The C3D data file format was developed by Andrew Dainis in 1987 as a convenient and efficient means for storing 3D coordinate and analog data, with all associated parameters, for a single measurement trial.

One design goal of the C3D file format was to provide the user a means to easily examine, and if necessary modify, any parameter contained in the file. This goal was achieved by providing the user with a program (PRM) which interactively allows access to the parameters stored internaly in binary format.

TABLE OF CONTENTS

1. Parameter Manipulation Program PRM
2. C3D Data File Format
3. Parameter Access From Programs

1. PARAMETER MANIPULATION PROGRAM PRM

1.1 INTRODUCTION

The PRM program provides the user a means of accessing the parameters in C3D and other ADTECH parameter files. It operates interactively through a few simple commands and provides output to terminal screen, printer, or file. The program allows the user to create, examine, change, and delete parameters in any file that contains data in the ADTECH parameter format, and acts as a friendly interface between the user and the parameters.

1.2 PARAMETER FORMAT

This section describes the format of the parameters as displayed by the PRM program. Internally to a file the parameters are stored in binary format and may be directly accessed using the techniques described in sections 2. and 3.

Each parameter is identified by a name, and has a data type. A parameter also has dimensions which describe how many pieces or elements of data it can hold, and optionally, a description. Additionally each parameter belongs to a parameter group having a group name and a group description. In listings and commands the group name and parameter name are separated by a colon (:). By a suitable choice of group and parameter names, and through the descriptions, it is possible to make the parameter functions largely self-explanatory. The ability to group parameters enables parameters pertaining to different functions to be included in the same file without risk of confusion.

A parameter or group name may consist of any number of characters made up from the letters A through Z, the numerals 0 through 9, and the underscore character _ . The name should not start with a numeral or the underscore character. When a name is interpreted only the first six characters are used. Hence all group names and all names within a group should show at least one difference in the first six characters. The same names may be used for two parameters if they occur in different groups.

A parameter's type determines the type of data that may be stored in it. Four parameter types are used; integer, real, character, and byte. These data types correspond to the conventional meaning of the terms in computer programming. An integer is a 16-bit signed number between -32768 and +32767, a real number is one containing a decimal point or written in scientific exponential representation, a character is a literal symbol such as a letter entered from the keyboard, and a byte data location can contain a 8-bit signed integer in the range -128 to +127.

The dimensions of a parameter define how many elements of the appropriate type may be stored in that parameter. The use of the term dimensions also follows normal programming conventions. If a parameter has no dimensions, then it may only hold one value of its data type. If it has one dimension it is presented in the form such as PARM(4) where the 4 indicates that the parameter called PARM is capable of holding four values. Examples of two- and three-dimensional parameter arrays are PARMA(4,5) and PARMB(3,5,7). The first example has 4 x 5 = 20 elements, and the second parameter holds 3 x 5 x 7 = 105 entries. The content of any parameter element may be selectively examined and modified by use of the PRM program.

The description part of a group or parameter definition simply provides information as to the function of the particular group or parameter, and is included for informational purposes only.

1.3 RUNNING THE PROGRAM

1.3.1 Starting the PRM Program

The Parameter Manipulation Program is started by entering

```
PRM  <file name>
```

from the keyboard in response to the operating system prompt. The screen will clear, the program title will be displayed, and the requested file will be opened. If no file name is provided, the program will request the input of a file name.

1.3.2 Accessing the Parameters

If the named file is found, it is opened and the program first tests to determine that the file does contain parameter data. If the format is valid, the data are read into memory and become available for manipulation. If the file does not contain parameter data a message to that effect is output to the terminal screen and the user is prompted for another file name. If the named file cannot be opened for any reason, a new file is opened in preparation for creating a new set of parameters and a new file message is displayed on the terminal screen. The user then may proceed to create parameter groups and parameters as described in the remainder of this chapter. Once the program is ready to accept parameter manipulation commands, the prompt symbol ? will be displayed. If a mistake was made in entering the file name and the user does not wish to create a new file, the abort command AB(ort) should be entered in response to the ? prompt. In this circumstance the program will ask for another file name.

When an existing file is opened, the program first lists the existing parameter groups before displaying the ? prompt.

1.3.3 Terminating the Program

Either of the commands ABort or EXit will terminate the PRM program. The ABort command will cause the program to exit without making any changes to the input parameter file, whereas the EXit command will cause the input file to be updated with the changes entered during this session.

1.4 COMMAND DESCRIPTION

The PRM program responds to eight commands: CREATE, DELETE, LIST, PRINT, WRITE, CHANGE, ABORT, and EXIT. All commands except CREATE may be abbreviated to their first letter. The shortest abbreviation allowed for CREATE is CR. The CREATE and DELETE commands will be described first. However, the user will very rarely need to use these two commands since the modification of existing parameter files, or of copies of existing files, provides a more efficient technique for obtaining new files. The creation of a new parameter file must be done very carefully because errors in parameter names or dimensions will cause programs to reject the parameter data. Occasionally the CREATE and DELETE facilities may be useful for changing the dimensions or description of a parameter, or replacing an incorrectly created parameter.

In the following command descriptions, di refers to the ith dimension of a parameter (for instance, the 7 in CAM_CHAN (7)).

1.4.1 Create (CR)

1.4.1.1 Group

To create a new parameter group, the command is

```
CReate group_name:/description
```

The group called group_name is created, and with it is associated a description. If a group is being created, the ':' must be specified. Also, the slash character '/' must be used to separate the description from the rest of the command.

1.4.1.2 Parameter

To create a parameter within an existing group, use the command

```
CReate {group_name:}parameter {(d1, d2,...)},type/description
```

If a group name is not placed in front of a parameter name, the parameter will be created in the default group. The default group is the one whose name was last specified in any command. If a group name is included before the parameter name then the parameter is created in the specified group.

A parameter may have up to seven dimensions, but normally none, one, or two are used. If no dimension is defined then that parameter can only hold one element. The parameter type must be specified as a single character from the list of four; I, R, C, and B where I denotes Integer, R denotes Real, C denotes Character, and B denotes Byte. The slash / must precede the description part of the definition.

When a parameter is created, it is filled with default values. For numerical parameters (those of type real, integer or byte) the default value is zero. For the character type of parameter the default entry in all locations is the space ' ' character. Once a parameter or group is created, its name, dimensions, or description may only be changed by deletion and re-creation.

There are no restrictions on the length of group or parameter names, which must be made up from the letters A - Z, the numerals 0 - 9, and the underscore character _ . A name should not start with a numeral or _ . Only the first six characters of a name are referenced when a name is interpreted. Therefore no two names of parameters in the same group should be identical in all first six characters. Parameters may only have the same names if they belong to different groups.

1.4.2 Delete (D)

Groups and parameters may be deleted. A group may be deleted only after all parameters within it have been deleted. A parameter may be deleted if it is not locked by the program that created it. A locked parameter (identified by and asterisk (*) in front of its description) may also not be modified by the CHANGE command. The delete command has the format

```
Delete group_name:

     or

Delete group_name:parameter_name
```

Note that a group name must always be used when a parameter is being deleted. Also, any parameter or group name may be abbreviated, even down to one letter. However, if the abbreviation can refer to more than one name, action may be taken on the wrong parameter.

1.4.3 List (L), Write (W), or Print (P)

The only difference between these commands is the output device to which the information is sent. The List command results in output to the terminal from which the command was entered, the Write command writes the output to a file named LISTPAR.DAT, and the Print command additionally sends the LISTPAR.DAT file to the system printer. Note that data are written to the output file until either the EXit or ABort command is given, at which point the file is closed (and sent to the printer if the Print command was used).

1.4.3.1 Groups

A listing of all groups may be obtained by the command

```
L :
```

where the : symbolically denotes groups. The space between the command and the colon is required. The parameter names, dimensions and descriptions contained in any group may be listed by the command

```
L group_name:
```

and an L without any group name will list all of the parameter names in the

current default group. It should be noted that any time a group name is used, that group becomes the current default group. Also, when a group is listed the content of any parameter that has a single element is also automatically listed.

1.4.3.2 Parameters

This output commands has the general syntax:

```
List {group _name:}parameter_name {(d1 {, d2,..})}
```

If the group name is omitted, then the current default group name is used. A parameter name must be specified, and the dimensions are optional. Abbreviations for group and parameter file names may be used. If no dimensions are specified, the values of all elements of the parameter are listed. By use of entries within the dimension parentheses, any subset of elements of the parameter may be listed. As an example, consider a parameter that was created with the command

```
CR COORD(3,20),R/Reference point coordinates
```

The organization of this parameter may be thought of as consisting of 60 storage locations arranged in an array that has three rows and 20 columns. Each location can store a real (floating point) number. The command

```
L COORD
```

will cause the PRM program to list all values in COORD starting with the value stored in location (1,1), followed by (2,1), (3,1), (1,2), (2,2), (3,2), (1,3),.... , (3,19), (1,20), (2,20), (3,20), i.e., the first dimension is varied most rapidly, followed by the second dimension, and so on. The command

```
L CO(2,15)
```

will list the value stored in the second row and the 15th column. If a dimension is omitted, all the elements for that dimension will be listed, i.e., the command

```
L COORD(,15)
```

will list the elements (1,15), (2,15), and (3,15), whereas the command

```
L COORD(3,)
```

will display the values in (3,1), (3,2),...., (3,20). The command L COORD(,) is equivalent to L COORD . The above example illustrates how program PRM provides a simple and flexible facility for selecting parameter data for display or modification.

1.4.4 Change (C)

1.4.4.1 Groups

The Change command may be applied to an entire group as in

```
Change group_name:
```

to have the program prompt for the modification of all single element parameters contained in the group.

1.4.4.2 Parameters

In its most general form the change command has the format

Change {group_name:}parameter_name {(d1, {d2, ...})}

If a group name is not given, the current default group name is assumed. The rules for accessing the elements of a dimensioned parameter are the same as those for the LIST command, i.e., all, some, or one of the elements of a parameter may be selected for modification. After a change command is entered, each element is listed on the terminal with its current value. The user is then prompted for the entry of a new value. Pressing just the <ENTER> key leaves the value in that particular element unchanged. If the user wishes to exit from the change mode without completing the list specified in the change command, then the PRM prompt symbol ? should be entered.

Some parameters are locked by their creating programs in order to protect the data from modifications which may cause succeeding programs to function incorrectly. A locked parameter can be recognized by an asterisk preceding its description, and any attempt to change or delete it results in a message stating that it is locked.

The first dimension of a parameter which has a character type determines how long the lines are for entering characters. For example, a character parameter with dimensions (30,5) holds five lines, with each line having room for 30 characters. A command to change or list the third line would specify the dimensions (,3) . Actually for character parameters, the first dimension may be omitted altogether, and the dimension may be specified as just (3).

NOTE: To clear a character line of any text, enter a single exclamation mark in response to the prompt for input for that line.

==========

2. C3D DATA FILE FORMAT

2.1 FILE STRUCTURE

The coordinate (and analog) data in .C3D files are written in 16-bit integer (INTEGER*2) format, or optionally floating point (REAL*4) format, consisting of records that are 512 bytes long. The files have three sections. The first section consists of just one header record and contains a few parameters as described in Section 2.2. The second section starts at record number 2 and contains parameters in ADTECH format. This section is variable in length but is typically about 10 records long. The rest of the file contains the 3D point coordinate data, and analog data if analog data were available when the .C3D file was created. Standard format of coordinate and analog data is in the integer format. However, the facility for the data to be written in REAL*4 format is also available for the storing of filtered data where the integer form may not provide sufficient precision.

2.2 HEADER RECORD

Although all parameters of interest are contained in the parameter section, a few essential parameters necessary for accessing the data in the

file are repeated in the header record in simple form. The parameters in this record are useful if one does not wish to use the parameter subroutines required to access the quantities written in parameter format. The ADTECH parameter format allows for very convenient interactive user access, but is more complex from the programming aspect. The first few words of the header record are utilized to store the abbreviated parameter data, and the rest of the header record contains zeros unless time event data have been written to the data file. The parameter values stored in the header record are as listed in Table 2.1. The 16-bit words hold integer values unless otherwise noted.

WORD	CONTENTS
1	Byte 1: number of 1st parameter record Byte 2: must contain 80(decimal)
2	Number of 3D points
3	Number of analog channels for which data is stored
4	Number of first video frame in .C3D file
5	Number of last video frame in .C3D file
6	Maximum interpolation gap allowed (in frames)
7,8	(REAL*4) Scale factor for converting integer 3D data to reference system units. If negative, 3D data is already in REAL*4 format.
9	Starting record number for 3D point and analog data
10	Number of analog frames / video frame
11,12	(REAL*4) Video frame rate (Hz)
13-149	Currently not used
150	12345(decimal) keyword to identify presence of event data
151	Number of defined time events -- maximum of 18
152	Not used
153-188	(REAL*4) Event times (in seconds) -- maximum of 18
189-198	(BYTE*1) Event display switches (0=ON, 1=OFF)
199-234	Event labels. 4 characters for each event.

Table 2.1 -- Contents of first record of .C3D file.

2.3 PARAMETER RECORDS

The parameter section is written in ADTECH parameter format which allows the user to interactively access the parameters with the program PRM. Programmers may access the parameters through the FORTRAN library PRMLIB or write programs to access the parameter data directly. Chapter 3 provides instructions for calling the subroutines in the PRMLIB library, and the rest of this section describes in detail the ADTECH parameter format for those that want to access the parameters the hard way.

The first two bytes of the first parameter record are either not used, or form the first two bytes of the file. If they are the first two bytes of the file (not the case for C3D files), then byte 1 contains the number of the first parameter record, and byte 2 contains 80 (decimal).

Byte 3 of the first parameter record contains the number of parameter records, and byte 4 contains 85(decimal) + processor_type. Processor_type may have the value 1, 2, or 3, and is defined in Section 3.1. The actual parameter data start at byte 5 of the first parameter record.

Within the parameter records, the data belonging to groups and parameters are stored in no particular order, and are located by searching through the total data block. The storage unit is the byte.

Group data are stored in the following format:

Byte 1	Number (n) of characters in group name
Byte 2	Group ID number (negative)
Bytes 3 --> 3 + n - 1	Group name
Bytes 3 + n , 3 + n + 1	(2-byte integer) offset in bytes pointing to start of next group/parameter
Byte 3 + n + 2	Number (m) of characters in group description
Bytes 3 + n + 3 --> 3 + n + 3 + m - 1	Group description

Parameter data are stored in the following format:

Byte 1	Number (n) of characters in parameter name
Byte 2	Group number (positive) to which parameter belongs
Bytes 3 --> 3 + n + 1	Parameter name
Bytes 3 + n , 3 + n + 1	(2-byte integer) offset in bytes pointing to start of next group/parameter
Byte 3 + n + 2	Length in bytes of each data element -1 for Character 1 for Byte

2 for Integer*2
4 for Real*4

Byte 3 + n + 3 Number (d) of dimensions of the parameter (max. = 7)
d = 0 if the parameter is undimensioned.

Bytes 3 + n + 4 --> 3 + n + 4 + d - 1 Parameter dimensions

Byte 3 + n + 4 + d --> 3 + n + 4 + d + t - 1
The parameter data. t = product of all dimensions
and element length

Byte 3 + n + 4 + d + t Number (m) of characters in description

Bytes 3 + n + 4 + d + t + 1 --> 3 + n + 4 + d + t + 1 + m - 1 Parameter description

The PRM program responds to the command SH, which will dump all parameter records in byte format, and their ASCII equivalent, to the screen.

2.4 DATA RECORDS

The 3D coordinate and analog data are written frame-sequentially starting at the beginning of the first data record. The data are packed into records such that frames may cross record boundaries. The 3D coordinate data are normally stored in 16-bit integer format and must be multiplied by the POINT:SCALE factor to generate values expressed in the external (reference coordinate system) measurement units. If the data is stored in floating point format (REAL*4), then the scale factor has already been applied and it is set to be negative.

Each 3D point is described by four words. If the data is stored in integer form, these 16-bit words contain the following:

WORD	CONTENTS
1	X-coordinate of point / scale factor
2	Y-coordinate of point / scale factor
3	Z-coordinate of point / scale factor
4	Byte 1: cameras that measured marker (1 bit for each camera) Byte 2: average residual for point measurement / scale factor

Table 2.2 -- 3D point data contents.

The video data for frame one are stored first, followed by one or more analog "frames" if analog data are present. If the analog data rate was the

same as the video frame rate then only one analog frame will follow each video frame. If analog rate was twice the video rate, then two analog frames will follow the video frame, etc. Both the video and analog data will always be in the same format, .i.e., INTEGER*2 or REAL*4.

NOTES:

In Byte 1 of Word 4, bits are set corresponding to which cameras contributed to the point's measurement. Bit 0 refers to the first camera, bit 1 the second, etc.

If a point was invalid in a frame (not observed by at least two cameras), its 4th word will be set to -1, and the X, Y, and Z coordinates will be set to zero. For points that were interpolated, the residual is set identically to zero.

In an integer file the coordinates and residuals are recorded in internal units, and must be multiplied by the scaling factor in POINT:SCALE to obtain reference coordinate system units.

The analog data are stored in the same order as in the raw data file, i.e., if the analog frame rate was higher than the video frame rate, the order through the channels is repeated as many times as is appropriate. The first word after the end of the analog data is the X-coordinate of the first point in the next frame.

If the data is stored in floating point format, the X, Y, and Z coordinates have already been multiplied by the scale factor. The 4th word is the normal 4th word integer value stored as a floating point number. To extract its data, it should be converted to integer, divided into high and low bytes, and the low byte must be multiplied by the scale factor to obtain the correct residual value.

==========

3. PARAMETER ACCESS FROM PROGRAMS

3.1 PARAMETERS IN FILES

The location of parameter records in any file is specified as follows:

1. Byte 1 of the first record of the file must contain the record number of the first parameter record in the file.

2. Byte 2 of the first record of the file must contain 80(decimal).

3. Byte 3 of the first parameter record must contain the number of parameter records to follow.

4. Byte 4 of the first parameter record must contain 83(decimal) + data_type.

Data_type is an integer having a value 1, 2, or 3 and designates the computer system type the file was created on, or is formatted for. Recognized values are;

1 = PC-DOS
2 = DEC (VAX , PDP-11)
3 = MIPS processor (SGI , SUN)

The library routine GPREC checks for the above four conditions when parameter data is read from a file, and the parameter output subroutine OUTPT ensures that the four bytes described above are written to the output file.

3.2 CALLING PROGRAM REQUIREMENTS

The data stored in ADTECH parameter format may be accessed from user programs by the use of calls to subroutines and functions contained in the parameter library PRMLIB. The library is provided in Microsoft 16-bit FORTRAN format.

The routines operate on the entire parameter data set which must be first read into memory by the function GPREC. Once in memory, parameters may be created, deleted, read, or modified. At conclusion, the entire parameter set may be output to a file by the subroutine OUTPT.

In order to use the library routines, the calling program or subprogram must contain the following statements:

```
PARAMETER (LMSIZE=5120)

INTEGER*1 LM(LMSIZE)

INTEGER*2 RECL,LBOT,LTOP
COMMON /PAR1/LBOT,RECL

RECL=512
LTOP=LMSIZE
```

The array LM must be large enough to store the entire parameter set, i.e. LMSIZE must be at least 512 * number of parameter records. An LMSIZE of 5120 bytes is adequate for most .C3D files.

Before the library routines can be used, the file containing the parameter data must be opened. In this chapter the variable LUN will be used for the data file logical unit number, and LCT will be used for the terminal output logical unit number. Normally they would be assigned in the calling program as for example in

```
LUN=2
LCT=5
```

In the following both LUN and LCT can be constants or variables.

The command to open the file will be similar to the following;

```
OPEN (UNIT=LUN,FILE='file_name',STATUS='OLD',
*  ACCESS='DIRECT',RECL=512,ERR=800)
```

If the file is opened successfully, the routines described in the next section can then be used to access the parameters.

3.3 SUBROUTINES AND FUNCTIONS

3.3.1 Read in all parameters

```
FUNCTION GPREC (LUN,LM,LTOP,ISTREC,INTYP)
```

This function reads in all parameter data into the array LM and converts it to the current processor type (see Section 3.1). GPREC must be declared as an INTEGER*2 function and it returns:

```
GPREC   =  0  if the read is successful
          -1    if there was a file read error
          -2    if parameter records were not found in the file
          -3    if LTOP was exceeded (LM was too small)
```

ISTREC is output, and returns the record number in the file where the parameters started. INTYP is also output and contains either 1, 2, or 3 specifying which processor type the file was formatted for. After the data is read in the input file may be closed.

3.3.2 Write parameters to file

```
SUBROUTINE OUTPT (LUN,LM,ISTREC,IRLAST,OUTYP)
```

Writes the full parameter set to logical unit LUN. The file must have previously been opened as a direct access file. ISTREC is input and specifies the record number where the first parameter record is to be written. This value will be stored in the first byte of the data file as a pointer to the start of the parameter data. IRLAST returns the record number of the last parameter record in the file, i.e., the number of parameter records written is given by IRLAST - ISTREC + 1. OUTYP is input to the routine and specifies in what processor format type the parameters are to be written. This parameter may have the values 1, 2, or 3 as described in Section 3.1.

3.3.3 Create a parameter group

```
SUBROUTINE DEFG (GNAME,DESCR,IFG,LM,LTOP)
```

Creates a parameter group whose name is in GNAME. GNAME may be a character string, or an INTEGER*1 or LOGICAL*1 array terminated with a null byte. Similarly, a group description is specified as a character string or byte array in DESCR. The variable IFG is output and contains the (negative) internal group ID number assigned to this group. An example of a call might be

```
CALL DEFG ('POINT','Point parameters',IFG,LM,LTOP)
```

Note that in Microsoft FORTRAN, the 'C' character must be appended to literal strings, as in 'POINT'C .

3.3.4 Create a parameter

```
SUBROUTINE DEFP (PNAME,TYP,GNAME,DESCR,LM,LTOP)
```

A parameter with the name contained in PNAME is created in an existing group having the name GNAME. TYP is one of the single character

literals 'I','R','B', or 'C' specifying that the parameter is of type Integer (INTEGER*2), Real (REAL*4), Byte (INTEGER*1 or LOGICAL*1), or Character, and DESCR contains a description string. PNAME may specify dimensions which are interpreted as described in Chapter 8 of the User's Manual. Example:

```
CALL DEFP ('NN(3,4)','I','GRP1',
*                'Descriptive text',LM,LTOP)
```

This call would create a parameter named NN of type integer, and having dimensions (3,4). If it is desired to use a variable for one or more of the dimensions, then the subroutine ENCN described in Section 3.3.8 must first be used to encode the dimensions into the parameter name string.

3.3.5 Delete a Group

SUBROUTINE DELG (GNAME,LM,LTOP)

Deletes the group entry for GNAME. All the parameters in the group must have been first deleted. e.g.,

```
CALL DELG ('GRP1',LM,LTOP)
```

3.3.6 Delete a Parameter

SUBROUTINE DELP (GNAME,PNAME,LM,LTOP)

Deletes the parameter named PNAME in the group GNAME and reclaims space, e.g.,

```
CALL DELP ('GRP1','NN',LM,LTOP)
```

3.3.7 Read or write parameters

SUBROUTINE PRPAR (LCT,PNAME,GNAME,IFN,ADDR,LM,LTOP,MAXD)

This subroutine performs the following functions depending on the input value of IFN.

IFN	FUNCTION
0	The contents of the parameter specified in PNAME are listed to unit LCT.
1	Causes a prompt for the interactive modification of the parameter(s) PNAME via logical unit LCT.
2	The parameter values specified by PNAME are extracted from the parameter set and stored in the variable ADDR. MAXD specifies in bytes the maximum amount of data that is allowed to be read into ADDR (i.e., the size of ADDR). If the data specified in PNAME is larger than MAXD, then no action is taken and MAXD is returned as -1. Otherwise MAXD is

returned as the number of bytes actually read in. If the parameter cannot be found MAXD is returned as zero. In an error condition a message is written to unit LCT.

3 The data starting at variable ADDR is stored in the parameter PNAME. MAXD should be set to the number of bytes expected to be stored. MAXD is returned as 0 if there was room for all data, and returned as the number of bytes actually stored if there was not enough room. If the parameter was not found MAXD is returned as -1.

-3 Data is stored as for IFN = 3, but the parameter is locked to prevent it from being modified interactively.

Examples:

```
CALL PRPAR (5,'NN(,2)','GRP1',0,ADDR,LM,LTOP,MAXD)
```

would list the values of NN(1,2), NN(2,2), and NN(3,2) on the logical unit 5.

```
CALL PRPAR (5,'NN','GRP1',2,ADDR,LM,LTOP,MAXD)
```

copies the contents of parameter NN to memory starting at the address occupied by the variable ADDR. Note that no checks for correct variable lengths are performed other than through the MAXD variable. Also MAXD must be a variable and not a constant.

```
CALL PRPAR (5,'NN(2,4)','GRP1',3,JJ,LM,LTOP,MAXD)
```

causes the value in JJ to be stored in the parameter location NN(2,4).

3.3.8 Encode dimensions into a parameter name

```
SUBROUTINE ENCN (NAMIN,NAMOUT,NCHR,NDIM,IDIM)
```

If it is wished to use variables rather than constants to specify dimensions in a parameter name, use this subroutine to form the parameter name before using it in DEFP or PRPAR.

ENCN encodes the first NDIM values in the variable array IDIM and adds the string to NAMIN to produce NAMOUT, which can then be used as PNAME in the subroutine calls, e.g., the call

```
CALL ENCN ('INNAME',NAMOUT,NCHR,NDIM,IDIM)
```

where NDIM = 3, IDIM(1) = 3, IDIM(2) = 6, IDIM(3) = 4, will result in the output of a string array NAMOUT containing 'INNAME (3,6,4)'

A zero value in IDIM will omit the value for that dimension. For example, if IDIM(2) was set to zero in the example, the NAMOUT would contain the string 'INNAME (3,,4)'

3.3.9 Create new parameter block

```
SUBROUTINE START (LM,LTOP)
```

Call this subroutine to initialize a new parameter block, i.e., if you wish to create a new parameter file without using GPREC to read in an existing parameter block.

3.3.10 Convert data between different processor systems

These subroutines are not directly related to accessing parameter data, but are included because they may be useful in converting data between the different computer system types.

SUBROUTINE RCONV (INTYP,OUTYP,RR,N)

RR is an array containing N floating point (REAL*4) numbers to be converted from INTYP to OUTYP. INTYP and OUTYP may have values chosen from 1, 2, or 3 as described in Section 3.1.

SUBROUTINE ICONV (INTYP,OUTYP,IR,N)
IR is an array containing N integer (INTEGER*2) numbers to be converted from INTYP to OUTYP. INTYP and OUTYP may have values chosen from 1, 2, or 3 as described in Section 3.1. The integer formats for type 1 and 2 processors are identical.

3.4 NOTES

1. All of the subroutines in PRMLIB have been compiled with the /4I2 switch for Microsoft FORTRAN. This switch causes all integers to be interpreted as INTEGER*2 rather than INTEGER*4 which is the default for the compiler. All integer variable arguments passed to the subroutines must be declared as INTEGER*2. An alternative technique is to compile the calling program also with the switch.

2. PNAME may contain partial dimensions in order to access a subset of the parameters stored in PNAME, and name and group abbreviations may be used. The program PRM is simply an implementation of the subroutines described in the last section.

3. If an error condition results, then a message is output to unit LCT. If LCT = 0, then no error messages are generated.