, it is necessary to reset the default values every time a file is read. This is done by setting the defaults for all optional records before reading the file; when the file is read, if an optional record occurs in it, the default values will be superseded. The setting of default values for optional records is the responsibility of the programmer. It is suggested that a sub-program be written to perform this task. The sub-program SDFLT (Set DeFauLT) outlined below would be appropriate for the example. SDFLT would be called prior to each call to RDFLRC.

```
SUB-PROGRAM SDFLT
<SDFLT variable declarations>

<Set default values for the MEAT record>
<Set default values for the BREAD record>
<Set default values for the PASTA record>
<Set default values for the WITH HOLES record>
<Set default values for the CEREAL record>
<Set default values for the DRINK record>
RETURN
END
```

3 The Standard Ship Offset File

In this section, the application of the record structure described in the previous two sections is applied to ship offset files. A cartesian coordinate system is used in which X is the distance aft of the forward perpendicular, Y is the distance from the centreplane, and Z is the distance above the baseline: see Figure 2. The hull is assumed symmetric in the centreplane so that it is not necessary to decide whether Y increases to port or starboard.

The record types that are currently recognized are described in detail in the following sections. Other records may be defined for use in particular programs; OFFSRF users should consult the user's guide for the program they are running to determine which records are defined. It is also important to remember that it is up to the programmer to define which of the standard records (or sub-records) are required by his program, which are optional, and which are ignored. The following record types are currently defined by OFFSRF.

1. COMMENT: The standard RDFLRC comment record as described in Section 2.
2. NAME: This record is used to specify the name of the hull.
3. PERPENDICULAR STATIONS: This record is used to specify the station numbers of the forward and aft perpendiculars.

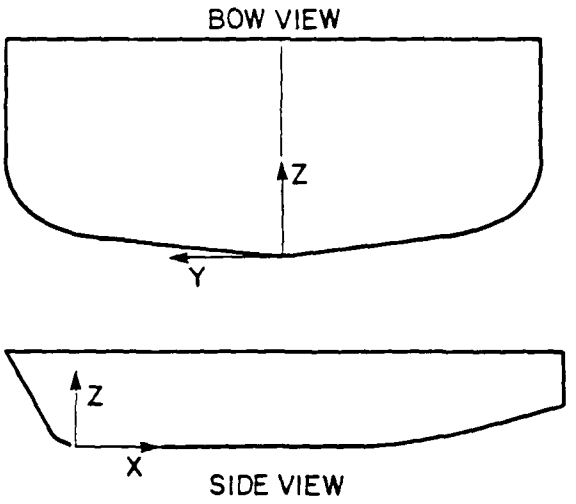


Figure 2: Coordinate System used in OFFSRF

4. DIMENSIONS: This record is used to specify the length, beam, and depth of the hull and the units in which these dimensions are given.

5. STATION: This record is used to specify the offset data at a single station. Normally many STATION records will appear in an offset file. It has two sub-records.

   (a) NUMBER: This sub-record contains the station number of the station record.

   (b) POINTS: This sub-record contains a list of offset points for the station. It also allows some offset points to be marked: those lying on knuckles, bilge keels, the waterline, the deck edge, and the half-siding.

6. BOW PROFILE OFFSETS: This record contains a list of offset points lying on the bow profile.

7. STERN PROFILE OFFSETS: This record contains a list of offset points lying on the stern profile.

8. TRANSOM: This record describes the position and slant of the transom. It also has a sub-record, TRANSOM OFFSETS, which contains offset points for the edge of the transom.

Figure 3 is an example of a simple offset file.

3.1 The COMMENT Record

The COMMENT record contains comments which will be written to the terminal (or batch log file) when the offset file is read; the comments lines beginning with ! are ignored completely. The COMMENT record is useful when it is desirable to alert the user to some feature of the data of which he may otherwise be unaware. For example, the second COMMENT record in the example of Figure 3 alerts the user to the fact that the stations run between 0 and 10, not the standard 0 and 20 used at DREA.

Each line in the comment record can contain up to 80 characters (the standard width of a terminal screen). Lines having more than 80 characters will be truncated.

If a colon is included after the record label, the first line of data is all characters (to a maximum of 80) immediately to the right of the colon. Thus, in the second COMMENT record of the example, the text written on the screen is:

```
WARNING: This hull has station numbers between 0 and 10.
```

The offset file may contain any number of COMMENT records; the contents of each will be written to the terminal as it is encountered when reading the file.

```
{COMMENT
This is a simple example of an offset file.

}COMMENT

{NAME:AN EXAMPLE HULL}

!The station numbers on this hull are between 0 at the forward
!perpendicular and 10 at the aft perpendicular.

{PERPENDICULAR STATIONS: 0 10
  {COMMENT: WARNING: This hull has station numbers between 0 and 10.}
}

{DIMENSIONS
 ft
 100  21  28.5
}DIMENSIONS

{STATION
  {NUMBER: 0.0}
  {POINTS
    0.00000        0.0000
    0.00000        6.0165
    0.06134        14.011    b1
    1.0017         21.480
    3.6592         28.500
  }POINTS
}STATION

{STATION
  {NUMBER: 5.0}
  {POINTS
    0.0000         0.0000
    4.3188         0.8259
   12.624          2.6323
   20.150          8.5839    b1
   20.946          19.009
   20.960          28.500
  }POINTS
}STATION
```

Figure 3: A Simple Offset File

3.2 The NAME Record

The NAME record contains the name of the hull. The name of the hull is read from the line immediately following the record label. Alternatively, if a colon is included after the record label, the text immediately to the right of the colon is the name. The name can contain up to 60 characters; any more characters will be ignored. The default value for the name is a blank string.

Data structures used for storing NAME record data are described in Appendix B.3.

3.3 The STATION Record

The STATION record is the only standard record which contains sub-records and is the only record other than the COMMENT record which will normally appear more than once in the file. The standard sub-records of the STATION record are NUMBER and POINTS. Other sub-records may be allowed by some programs; check the user's guide for the program. By convention, the stations must appear in the file in order of increasing station number.

Because the data in the STATION records must be stored in fixed length arrays, a limit must be put on the number of stations which are allowed in the file so that the arrays don't overflow; this limit is currently set to 50.

Data structures used for storing STATION record data are described in Appendix B.1.

3.3.1 The NUMBER Sub-record

The NUMBER sub-record contains the station number for the STATION record in which it is contained. The format for the record is

{NUMBER
station-number
}NUMBER

or

{NUMBER: *station-number* }

where *station-number* denotes the station number. Data structures used for storing NUMBER record data are described in Appendix B.1.

3.3.2 The POINTS Sub-record

The POINTS sub-record contains offset points for the STATION record in which it is contained. By convention, these points should be in order from the keel towards the waterline or deck. However, there is no restriction on where the last point on a station must lie; it need not be on the waterline or the deck edge.

Because the offset points must be stored in fixed length arrays, a limit must be put on the number of points allowed per station so that the arrays don't overflow; this limit is currently set to 70.

The format for the record is

```
{POINTS
    Y-value-1   Z-value-1   optional-point-marks
    Y-value-2   Z-value-2   optional-point-marks
        ⋮           ⋮               ⋮
    Y-value-n   Z-value-n   optional-point-marks
}POINTS
```

The POINTS sub-record does not allow a colon after the sub-record label. Data structures used for storing POINTS record data are described in Appendix B.1.

The optional point marks are a series of alphabetic characters which are used to indicate whether a point is associated with some hull feature: for example, a 'b' is used to indicate that a point lies on a bilge keel. The following marks are recognized by OFFSRF; others may be defined for use in particular programs. OFFSRF users should consult the user's guide for the program they are running to determine which point marks are defined.

1. b or B: the point lies on a bilge keel.
2. w or W: the point lies on the design waterline.
3. k or K: the point lies on a knuckle.
4. h or H: the point lies on the half-siding.
5. d or D: the point lies on the deck edge.

Since one expects only one point at each station to lie on the waterline and the half-siding, the point marks for these features are straightforward; the marks for the other features may be more complicated.

Data structures used for storing the point marks are described in Appendix B.2.

Point Marks for Knuckles and Bilge Keels

If there are two bilge keels or two knuckles at a given station, the points for each must be distinguished. This is done by adding a single digit after the alphabetic character. For example, *the second point in the POINTS record*

```
{POINTS
  0.0   0.0
  5.0   0.0 b1
  9.0   1.0 b2
  10.0  5.0 wk
  10.0 10.0
}POINTS
```

lies on the first of two bilge keels (marked by the 'b1'), the third point lies on the second bilge keel (marked by the 'b2'), while the fourth point lies both on the waterline (marked by the 'w') and a knuckle (marked by the 'k').

Point Marks for the Deck Edge

As the offset points will often stop at or before the deck edge, it is usually not necessary to identify the points which lie on it. However, when the points are given along the deck as well as the hull (as is sometimes necessary in structural applications), it will be desirable to mark the point on the deck edge. Also, when there are corners in the deck, it may be necessary to know at which stations the corners occur; the two marks 'd' and 'D' are used to identify the sections of the deck edge between corners.

Figure 4 shows how corners can be marked; the dots represent the offset points on the deck edge at successive stations. The segment of the deck before the corner is marked with 'd'; the segment after the corner is marked with 'D'; the corner itself is marked with 'dD' to indicate that it belongs, in part, to each segment.

If there are two corners, the marking on the third segment reverts to 'd'; i.e. the mark alternates between 'd' and 'D' on successive segments. A segment which only extends between two stations, should be marked as in Figure 5.

If there is a step in the deck (i.e. a corner whose angle is 90°), it may be marked as in Figure 6.

In this way, by alternating between 'd' and 'D' on successive deck segments, any deck edge can be marked, provided that the corners in the deck all have angles greater than 90°; Figure 7 shows a deck edge that cannot be marked in this way. OFFSRF currently has no way of representing deck edges with acute corners.

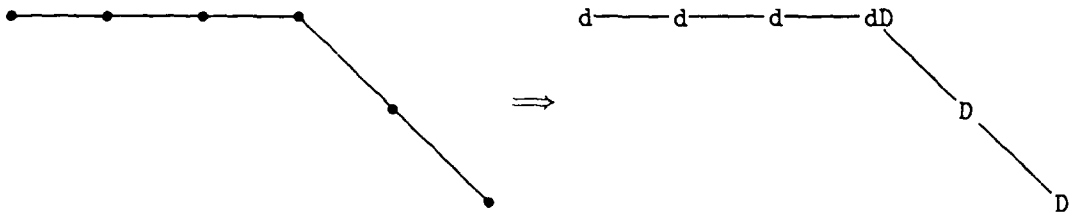


Figure 4: Marking a corner in the deck edge

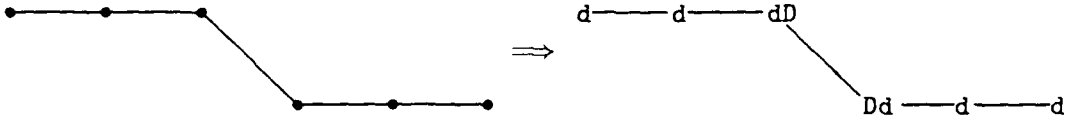


Figure 5: Marking a double corner in the deck edge

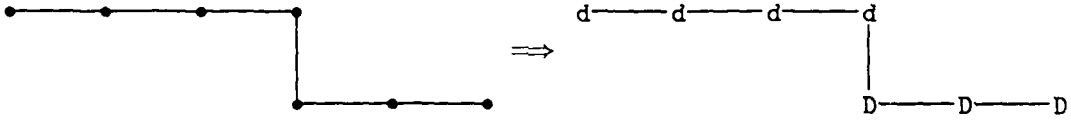


Figure 6: Marking a step in the deck edge

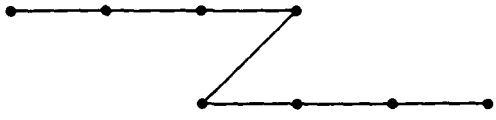


Figure 7: Deck edge with an acute angle

3.4 The DIMENSIONS Record

The DIMENSIONS record contains data describing the overall dimensions of the hull: the length between perpendiculars, the half-breadth, and the depth. It also contains a character string which specifies the units in which the dimensions are given.

The format for the DIMENSIONS record is

{DIMENSIONS
units
LBP half-breadth depth
}DIMENSIONS

The character string *units* specifies the units in which the hull dimensions are given (e.g., 'feet', 'ft', 'mm'); it may be up to 10 characters long. The positions of the three real numbers *LBP*, *half-breadth* and *depth* on the second line of the record is immaterial but they must be in the correct order.

The DIMENSIONS record does not allow a colon after the record label.

Data structures used for storing DIMENSIONS record data are described in Appendix B.4.

3.5 The PERPENDICULAR STATIONS Record

The PERPENDICULAR STATIONS record contains the station numbers of the forward and aft perpendiculars. The format for the record is

{PERPENDICULAR STATIONS
fp ap
}

or

{PERPENDICULAR STATIONS: *fp ap* }

where *fp* and *ap* denote the station numbers of the forward and aft perpendiculars respectively. Their default values are 0 and 20 respectively.

Data structures used for storing PERPENDICULAR STATIONS record data are described in Appendix B.5.

3.6 The BOW PROFILE OFFSETS Record

The BOW PROFILE OFFSETS record contains offset points on the bow profile. Because the offset points must be stored in fixed length arrays, a limit must be put on the number of points allowed so that the arrays don't overflow; this limit is currently set to 70.

The format for the record is

{BOW PROFILE OFFSETS
X-value-1 *Z-value-1*
X-value-2 *Z-value-2*
⋮ ⋮
X-value-n *Z-value-n*
}BOW PROFILE OFFSETS

The offset points 1 to n should begin at the keel and proceed towards the deck edge (waterline).

Data structures used for storing BOW PROFILE OFFSETS record data are described in Appendix B.6.

3.7 The STERN PROFILE OFFSETS Record

The STERN PROFILE OFFSETS record contains offset points on the stern profile. As with the bow profile offsets, there is a limit of 70 offset points.

The format for the record is similar to the BOW PROFILE OFFSETS record.

{STERN PROFILE OFFSETS
X-value-1 *Z-value-1*
X-value-2 *Z-value-2*
⋮ ⋮
X-value-n *Z-value-n*
}STERN PROFILE OFFSETS

The offset points 1 to n should begin at the keel and proceed towards the deck edge (waterline).

Data structures used for storing STERN PROFILE OFFSETS record data are described in Appendix B.7.

3.8 The TRANSOM Record

The TRANSOM record contains data describing the position and orientation of the transom. This record allows representation of the transom as a plane which intersects the hull near the stern (see Figure 8). The TRANSOM record does not allow the representation of curved transoms or more complex transoms consisting of several different planes as in Figure 9. However, for most purposes, the hull of Figure 9 could be adequately represented by including the line A–B in the deck edge rather than the transom.

The transom plane is defined by specifying a point through which it passes, and a normal vector. Let the cartesian coordinate vector of the point through which the transom passes be

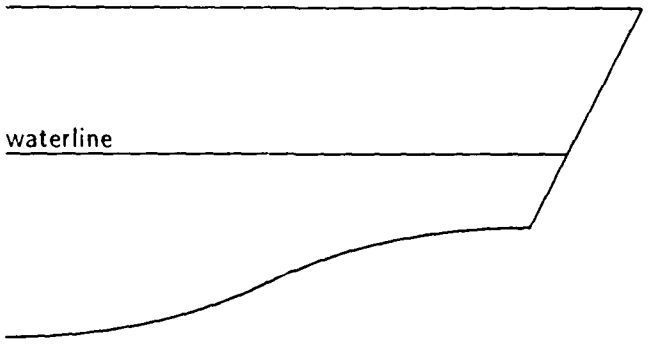


Figure 8: Profile of a simple transom

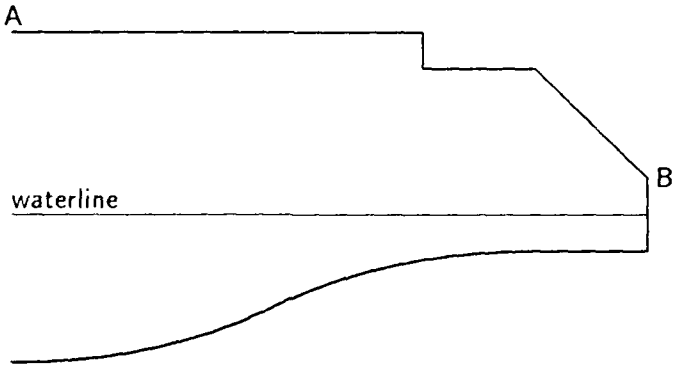


Figure 9: Profile of a complex transom

$\mathbf{X}_t$ and its normal $\mathbf{n}$. A point $\mathbf{X}$ lies on the transom if

$$(\mathbf{X} - \mathbf{X}_t) \cdot \mathbf{n} = 0 \tag{3.1}$$

By symmetry, n_Y is zero, and one can choose $\mathbf{X}_t$ to lie on the centreplane so that $Y_t = 0$. Hence, to define the transom, only X_t, Z_t, n_X and n_Z need be given.

The TRANSOM record has a sub-record, TRANSOM OFFSETS which gives a list of points lying on the edge of the transom. The format of the TRANSOM record is

{TRANSOM: X_t Z_t n_X n_Z
  {TRANSOM OFFSETS
    X-value-1 *Y-value-1* *Z-value-1*
    X-value-2 *Y-value-2* *Z-value-2*
    ⋮ ⋮ ⋮
    X-value-n *Y-value-n* *Z-value-n*
  }TRANSOM OFFSETS
}TRANSOM

or

{TRANSOM
  X_t Z_t n_X n_Z
  {TRANSOM OFFSETS
    X-value-1 *Y-value-1* *Z-value-1*
    X-value-2 *Y-value-2* *Z-value-2*
    ⋮ ⋮ ⋮
    X-value-n *Y-value-n* *Z-value-n*
  }TRANSOM OFFSETS
}TRANSOM

The transom offset points 1 to n should begin at the keel and proceed towards the deck edge (waterline).

Data structures used for storing TRANSOM record data are described in Appendix B.8.

4 Concluding Remarks

It has been argued that a standard format for representation of offsets would save time for both users and programmers of hydrodynamic programs. The OFFSRF system described in this memorandum is suitable for adoption as a standard. It has the following features.

1. It strikes a balance between generality in the types of hull features which can be described, and ease of use for both users and programmers.

2. It has standard records which describe the following hull features: the hull dimensions; offset points at stations, on bow and stern profiles, and on the transom; the location of knuckles, the deck edge, the design waterline, bilge keels, and the half-siding; and the station numbers of the perpendiculars.

3. It is extensible so that additional data required by some programs can be incorporated into the offset file.

4. It has been designed so that the offset files can be edited easily. In particular, whole stations of offsets can be deleted or inserted with no other modifications to the file: for example, a datum giving the number of stations need not decremented or incremented, since OFFSRF determines the number of stations by counting the number of STATION records in the file. Similarly, offset points on the stations can be deleted or inserted with no other modifications needed.

5. Subroutines are provided to read the offset files: data describing the standard hull features are loaded into standard data structures and made available to the programmer via common blocks. The subroutines provide extensive error checking, removing much of the burden from the programmer, and ensuring high performance for the user. Because of the record structure of the file, the location of errors can be pinpointed accurately thus allowing the user to make corrections quickly.

Appendix A Sub-Programs for Manipulating the Record Structure

This appendix describes the standard sub-programs for manipulating the record structure; each of these sub-programs is available to the programmer when implementing a file having the record structure described in Section 2. The following sub-programs are given.

RDFLRC: Reads data from an ASCII file formatted using the record structure. The data read from the file are made globally accessible by storing them in named common blocks.

RDREC: Reads data from any single record in the input file; it is called by RDFLRC and is the sub-program which must be modified by the programmer if non-standard records are to be read by RDFLRC.

CHKREC: Checks to see whether a record contains any explicit data or only sub-records. CHKREC is called by RDREC.

GNXTLN: Reads the next non-blank, non-comment line from an input file and returns it in a character variable via its argument list. GNXTLN is provided as a convenient means for the programmer to read the input file while ignoring blank lines and comment lines.

GNXTNM: Reads the next series of non-space digits from a character string and interprets them as a number; if there are no non-space digits in the string, the next non-blank, non-comment line in the input file is read first using GNXTLN. GNXTNM is provided as a convenient way to read numbers from the input file while discarding blank lines and comment lines.

A.1 RDFLRC Programmer's Guide

Purpose: The logical function RDFLRC reads data from a file in ASCII format having the record structure described in Section 2. It is assumed that the file has already been opened and assigned a logical unit number. The data read from the file is made globally accessible by storing them in named common blocks; the variables and common blocks for the standard OFFSRF records are described in Appendix B.

RDFLRC calls the sub-program RDREC described in Section 2.2.1 and Appendices A.2 and B.11; it is passed to RDFLRC as an argument. RDREC must be modified by the programmer if non-standard records are to be included in the file.

RDFLRC also makes use of the GETWRD Package[1] of sub-programs which are used to handle all output to the terminal. On the DREA DEC-20/60 the GETWRD Package may be found in PS:<HALLY>GETWRD.FOR.

Input Arguments:

IUNIT: The unit number assigned to the file.

RDREC: A logical function used to read data from individual records. Its format is described in Appendix A.2.

RECRDS: A character variable used to specify which records are required to appear in the file, which are optional, and which may be ignored. RECRDS has the following format.

{required records}[optional records]

or

[optional records]{required records}

If there are no required records, the required list (including the curly brackets) is left out; similarly if there are no optional records. Within the required and optional record lists, the labels of the records are separated by commas. Each record label may be followed by lists of required and optional sub-records in curly and square brackets respectively. The sub-record lists have exactly the same format as the record lists and may contain lists of sub-sub-records and so on. For example, with reference to the file given in Figure 1, if RECRDS has the value

`{MEAT,CARBOHYDRATES[PASTA{WITH HOLES},CEREAL]}[VEGETABLES,DRINK]`

then the MEAT and CARBOHYDRATES records are required; the VEGETABLES and DRINK records are optional. The CARBOHYDRATES record has two optional sub-records, PASTA and CEREAL, and no required sub-records. Also, the PASTA sub-record has a required sub-sub-record, WITH HOLES. A required sub-record of an optional record must be present in the file when its parent record is, but need not appear when the parent record is absent. Thus, the WITH HOLES record must appear whenever the PASTA record is present, but may be missing if the PASTA record is also missing.

Any record or sub-record which does not appear in RECRDS will be ignored if it is encountered in the input file. The BREAD sub-record which appears in Figure 1 as a sub-record of CARBOHYDRATES is not included in the example record list; it will be skipped, without error, when the file is read.

Returned Value: RDFLRC returns true if the file is read without error, false otherwise.

Output Arguments:

IER: An integer error flag which may have the following values; other values may be assigned to IER by programmer-supplied subroutines to read in data from non-standard records: see Section 2.2.1 and Appendix A.2.

0, no error.

100, the record structure is too highly nested; only 10 levels of nesting are allowed.

101, data was found in a file when a record label was expected; this is normally caused by superfluous data in a record.

102, A record label was empty: i.e. a line beginning with { had no record label.

103, The end-of-file was encountered before all records had been terminated.

104, there is an error in the format of the list of records, RECRDS; this is a programming error, not a user error.

105, a required record was not found in the file.

108, a required or optional record containing data was found in the file, but there is no sub-program assigned to read the data from it. This is a programming error, not a user error.

109, an error occurred when repositioning the position marker in the input file to reread a line; this error should never occur.

110, a read error occurred when reading a line from the file.

111, the end of a record was not correctly matched with its beginning.

Output to the terminal: RDFLRC writes error messages on the terminal whenever an error occurs. The data within a COMMENT record is also written on the terminal. All terminal output is handled using the GETWRD Package[1] sub-program WRTERM.

Calling Sequence: On the DREA DEC-20/60, a program `main-program` using RDFLRC can be compiled and executed as follows.

```
DEFINE H: PS:<HALLY>
EXECUTE main-program,H:RDFLRC,H:GETWRD/LIB
```

If an optional record is not found in the input file, the data structures which it governs are left unaltered by RDFLRC: i.e. RDFLRC uses as default values, those values which the data structures had when RDFLRC was called. As described in Section 2.2.2, this is often not adequate if more than one file is to be read. Hence, it is usual to define default values for the record prior to each call to RDFLRC.

In `main-program` RDFLRC may be called as follows.

```
CHARACTER RECRDS*(*)
INTEGER IUNIT, IER
LOGICAL RDREC, RDFLRC
EXTERNAL RDREC

<Set default values for optional records>
IF (RDFLRC(IUNIT,RDREC,RECRDS,IER)) THEN
   <File is OK: continue normally>
ELSE
   <File has errors>
END IF
```

A.2 RDREC Programmer's Guide

Purpose: The logical function RDREC is called by RDFLRC to read any data which is contained explicitly in a record. If non-standard records are to be contained in the input file, RDREC must be modified by the programmer to include Fortran code which will read the data from the records which he has defined. Appendix B.11 lists the complete Fortran source code for the version of RDREC used to read the standard OFFSRF records. This code may be used as a template by the programmer wishing to modify RDREC.

RDREC begins reading data from the file on the line following the record label.

Input Arguments:

IUNIT: The unit number assigned to the input file.

RECLAB: A character variable containing the label of the record whose data is to be read. The length of RECLAB is set by the calling program.

FRSTLN: A character variable which is used when a colon is allowed on the line containing the record label. FRSTLN contains the portion of the line to the right of the record label.

Returned Value: RDREC returns true if the record data is read without error, false otherwise.

Output Arguments:

IER: An integer error flag which may have the following values; other values may be assigned to IER by sub-programs which read in data from records: see Section 2.2.1.

- 0, no error.
- 108, a required or optional record containing data was found in the file, but there is no Fortran code or sub-program assigned to read the data from it.
- 109, an error occurred when repositioning the position marker in the input file to reread a line; this error should never occur.
- 110, a read error occurred when reading a line from the file.

Output to the terminal: RDREC writes error messages on the terminal whenever an error occurs. The data within a COMMENT record is also written on the terminal. All terminal output is handled using the GETWRD Package[1] sub-program WRTERM.

Calling Sequence: In RDFLRC, RDREC is called as follows.

```
CHARACTER RECLAB*(*), FRSTLN*(*)
INTEGER IUNIT, IER
LOGICAL RDREC

IF (RDREC(IUNIT,RECLAB,FRSTLN,IER)) THEN
   <Record was read without error: continue normally>
ELSE
   <An error occurred>
END IF
```

Example: When the record

```
{VEGETABLES: Beets
  Eggplant
  {GREEN: Broccoli Cucumber}
  {YELLOW: Turnips Squash}
}VEGETABLES
```

is encountered in the input file, RDREC is called with RECLAB = 'VEGETABLES' and FRSTLN = ' Beets'. The first line read from the file will be the one containing the string 'Eggplant'. When RDREC returns, the position marker in the file will be such that the next line read from the file will be the one starting ' {GREEN:...'.

A.2.1 CHKREC Programmer's Guide

Purpose: The logical function CHKREC is used by RDREC to determine whether a record contains any explicit data. As explained in Section 2.2.1, CHKREC is required to determine whether an error has occurred if no Fortran code or sub-program can be found to read the data for a record. When CHKREC returns, the position marker in the file is at the beginning of the line following the record label.

Arguments: as in RDREC.

Output to the terminal: CHKREC writes error messages on the terminal whenever an error occurs. All terminal output is handled using the GETWRD Package[1] sub-program WRTERM.

Calling Sequence: CHKREC can be called as follows.

```
CHARACTER RECLAB*(*), FRSTLN*(*)
INTEGER IUNIT, IER
LOGICAL CHKREC

IF (CHKREC(IUNIT,RECLAB,FRSTLN,IER)) THEN
   <There is no error: continue normally>
ELSE
   <An error occurred>
END IF
```

Example: When the record is

```
{VEGETABLES: Beets
  Eggplant
  {GREEN: Broccoli Cucumber}
}VEGETABLES
```

and RDREC has no code for reading data from a VEGETABLES record, then CHKREC will be called with RECLAB = 'VEGETABLES' and FRSTLN = ' Beets'. IER is returned with a value of 108 and an error message is sent to the terminal.

Alternatively, if the record is

```
{VEGETABLES
  {GREEN: Broccoli Cucumber}
}VEGETABLES
```

and RDREC has no code for reading data from a VEGETABLES record, then CHKREC will be called with RECLAB = 'VEGETABLES' and FRSTLN = ' '. IER is returned with a value of zero and no error message is sent.

An example of the use of CHKREC is given in Appendix B.11.

A.3 GNXTLN Programmer's Guide

Purpose: The logical function GNXTLN reads the next line in the input file which is neither blank nor is a comment line; a comment line is any line beginning with !. Leading and trailing blanks are stripped from the line.

Input Argument:

IUNIT: The unit number from which the line is read.

Returned Value: GNXTLN returns true if the next line is obtained without error; in this case IER will have a value of zero. If the end of the file was encountered before the next line, GNXTLN returns false and IER is negative. On other errors, GNXTLN returns false and IER is positive.

Output Arguments:

LINE: A character string in which the next line, stripped of leading and trailing blanks, is returned.

LENLIN: The position in LINE of the last non-space character.

IER: An integer error flag.

- 0, no error.
- 110, an error occurred while reading the line.
- < 0, the end of the file was encountered before the next non-comment, non-blank line.

Calling Sequence: GNXTLN may be called as follows.

```
CHARACTER LINE*80
INTEGER IER
LOGICAL GNXTLN

IF (GNXTLN(IUNIT,LINE,LENLIN,IER)) THEN
   <Continue normally>
ELSE IF (IER.LT.0) THEN
   <End of file>
ELSE
   <An error occurred>
END IF
```

A.4 GNXTNM Programmer's Guide

Purpose: The logical function GNXTNM is provided as an aid to reading data from a file while discarding all blank lines and comment lines. Given a character variable, STRING, and an integer pointer, STRPNT, GNXTNM finds the first 'word' of data on the line and attempts to interpret it as a number. It returns the number in a real variable. STRPNT is left pointing to the character after the end of the number. A 'word' is any group of characters delimited by spaces. If there is no 'word' in STRING, the next non-blank, non-comment line is read from the input file, and the next 'word' in that is interpreted as the number.

Successive numbers in the file may be read by clearing STRING of all non-space characters initially and calling GNXTNM repeatedly.

Input Arguments:

IUNIT: The logical unit number of the input file.

STRING: The character variable from which the number is read.

STRPNT: An integer pointer to the current position in STRING. STRPNT should point to a character before the first digit in the number; to find the first number in the string, set STRPNT to zero.

Returned Value: GNXTNM returns true if a number was read without error; false if an error occurred.

Output Arguments:

RNUM: The real number returned.

IER: An integer error flag.

0, no error.

1, the string was blank.

2, *the string contained more than 30 characters.*

other, an IOSTAT error code returned from the host computer; this error occurs when reading the number from the character string.

Calling Sequence: GNXTNM may be called as follows.

```
CHARACTER STRING*(*)
INTEGER IER
LOGICAL GNXTNM
REAL RNUM
```

```
IF (GNXTNM(IUNIT,STRING,STRPNT,RNUM1,IER)) THEN
   <Continue normally>
ELSE
   <An error occurred>
END IF
```

Example: If the input file contains the following data

```
  123  456.7
89.0
```

then after the following code has been executed

```
STRING=' '
NOERR1=GNXTNM(IUNIT,STRING,STRPNT,RNUM1,IER)
IF (NOERR1) THEN
   NOERR2=GNXTNM(STRING,STRPNT,RNUM2,IER)
   IF (NOERR2) NOERR2=GNXTNM(STRING,STRPNT,RNUM3,IER)
END IF
```

the logical variables NOERR1, NOERR2, and NOERR3 will all be true, RNUM1 will have the value 123.0, RNUM2 will have the value 456.7, RNUM3 will have the value 89.0, STRING will have the value '89.0' and STRPNT will have the value 5. After the first call to GNXTNM, STRING has the value ' 123 456.7' and STRPNT has the value 6. After the second call STRING has the value ' 123 456.7' and STRPNT has the value 13.

Appendix B OFFSRF Programmer's Guide

This appendix describes all the data structures used to define the standard OFFSRF records. Since no data from the COMMENT record is stored, there are no data structures associated with it; hence, it need not be discussed here.

B.1 STATION Record Data Structures

The STATION record contains no explicit data, only that contained in its sub-records NUMBER and POINTS. The following data structures are used to store the data found in all sub-records from all STATION records occurring in a file.

NSTMAX: The maximum number of stations allowed. NSTMAX is used to set dimensions for some of the arrays below.

NST: The number of stations. NST must be between zero and NSTMAX inclusive.

STATNO: A real array of length NPPST the first NST elements of which contain the station numbers of the stations. The values of the station numbers are set by the NUMBER sub-record. The default values for the station numbers are that STATNO(I) = I. The station numbers must increase with I.

NPPSMX: The maximum number of points allowed per station.

NPPST: An integer array of length NPPST the first NST elements of which give the number of points on each of the stations. Each element of NPPST must be in the range zero to NPPSMX.

OFFPNT: A real array of dimensions NPPSMX×NSTMAX×2. OFFPNT(I,J,1) contains the Y-value of the I^{th} offset point on the J^{th} station; OFFPNT(I,J,2) contains the Z-value of the I^{th} offset point on the J^{th} station. The offset points are ordered from keel to the deck edge (waterline) as I increases from 1 to NPPST(J).

MARKS: A character array of dimensions NPPSMX×NSTMAX each element of which is of length MAXMRK, where MAXMRK is an integer parameter. The elements of MARKS contain extra information about each of the offset points; in particular, non-space characters in the elements of MARKS are used to indicate which points lie on knuckles, the waterline, bilge keels, etc. For example, if MARKS(12,3) = 'bk', then the twelfth point on the third station lies on a bilge keel (marked by the 'b') and also on a knuckle (marked by the 'k').

These variables are made available globally via the common blocks /STTREC/ and /MRKPNT/ which have the following form. Notice that /STTREC/ and /MRKPNT/ should

be saved (using the Fortran 77 SAVE statement) so that the values of the variables they contain will be preserved.

```
      PARAMETER (NSTMAX=50, NPPSMX=70, MAXMRK=20)
      CHARACTER MARKS*(MAXMRK)
      INTEGER NST, NPPST
      REAL STATNO, OFFPNT
      COMMON /STTREC/ NST, STATNO(NSTMAX), NPPST(NSTMAX),
     +                OFFPNT(NPPSMX,NSTMAX,2)
      COMMON /MRKPNT/ MARKS(NPPSMX,NSTMAX)
      SAVE /STTREC/, /MRKPNT/
```

B.2 Interpreting the Point Marks

After reading a file by calling RDFLRC, the data describing the point marks are stored in the character array MARKS. These data can be re-interpreted and stored in alternative data structures associated with the knuckles, bilge keels, waterline, etc.; this is done by the logical function FRMMRK described in Appendix B.2.6. The second set of data structures is usually the more convenient to use.

When writing an offset file, one will want to convert from the new data structures back to MARKS; this may be done by the sub-program TOMRK described in Appendix B.2.7. The data structures associated with each hull feature are described in the following sections.

B.2.1 Knuckle Data Structures

The following data structures are used to describe knuckles on the hull.

NKNMAX: A parameter which specifies the maximum number of knuckles allowed; the current value is five. NKNMAX is necessary for setting the dimensions of some of the following arrays.

NKNUK: The number of knuckles. NKNUK must be in the range [1,NKNMAX].

STAKNK: A real array which lists the starting station of each knuckle; i.e. the I^{th} knuckle begins at station number STAKNK(I).

ENDKNK: A real array which lists the last station of each knuckle; i.e. the I^{th} knuckle ends at station number ENDKNK(I).

KNKPNT: An integer array of dimensions NSTMAX×NKNMAX. KNKPNT(I,J) is the number of the offset point on the I^{th} station which lies on the J^{th} knuckle; if no point on the I^{th} station lies on the J^{th} knuckle (i.e. if STATNO(I) is not between STAKNK(J) and ENDKNK(J)), then KNKPNT(I,J) is zero. Therefore, the Y and Z values of the offset points on the J^{th} knuckle are OFFPNT(KNKPNT(I,J),I,1) and OFFPNT(KNKPNT(I,J),I,2) respectively.

These variables are made available globally via the common block /KNKDAT/ which has the following form. Notice that /KNKDAT/ should be saved (using the Fortran 77 SAVE statement) so that the values of the variables it contains will be preserved.

```
      PARAMETER (KNKMAX=5, NSTMAX=50)
      INTEGER NKNUK, KNKPNT
      REAL STAKNK, ENDKNK
      COMMON /KNKDAT/ NKNUK, STAKNK(KNKMAX), ENDKNK(KNKMAX),
     +                KNKPNT(NSTMAX,NKNMAX)
      SAVE /KNKDAT/
```

B.2.2 Bilge Keel Data Structures

The following data structures are used to describe bilge keels on the hull.

NBKMAX: A parameter which specifies the maximum number of bilge keels allowed; the current value is five. NBKMAX is necessary for setting the dimensions of some of the following arrays.

NBLGKL: The number of bilge keels. NBLGKL must be in the range [1,NBKMAX].

STABKL: A real array which lists the starting station of each bilge keel; i.e. the I^{th} bilge keel begins at station number STABKL(I).

ENDBKL: A real array which lists the last station of each bilge keel; i.e. the I^{th} bilge keel ends at station number ENDBKL(I).

BKLPNT: An integer array of dimensions NSTMAX×NBKMAX. BKLPNT(I,J) is the number of the offset point on the I^{th} station which lies on the J^{th} bilge keel; if no point on the I^{th} station lies on the J^{th} bilge keel (i.e. if STATNO(I) is not between STABKL(J) and ENDBKL(J)), then BKLPNT(I,J) is zero. Therefore, the Y and Z values of the offset points on the J^{th} bilge keel are OFFPNT(BKLPNT(I,J),I,1) and OFFPNT(BKLPNT(I,J),I,2) respectively.

These variables are made available globally via the common block /BKLDAT/ which has the following form. Notice that /BKLDAT/ should be saved (using the Fortran 77 SAVE statement) so that the values of the variables it contains will be preserved.

```
      PARAMETER (NBKMAX=5, NSTMAX=50)
      INTEGER NBLGKL, BKLPNT
      REAL STABKL, ENDBKL
      COMMON /BKLDAT/ NBLGKL, STABKL(NBKMAX), ENDBKL(NBKMAX),
     +                BKLPNT(NSTMAX,NBKMAX)
      SAVE /BKLDAT/
```

B.2.3 Waterline Data Structures

The following data structures are used to describe the design waterline.

WTRFLG: A logical flag which is true if some of the offsets points are marked as lying on the design waterline, false otherwise.

DESZW: The height of the design waterline above of baseline. DESZW is set to the average of the Z values of all points on the waterline.

STAWTR: A real variable which contains the starting station of the waterline; i.e. the waterline begins at station number STAWTR.

ENDWTR: A real variable which contains the last station of the waterline; i.e. the waterline ends at station number ENDWTR.

WTRPNT: An integer array of length NSTMAX. WTRPNT(I) is the number of the offset point on the I^{th} station which lies on the waterline; if no point on the I^{th} station lies on the waterline (i.e. if STATNO(I) is not between STAWTR and ENDWTR), then WTRPNT(I) is zero. Therefore, the Y and Z values of the offset points on the waterline are OFFPNT(WTRPNT(I),I,1) and OFFPNT(WTRPNT(I),I,2) respectively.

These variables are made available globally via the common block /WTRDAT/ which has the following form. Notice that /WTRDAT/ should be saved (using the Fortran 77 SAVE statement) so that the values of the variables it contains will be preserved.

```
PARAMETER (NSTMAX=50)
INTEGER WTRPNT
LOGICAL WTRFLG
REAL DESZW, STAWTR, ENDWTR
COMMON /WTRDAT/ WTRFLG, DESZW, STAWTR, ENDWTR, WTRPNT(NSTMAX)
SAVE /WTRDAT/
```

B.2.4 Half-Siding Data Structures

The following data structures are used to describe the half-siding.

HFSFLG: A logical flag which is true if some of the offset points are marked as lying on the half-siding, false otherwise.

STAHFS: A real variable which contains the starting station of the half-siding; i.e. the half-siding begins at station number STAHFS.

ENDHFS: A real variable which contains the last station of the half-siding; i.e. the half-siding ends at station number ENDHFS.

HFSPNT: An integer array of length NSTMAX. HFSPNT(I) is the number of the offset point on the I^{th} station which lies on the half-siding; if no point on the I^{th} station lies on the half-siding (i.e. if STATNO(I) is not between STAHFS and ENDHFS), then HFSPNT(I) is zero. Therefore, the Y and Z values of the offset points on the half-siding are OFFPNT(HFSPNT(I),I,1) and OFFPNT(HFSPNT(I),I,2) respectively.

These variables are made available globally via the common block /HFSDAT/ which has the following form. Notice that /HFSDAT/ should be saved (using the Fortran 77 SAVE statement) so that the values of the variables it contains will be preserved.

```
PARAMETER (NSTMAX=50)
INTEGER HFSPNT
LOGICAL HFSFLG
REAL STAHFS, ENDHFS
COMMON /HFSDAT/ HFSFLG, STAHFS, ENDHFS, HFSPNT(NSTMAX)
SAVE /HFSDAT/
```

B.2.5 Deck Edge Data Structures

The deck edge can exhibit three different types of behaviour at a station.

1. It can be continuous and smooth at the station. Only one point at the station lies on the deck edge and it will be marked with 'd' or 'D'.
2. It can have a corner whose angle in greater than 90 degrees. Only one point at the station lies on the deck edge and it will be marked with 'dD' or 'Dd'.
3. It can have a step (a 90° corner) at the station. This means that the deck edge jumps discontinuously from one value to the next at the station; two points at the station lie on the deck edge.

Corners in the deck having angles less than a right angle cannot currently be represented in OFFSRF.

The following data structures are used to describe the deck edge.

DCKFLG: A logical flag which is true if some of the offsets points are marked as lying on the deck edge, false otherwise.

DCKTYP: An integer array of length NSTMAX containing flags which indicate where the discontinuities in the deck edge are. DCKTYP(I) may have the following values.

0, if the deck edge has no corner at the I^{th} offset station.

1, if the deck edge has a corner at this station but the angle of the corner is greater than a right angle.

2, if the deck edge has a step this station.

DCKPNT: An integer array of length 2×NSTMAX. DCKPNT gives the positions in the array OFFPNT of the points on the deck edge; thus, OFFPNT(DCKPNT(J,I),I,1) and OFFPNT(DCKPNT(J,I),I,2) are the Y and Z values of the J^{th} deck edge point on the I^{th} station. J will normally only take the value 1; only if DCKTYP(I) is 2 will there be two deck edge points on a station. In this case, the point DCKPNT(1,I) is on the same deck segment as the point marked for the previous station; the point DCKPNT(2,I) is on the same deck segment as the point marked for the following station.

NDCMAX: A parameter which specifies the maximum number of deck edge corners allowed; the current value is five. NDCMAX is necessary for setting the dimensions of some of the following arrays.

NDKCRN: The number of corners and steps in the deck edge. NDKCRN must be in the range [1,NDCMAX].

DKCRST: A real array of length NDCMAX which contains the station numbers of the corners in the deck edge.

DKCRTP: An integer array of length NDCMAX which indicates the type of each corner in the deck edge; the elements of DKCRTP correspond to the elements of DKCRTP and its values are the same as used in DCKTYP: i.e. DKCRTP(I) is 1 if the I^{th} corner in the deck has an angle greater than 90°; it is 2 if the I^{th} corner in the deck is a step.

These variables are made available globally via the common block /DCKDAT/ which has the following form. Notice that /DCKDAT/ should be saved (using the Fortran 77 SAVE statement) so that the values of the variables it contains will be preserved.

```
      PARAMETER (NSTMAX=50, NDCMAX=5)
      INTEGER DCKTYP, DCKPNT, DKCRTP, NDKCRN
      LOGICAL DCKFLG
      REAL DKCRST
      COMMON /DCKDAT/ DCKFLG, DCKTYP(NSTMAX), DCKPNT(2,NSTMAX), NDKCRN,
     +                DKCRST(NDCMAX), DKCRTP(NDCMAX)
      SAVE /DCKDAT/
```

B.2.6 Logical Function FRMMRK

Purpose: FRMMRK performs the complete conversion of the array MARKS to the alternative data structures associated with the hull features. It does so by calling procedures to perform the conversion for each separate hull feature.

Input via Common Block /MRKPNT/: The character array MARKS.

Returned Value: FRMMRK returns true if the conversion is made without error, false otherwise.

Output via Common Blocks: Values are set for the variables in the common blocks /KNKDAT/, /BKLDAT/, /WTRDAT/, /HFSDAT/, and /DCKDAT/ described in the Appendices B.2.1–B.2.5.

Output Arguments:

IER: An integer error flag which will be zero if no error occurred, non-zero otherwise. See Appendix D for a list of possible values for IER.

Calling Sequence: FRMMRK may be called as follows.

```
INTEGER IER
LOGICAL FRMMRK

IF (FRMMRK(IER)) THEN
   <Continue normally>
ELSE
   <An error occurred>
END IF
```

B.2.7 Logical Function TOMRK

Purpose: TOMRK performs the complete conversion of the data structures associated with the hull features (i.e. knuckles, bilge keels, etc.) to the array MARKS. It does so by calling procedures to perform the conversion for each separate hull feature.

Input via Common Blocks: Data is obtained from the common blocks /KNKDAT/, /BKLDAT/, /WTRDAT/, /HFSDAT/, and /DCKDAT/ described in the Appendices B.2.1-B.2.5.

Returned Value: TOMRK returns true if the conversion is made without error, false otherwise.

Output Arguments:

IER: An integer error flag which will be zero if no error occurred, non-zero otherwise. See Appendix D for a list of possible values for IER.

Output via Common Block /MRKPNT/: The character array MARKS.

Calling Sequence: TOMRK may be called as follows.

```
INTEGER IER
LOGICAL TOMRK

IF (TOMRK(IER)) THEN
   <Continue normally>
ELSE
   <An error occurred>
END IF
```

B.3 NAME Record Data Structures

The NAME record contains the name of the hull. The name is stored in the character variable HLLNAM of length 60; a name with more than 60 characters will be truncated. HLLNAM is stored in the common block /NAMREC/. Notice that /NAMREC/ should be saved (using the Fortran 77 SAVE statement) so that the value of the HLLNAM will be preserved.

```
CHARACTER HLLNAM*60
COMMON /NAMREC/ HLLNAM
SAVE /NAMREC/
```

The sub-program GHNAME is provided as a convenient alternative for obtaining the hull name after the input file has been read. It returns the hull name in a character variable via its argument list.

B.3.1 Logical Function GHNAME

Purpose: GHNAME returns the hull name to the calling program; this is an alternative to using the common block /NAMREC/.

Returned Value: GHNAME returns true if the NAME record is defined (i.e. if HLLNAM is not blank); false otherwise.

Output Argument:

UHNAME: A character variable in which the name of the hull is returned. UHNAME can be of any length but the hull name may be truncated if UHNAME has fewer than 60 characters.

Calling Sequence: GHNAME may be called as follows.

```
CHARACTER UHNAME*60
LOGICAL DEFIND, GHNAME

DEFIND=GHNAME(UHNAME)
```

B.4 DIMENSIONS Record Data Structures

The DIMENSIONS record contains the dimensions of the hull (the length between perpendiculars, the half-breadth, and the depth) along with the units in which the dimensions are given. The following data structures are used to describe the hull dimensions.

LHDIM: A logical flag which indicates whether the DIMENSIONS variables have been defined.

HUNITS: A character variables which contains the units in which the hull dimensions are given. The string describing the units may be up to ten characters long; longer strings will be truncated.

LBP: A real variable which contains the length of the hull between perpendiculars.

HFBDTH: A real variable which contains the half-breadth of the hull.

DPTH: A real variable which contains the depth of the hull.

These variables are made available globally via the common blocks /DIMREC/ and /UNTREC/ which have the following form.

```
CHARACTER HUNITS*10
LOGICAL LHDIM
REAL LBP, HFBDTH, DPTH
COMMON /DIMREC/ LHDIM, LBP, HFBDTH, DPTH
COMMON /UNTREC/ HUNITS
SAVE /DIMREC/, /UNTREC/
```

Two common blocks are required as real and character variables cannot share the same common block. Notice that /DIMREC/ and /UNTREC/ should be saved (using the Fortran 77 SAVE statement) so that the values of the variables they contain will be preserved.

The following sub-programs are provided as alternatives methods for obtaining the values of the hull dimensions and their units; they should only be called after a file has been read by RDFLRC.

GHDIM: returns the hull dimensions to the calling program; this is an alternative to using the common block /DIMREC/.

GHUNIT: returns the hull units to the calling program; this is an alternative to using the common block /UNTREC/.

B.4.1 Logical Function GHDIM

Purpose: GHDIM returns the hull dimensions to the calling procedure via its argument list.

Returned Value: GHDIM returns true if a DIMENSIONS record is defined; false otherwise. In the latter case an error message is sent to the terminal the GETWRD Package[1] sub-program WRTERM.

Output Arguments:

ULBP: A real variable in which the length of the hull between perpendiculars is returned.

UHFBDT: A real variable in which the half-breadth of the hull is returned.

UDPTH: A real variable in which the depth of the hull is returned.

Calling Sequence: GHDIM may be called as follows.

```
REAL ULBP, UHFBDT, UDPTH
LOGICAL DEFIND, GHDIM

DEFIND=GHDIM(ULBP,UHFBDT,UDPTH)
```

B.4.2 Logical Function GHUNIT

Purpose: GHUNIT returns the units of hull dimensions to the calling procedure via its argument list.

Returned Value: GHUNIT returns true if a DIMENSIONS record is defined; false otherwise. In the latter case an error message is sent to the terminal via the GETWRD Package sub-program WRTERM.

Output Arguments:

UHUNIT: A character variable in which the units of the hull dimensions are returned.

Calling Sequence: GHUNIT may be called as follows.

```
CHARACTER UHUNIT
LOGICAL DEFIND, GHUNIT

DEFIND=GHUNIT(UHUNIT)
```

B.5 PERPENDICULAR STATIONS Record Data Structures

The PERPENDICULAR STATIONS record contains the station numbers of the forward and aft perpendiculars. The following data structures are used to describe these station numbers.

XFP: The station number of the forward perpendicular.

XAP: The station number of the aft perpendicular.

These variables are made available globally via the common blocks /PSTREC/ which has the following form.

```
REAL XFP, XAP
COMMON /PSTREC/ XFP, XAP
SAVE /PSTREC/
```

The default value for the forward perpendicular is 0 and of the the aft perpendicular is 20. Notice that /PSTREC/ should be saved (using the Fortran 77 SAVE statement) so that the values of XFP and XAP will be preserved.

The logical function GPERP is provided as an alternative method for obtaining the values of the station numbers of the perpendiculars. It returns the values of the station number of the perpendiculars via its argument list. Its returned value is true if the station numbers have the default values of 0 and 20, false otherwise. GPERP should only be called after a file has been read by RDFLRC.

B.5.1 Logical Function GPERP

Purpose: GPERP returns the station numbers of the forward and aft perpendiculars to the calling procedure via its argument list.

Returned Value: GPERP returns true if the station numbers have the default values of 0.0 and 20.0, false otherwise.

Output Arguments:

UXFP: A real variable in which the station number of the forward perpendicular is returned.

UXAP: A real variable in which the station number of the aft perpendicular is returned.

Calling Sequence: GPERP may be called as follows.

```
REAL UXFP, UXAP
LOGICAL DEFALT, GPERP

DEFALT=GPERP(UXFP,UXAP)
```

B.6 BOW PROFILE OFFSETS Record Data Structures

The BOW PROFILE OFFSETS record contains offset points on the bow profile. The following data structures are used to describe these points.

NBPOMX: A parameter which specifies the maximum number of offsets allowed on the bow profile; the current value is 50. NBPOMX is used to set the dimensions for the array BPOPNT.

NBPFO: The number of offset points on the bow profile.

BPOPNT: A real array of dimensions NBPOMX×2. The Y and Z values of the I^{th} offset point on the bow profile are stored in BPOPNT(I,1) and BPOPNT(I,2) respectively. The bow profile offset points are ordered from keel to the deck edge (waterline) as I increases from 1 to NBPFO.

These variables are made available globally via the common block /BPOREC/ which has the following form. Notice that /BPOREC/ should be saved (using the Fortran 77 SAVE statement) so that the values of the variables it contains will be preserved.

```
PARAMETER (NBPOMX=50)
INTEGER NBPFO
REAL BPOPNT
COMMON /BPOREC/ NBPFO, BPOPNT(NBPOMX,2)
SAVE /BPOREC/
```

B.7 STERN PROFILE OFFSETS Record Data Structures

The STERN PROFILE OFFSETS record contains offset points on the stern profile. The following data structures are used to describe these points.

NSPOMX: A parameter which specifies the maximum number of offsets allowed on the stern profile: the current value is 50. NSPOMX is used to set the dimensions of the array SPOPNT.

NSPFO: The number of offset points on the stern profile.

SPOPNT: A real array of dimensions NSPOMX×2. The Y and Z values of the I^{th} offset point on the stern profile are stored in SPOPNT(I,1) and SPOPNT(I,2) respectively. The stern profile offset points are ordered from keel to the deck edge (waterline) as I increases from 1 to NSPFO.

These variables are made available globally via the common block /SPOREC/ which has the following form. Notice that /SPOREC/ should be saved (using the Fortran 77 SAVE statement) so that the values of the variables it contains will be preserved.

```
PARAMETER (NSPOMX=50)
INTEGER NSPFO
REAL SPOPNT
COMMON /SPOREC/ NSPFO, SPOPNT(NSPOMX,2)
SAVE /SPOREC/
```

B.8 TRANSOM Record Data Structures

The TRANSOM record contains data describing the position and orientation of the transom; in addition, it has a sub-record, TRANSOM OFFSETS, which contains offset points lying on the transom. The following data structures are used to describe the transom.

LTRAN: A logical flag which is true if a transom is included in the hull representation, false if not.

XTRAN: A real variable containing the value of X_t (see Section 3.8).

ZTRAN: A real variable containing the value of Z_t.

NRMX: A real variable containing the value of n_X.

NRMZ: A real variable containing the value of n_Z.

NTRNMX: A parameter which specifies the maximum number of offsets allowed on the transom; the current value is 50. NTRNMX is used to set the dimensions of the array TRNPNT.

NTRNO: The number of offset points on the transom.

TRNPNT: A real array of dimensions NTRNMX×3. The X, Y and Z values of the I^{th} offset point on the transom are stored in the elements TRNPNT(I,1), TRNPNT(I,2), and TRNPNT(I,3) respectively. The transom offset points are ordered from keel to the deck edge (waterline) as I increases from 1 to NTRNO.

These variables are made available globally via the common block /TRNREC/ which has the following form. Notice that /TRNREC/ should be saved (using the Fortran 77 SAVE statement) so that the values of the variables it contains will be preserved.

```
      PARAMETER (NTRNMX=50)
      INTEGER NTRNO
      LOGICAL LTRAN
      REAL XTRAN, ZTRAN, NRMX, NRMZ, TRNPNT
      COMMON /TRNREC/ LTRAN, XTRAN, ZTRAN, NRMX, NRMZ, NTRNO,
     +                TRNPNT(NTRNMX,3)
      SAVE /TRNREC/
```

B.9 Reading and Writing Offset Files

The logical function RDFLRC is used to read an offset file. If an optional record is not found in the input file, the data structures which it governs are left unaltered by RDFLRC; i.e. RDFLRC uses as default values those values which the data structures had when RDFLRC was called. As described in Section 2.2.2, this is often not adequate if more than one file is to be read. Hence, the sub-program SDFLT is provided to set defaults for all the standard OFFSRF records. SDFLT should be called prior to each call to RDFLRC. SDFLT is described in Appendix B.10.

When RDFLRC returns, the data describing the point marks are stored in the character array MARKS. These data can be re-interpreted and stored in the data structures associated with the knuckles, bilge keels, waterline, etc.; this is done by the logical function FRMMRK. The second set of data structures is usually the more convenient to use. Hence, a common calling sequence for reading offset files is

```
CALL SDFLT
IF (.NOT.RDFLRC(IUNIT,RECRDS,IER)) THEN
   <An error occurred>
ELSE IF (.NOT.FRMMRK(IER)) THEN
   <An error occurred>
ELSE
   <Continue normally>
END IF
```

The sub-program WRFLRC is also provided with OFFSRF; it examines all the data structures associated with the offset records and writes an offset file. WRFLRC uses the point marks in the array MARKS when writing the POINTS sub-records. Hence, before calling WRFLRC, one will want to convert from the data structures governing the knuckles, bilge keels, etc., back to MARKS; this may be done by the sub-program TOMRK. Hence, a common calling sequence for writing offset files is

```
IF (.NOT.TOMRK(IER)) THEN
   <An error occurred>
ELSE IF (.NOT.WRFLRC(IUNIT,IER)) THEN
   <An error occurred>
ELSE
   <Continue normally>
END IF
```

A program main-program which reads or writes an OFFSRF data file using RDFLRC or WRFLRC can be compiled and executed on the DREA DEC-20/60 as follows.

```
DEFINE H: PS:<HALLY>
EXECUTE main-program,H:RDFLRC,H:OFFSRF/LIB,H:GETWRD/LIB
```

B.9.1 Logical Function WRFLRC

Purpose: WRFLRC writes all defined OFFSRF offset data into an ASCII file using the record structure described in Section 2. It is assumed that the file has already been opened and assigned a logical unit number.

WRFLRC makes use of the GETWRD Package[1] of sub-programs which are used to handle all output to the terminal. On the DREA DEC-20/60 the GETWRD Package may be found in PS:<HALLY>GETWRD.FOR.

Input Arguments:

IUNIT: The unit number assigned to the file.

Returned Value: WRFLRC returns true if the file is written without error, false otherwise.

Output Arguments:

IER: An integer error flag which will be zero if no error occurred, non-zero otherwise. See Appendix D for a list of possible values for IER.

Output to the terminal: WRFLRC writes error messages on the terminal whenever an error occurs. All terminal output is handled using the GETWRD Package sub-program WRTERM.

Calling Sequence: WRFLRC may be called as follows.

```
INTEGER IUNIT, IER
LOGICAL WRFLRC

IF (WRFLRC(IUNIT,IER)) THEN
   <Everything is OK: continue normally>
ELSE
   <An error occurred>
END IF
```

B.10 SDFLT Programmer's Guide

Purpose: The subroutine SDFLT is called to set default values for all standard OFFSRF records. If non-standard optional records are to be contained in the input file, the programmer should ensure that they are also given default values before RDFLRC is called.

Arguments: none.

Calling Sequence: SDFLT can be called as follows.

```
      CALL SDFLT
```

The following is the Fortran code for the sub-program SDFLT as used in OFFSRF. It can be used by programmers as a template when extending OFFSRF to include new records or sub-records. For each record for which defaults must be set, an appropriate sub-program (in this case they are all logical functions) is called to set the defaults.

```
      SUBROUTINE SDFLT
C-----------------------------------------------------------------------C
C                                                                       C
C Sets default values for all records by calling their associated       C
C default-setting procedures.                                           C
C                                                                       C
C-----------------------------------------------------------------------C
      LOGICAL LDUM, SBPFO, SHDIM, SHNAME, SPERP, SSPFO, SSTAT, STRAN

C Set defaults for BOW PROFILE OFFSETS record.
      LDUM=SBPFO()

C Set defaults for DIMENSIONS record.
      LDUM=SHDIM()

C Set defaults for NAME record.
      LDUM=SHNAME()

C Set defaults for PERPENDICULAR STATIONS record.
      LDUM=SPERP()

C Set defaults for STERN PROFILE OFFSETS record.
      LDUM=SSPFO()

C Set defaults for STATION record.
      LDUM=SSTAT()
```

```
C Set defaults for TRANSOM record.
      LDUM=STRAN()
      RETURN
      END
```

B.11 RDREC as used in OFFSRF

The following is the Fortran code for the sub-program RDREC as used in OFFSRF. It can be used by programmers as a template when extending OFFSRF to include new records or sub-records. For each record or sub-record, an appropriate sub-program is called which reads in the data for that record.

```
      LOGICAL FUNCTION RDREC(IUNIT,RECLAB,FRSTLN,IER)
C-----------------------------------------------------------------------C
C                                                                       C
C RDREC reads the contents of all records which contain data and not    C
C sub-records.                                                          C
C                                                                       C
C Input Arguments:                                                      C
C  IUNIT = The unit number of the input file.                           C
C  RECLAB= A character variable containing the name of the record to    C
C          be read.                                                     C
C                                                                       C
C Output Argument:                                                      C
C  IER = An integer error flag.                                         C
C                                                                       C
C Returned Value:                                                       C
C  True if the record is read without error.                            C
C  False if an error occurred.                                          C
C                                                                       C
C-----------------------------------------------------------------------C
      CHARACTER RECLAB*(*), FRSTLN*(*)
      INTEGER IUNIT, IER
      LOGICAL INCNST, RDBPFO, RDHDIM, RDHNAM, RDNMBR, RDPERP,
     + RDPNTS, RDSPFO, RDTRAN, RDTRNO, LDUM

C Read data from BOW PROFILE OFFSETS record.
      IF (RECLAB.EQ.'BOW PROFILE OFFSETS') THEN
         LDUM=RDBPFO(IUNIT,IER)

C Read data from DIMENSIONS record.
      ELSE IF (RECLAB.EQ.'DIMENSIONS') THEN
         LDUM=RDHDIM(IUNIT,FRSTLN,IER)

C Read data from NAME record.
      ELSE IF (RECLAB.EQ.'NAME') THEN
         LDUM=RDHNAM(IUNIT,FRSTLN,IER)

C Read data from NUMBER record.
      ELSE IF (RECLAB.EQ.'NUMBER') THEN
```

```
         LDUM=RDNMBR(IUNIT,FRSTLN,IER)

C Read data from PERPENDICULAR STATIONS record.
      ELSE IF (RECLAB.EQ.'PERPENDICULAR STATIONS') THEN
         LDUM=RDPERP(IUNIT,FRSTLN,IER)

C Read data from POINTS sub-record.
      ELSE IF (RECLAB.EQ.'POINTS') THEN
         LDUM=RDPNTS(IUNIT,IER)

C Increment the number of stations, NST.
      ELSE IF (RECLAB.EQ.'STATION') THEN
         LDUM=INCNST(IER)

C Read data from STERN PROFILE OFFSETS record.
      IF (RECLAB.EQ.'STERN PROFILE OFFSETS') THEN
         LDUM=RDSPFO(IUNIT,IER)

C Read data from TRANSOM record.
      ELSE IF (RECLAB.EQ.'TRANSOM') THEN
         LDUM=RDTRAN(IUNIT,FRSTLN,IER)

C Read data from TRANSOM OFFSETS sub-record.
      ELSE IF (RECLAB.EQ.'TRANSOM OFFSETS') THEN
         LDUM=RDTRNO(IUNIT,FRSTLN,IER)

C Check to see whether the record contains data; an error message is
C written if it does.
      ELSE
         CALL CHKREC(IUNIT,RECLAB,FRSTLN,IER)
      END IF
      RDREC=IER.EQ.0
      RETURN
      END
```

Appendix C Output to the Terminal

All output to the terminal by the both the RDFLRC and OFFSRF sub-programs is handled using the GETWRD Package[1] message buffering facility. This offers the programmer several features for controlling the output of messages to the user at the terminal.

C.1 Limiting Columns of Output

The portion of the terminal screen used for writing messages can be restricted by the following call.

```
CALL SETCOL(ILO,IHI)
```

Messages will then be restricted to lie between columns ILO and IHI. If a message is too long to fit between these columns, it is broken at the first appropriate space. For example, the message

```
This is a message which is too long to fit between columns 10 and 30.
```

would appear as

```
This is a message
which is too long to
fit between columns
10 and 30.
```

after the call

```
CALL SETCOL(10,30)
```

The default columns are 1 to 80. If a long line cannot be broken, it will be allowed to continue past the high column. For example, with the output restricted between columns 1 and 20, the message

```
This-is-a-long-message-with-no-spaces.
```

will appear as such on the terminal even though it extends past column 20.

Restricting the width of the terminal output is particularly useful in graphics programs: one portion of the screen can be allocated for graphic displays and another for interaction with the user.

The current values of the terminal display columns can be obtained by calling GETCOL.

```
CALL GETCOL(ILO,IHI)
```

C.2 Temporary Suppression of Terminal Output

Another feature which is also useful in graphics programs is the buffering of the terminal output; that is, all messages sent to the terminal via the GETWRD Package sub-programs can be suppressed temporarily, then written later. The call

```
CALL SETHLD(.TRUE.)
```

causes all messages to the terminal to be suppressed temporarily; they are stored in a buffer. The subsequent call

```
CALL SETHLD(.FALSE.)
```

causes all stored messages to be written to the terminal. The state of the message suppression may be determined using the call

```
CALL GETHLD(HLDBUF)
```

where HLDBUF is a logical variable. Its value will be true if terminal messages are currently being suppressed, false if they are not.

Message suppression is useful when errors occur while creating graphic displays. Messages are suppressed before the display begins. When an error occurs, a message can be sent to the terminal immediately. The terminal can then be cleared or taken out of graphics mode before the error message is written by dumping the message buffer.

Appendix D OFFSRF Errors

When an error occurs while reading or writing an OFFSRF file, the integer error flag IER is set non-zero. The following is a list of the possible errors which can occur and the corresponding values of IER.

Errors relating to the record structure:

0 No error.

100 The record structure is too highly nested; only 10 levels of nesting are allowed.

101 Data was found in a file when a record label was expected. This is normally caused by superfluous data in a record.

102 A record label was empty: i.e. a line beginning with { had no record label.

103 The end-of-file was encountered before all records had been terminated.

104 There is an error in the format of the list of records, RECRDS. This is a programming error, not a user error.

105 A required record was not found in the file.

108 A required or optional record containing data was found in the file, but there is no sub-program assigned to read the data from it. This is a programming error, not a user error.

109 An error occurred when repositioning the position marker in the input file to reread a line. This error should never occur.

110 A read error occurred when reading a line from the file.

111 The end of a record was not correctly matched with its beginning.

Errors relating to the DIMENSIONS record:

201 An error occurred when reading the units for the hull dimensions from a file.

202 An error occurred when reading the hull dimensions from a file.

203 The end of the file was encountered before all data for the DIMENSIONS record had been read.

204 One of the hull dimensions was non-positive.

205 An error occurred when writing the DIMENSIONS record into a file.

Errors relating to the PERPENDICULAR STATIONS record:

401 An error occurred when reading the station numbers of the perpendiculars from a file.

402 The end of file was encountered before the station numbers of both perpendiculars were read.

403 The station number of the forward perpendicular is the same as the station number of the aft perpendicular.

450 An error occurred when writing the PERPENDICULAR STATIONS record into a file.

Errors relating to the NAME record:

501 An error occurred when writing the hull name record into a file.

Errors relating to the knuckles:

601 The number of knuckles was negative or exceeded NKNMAX.

602 Two different points on a single station were marked as lying on a knuckle.

603 A point was marked as lying on two knuckles.

604 There is a knuckle for which no points were marked. Note that this does not mean that there will be an error if there are no knuckles; rather it means that, for example, that some points have been marked as lying on knuckle #2 but none have been marked for knuckle #1.

605 For one of the knuckles, not every station between the start and end of the knuckle was marked.

606 A point is marked on a station outside of the range of the knuckle on which it lies: i.e. for some IKNK in [1,NKNUK] and some IST in [1,NST], KNKPNT(IST,IKNK) is non-zero but STATNO(IST) does not lie between STAKNK(IKNK) and ENDKNK(IKNK).

607 The value of KNKPNT(IST,IKNK) was outside the range [1,NPPST(IST)] for some IKNK in [1,NKNUK] and some IST in [1,NST].

608 There are too many point marks at a single point; the elements of MARKS are not long enough to hold all the marks.

Errors relating to the waterline:

802 Two different points on a single station were marked as lying on the waterline.

803 A point was twice marked as lying on the waterline.

805 Not every station between the start and end of the waterline was marked.

806 A point is marked outside of the range of the waterline: i.e. for some IST in [1,NST], WTRPNT(IST) is non-zero but STATNO(IST) does not lie between STAWTR and ENDWTR.

807 The value of WTRPNT(IST) was outside the range [1,NPPST(IST)] for some IST in [1,NST].

808 There are too many point marks at a single point; the elements of MARKS are not long enough to hold all the marks.

Errors relating to the deck edge:

901 There is no corner point or step between different segments of the deck edge: i.e. the marks changed from 'd' to 'D' (or *vice versa*) on successive stations but there was no point marked 'dD'.

902 Two different points on a single station were marked as lying on the same segment of the deck edge: i.e. both points were marked with 'd' or both were marked with 'D'.

903 A point was twice marked as lying on the deck edge.

904 The number of corners in the deck edge exceeded NDCMAX.

905 Not every station between the start and end of the deck edge was marked.

906 A point is marked outside of the range of the deck edge: i.e. for some IST in [1,NST], DCKPNT(IST) is non-zero but STATNO(IST) does not lie between STADCK and ENDDCK.

907 The value of DDCKPNT(IST) was outside the range [1,NPPST(IST)] for some IST in [1,NST].

908 There are too many point marks at a single point; the elements of MARKS are not long enough to hold all the marks.

Errors relating to the TRANSOM record:

1201 An error occurred when reading the TRANSOM record data from the file.

1202 The end of the input file was encountered before all the data required for the TRANSOM record was read.

1203 The X component of the transom normal is zero.

1204 An error occurred when writing the TRANSOM record into the output file.

1205 The number of transom offset points is not in the range [1,NTRNMX].

1206 An error occurred when reading the X value of an offset point on the transom.

1207 An error occurred when reading the Y value of an offset point on the stern profile.

1208 An error occurred when reading the Z value of an offset point on the stern profile.

1209 There are no points in the TRANSOM OFFSETS sub-record.

1210 Two adjacent points in the TRANSOM OFFSETS sub-record are the same.

1250 An error occurred when writing the transom offset points into a file.

Errors relating to the bilge keels:

1601 The number of bilge keels was negative or exceeded NKNMAX.

1602 Two different points on a single station were marked as lying on a single bilge keel.

1603 A point was marked as lying on two bilge keels.

1604 There is a bilge keel for which no points were marked. Note that this does not mean that there will be an error if there are no bilge keels; rather it means that, for example, that some points have been marked as lying on bilge keel #2 but none have been marked for bilge keel #1.

1605 For one of the bilge keels, not every station between the start and end of the bilge keel was marked.

1606 A point is marked on a station outside of the range of the bilge keel on which it lies: i.e. for some IBKL in [1,NBLGKL] and some IST in [1,NST], BKLPNT(IST,IBKL) is non-zero but STATNO(IST) *does not lie between* STABKL(IBKL) and ENDBKL(IBKL).

1607 The value of BKLPNT(IST,IBKL) was outside the range [1,NPPST(IST)] for some IBKL in [1,NBLGKL] and some IST in [1,NST].

1608 There are too many point marks at a single point; the elements of MARKS are not long enough to hold all the marks.

Errors relating to the half-siding:

1702 Two different points on a single station were marked as lying on the half-siding.

1703 A point was twice marked as lying on the half-siding.

1705 Not every station between the start and end of the half-siding was marked.

1706 A point is marked outside of the range of the half-siding: i.e. for some IST in [1,NST], HFSPNT(IST) is non-zero but STATNO(IST) does not lie between STAHFS and ENDHFS.

1707 The value of HFSPNT(IST) was outside the range [1,NPPST(IST)] for some IST in [1,NST].

1708 There are too many point marks at a single point; the elements of MARKS are not long enough to hold all the marks.

Errors relating to the STATION record:

6001 There are too many offset stations: i.e. NST will exceed NSTMAX.

6002 A NUMBER or POINTS was found before a STATION record.

6003 The end of file was found when trying to read the number of a station.

6004 An error occurred when reading the station number from the file.

6005 The station numbers are not in increasing order.

6006 There are too many points on a station: i.e. NPPST(I) will exceed NPPSMX.

6007 An error occurred when reading the Y value of an offset point in a POINTS sub-record.

6008 An error occurred when reading the Z value of an offset point in a POINTS sub-record.

6009 There are too many point marks for a point in a POINTS sub-record: the total number of characters in the marks exceeds MAXMRK.

6010 There is a POINTS sub-record with no points.

6011 Two adjacent points in a POINTS sub-record were the same.

6050 An attempt was made to write data for a station which is not defined.

6051 An error occurred when writing data for a STATION record or one of its NUMBER or POINTS sub-records. sub-record into a file.

Errors relating to the BOW PROFILE OFFSETS record:

6101 The number of bow profile offset points is not in the range [1,NBPOMX].

6102 An error occurred when reading the X-value of an offset point on the bow profile.

6103 An error occurred when reading the Z-value of an offset point on the bow profile.

6104 There are no points in the BOW PROFILE OFFSETS record.

6105 Two adjacent points in the BOW PROFILE OFFSETS record are the same.

6150 An error occurred when writing the bow profile offset points into a file.

Errors relating to the STERN PROFILE OFFSETS record:

6201 The number of stern profile offset points is not in the range '1,NSPOMX'.

6202 An error occurred when reading the X-value of an offset point on the stern profile.

6203 An error occurred when reading the Z-value of an offset point on the stern profile.

6204 There are no points in the STERN PROFILE OFFSETS record.

6205 Two adjacent points in the STERN PROFILE OFFSETS record are the same.

6250 An error occurred when writing the stern profile offset points into a file.

References

1. D. Hally, "The GETWRD Package: Version 4.0," DREA Tech. Comm., in preparation.

Index

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM

(highest classification of Title, Abstract, Keywords)

DOCUMENT CONTROL DATA

Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified

1 ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractors report, or tasking agency, are entered in section 8.)

Defence Research Establishment Atlantic

2 SECURITY CLASSIFICATION (overall security classification of the document, including special warning terms if applicable)

UNCLASSIFIED

3 TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C,R or U) in parentheses after the title.)

OFFSRF: A System for Representing Ship Offset Data

4 AUTHORS (Last name, first name, middle initial. If military, show rank, e.g. Doe, Maj. John E.)

Hally, David

5 DATE OF PUBLICATION (month and year of publication of document)

June 1989

6a NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.)

72

6b NO. OF REFS (total cited in document)

1

7 DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

DREA Technical Communication

8 SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)

Defence Research Establishment Atlantic
P.O. Box 1012, Dartmouth, N.S., B2Y 3Z7

9a PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)

1AF

9b CONTRACT NO. (if appropriate, the applicable number under which the document was written)

10a ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DREA Technical Communication 89/305

10b OTHER DOCUMENT NOS. (any other numbers which may be assigned this document either by the originator or by the sponsor)

11 DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify)

12 DOCUMENT ANNOUNCEMENT (any limitations to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM

DCD03 2/81

Unclassified

SECURITY CLASSIFICATION OF FORM

13 ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual)

It is argued that a standard format for representation of offset data in computer files would save time for both users and programmers of hydrodynamic programs. A system suitable for adoption as a standard is described. It includes standard formats for offset data on stations, on bow and stern profiles, and on transoms, as well as formats for common hull features such as knuckles, bilge keels, the design waterline, the deck edge, and the half-siding. The format is extensible so that additional data may also be included in the offset files.

To ease the burden on the programmer, Fortran sub-programs to read the standard data are provided; they provide extensive error checking and accurate pin-pointing of the location of errors in the file. Programmer's manuals for the sub-programs are included.

14 KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Ship hulls
Computer programs
Hydrodynamics

Unclassified

SECURITY CLASSIFICATION OF FORM