.pl 60
.mb 3
.. fare ^OR 72

The new user-program communication package

Software and documentation prepared by:
L.Chiappetti - IFC Milano

12 August 1987

1. Introduction

1.1 Motivations

The **NEW** user-program communication package here described replaces the already existing interface, commonly known as RHPAR package. The reason to develop a user-program communication package has a twofold orientation :

One direction is towards the **USER**. This was the basic motivation for the development of the original RHPAR package. At the origin one had a few basic observations : on one hand users in the astronomical community are accustomed to deal with command languages, which however have a very strict, often unfriendly syntax. On the other hand fully interactive programs (either driven by a terminal dialogue, or by a menu form) are useful for the beginner, but later it could get boring to have to type a lot of responses (and also prone to errors in the case of standard reduction procedures, when the responses are always the same). Therefore the attempt to develop a common interface, which could handle :

Parameters passed in the run string (pseudo command language)

Fully interactive terminal dialogue

Standard set-ups using a command file (pseudo batch execution)

More user-directed motivations were the need for a machine-independent interface (particularly considering that even a simple thing like "list-directed" - i.e. "free-format" - input is implemented in different ways

on different machines). For this reason it has been decided not to include form-filling (menu-driven) input, as this is generally very machine-dependent.

The second direction is towards the **PROGRAMMER**. The original RHPAR interface (hereafter referred to as **Version 1**) was quite awkward to use, had a complex syntax, etc. At the time this was not felt as a problem, but now, with an expanded usage of the interfaces, the time is ripe to provide a new, fully machine-independent, simplified interface, which any scientific programmer can use.

.he User-program communication
Page #
.cp6

1.2 History

The original Version 1 package was developed by the author at the Exosat Observatory, ESOC on an HP RTE4 system under FTN4X Fortran for use in the Exosat LE calibration software, and tuned, after discussion with P.Giommi for use in the LE Interactive Analysis software. Both software packages used a pseudo command language approach (i.e. each command was an independent program). The Version 1 package was built around the HP system-provided routine RHPAR (a command string parsing routine), and made no use of the CHARACTER data type (not available at the time), nor was optimized for programmer-friendliness, although it served all the purposes it was built for.

A later implementation by the author, still on the same system as above, was the replacement of the system-provided RHPAR routine with an emulating parsing routine (so called RXPAR-package), which is able to access command strings generated inside a program. This, together with the rest of the Version 1 package, is used in L.Stella's Timing Interactive package TINTE.

Both variants were succesfully imported on the IFC HP system, under RTE 6/VM and FTN7X, although no attempt was made to implement new features (like e.g. CHARACTER variables).

Finally a Version 1 package for IBM systems (under VM/SP and VS Fortran) was developed by the author at IFC, for use in the Milano Exosat analysis facilities. The only actual requirements for such an implementation were

a way of passing argument strings to programs, and a new RHPAR emulation routine. However some modifications were made to the rest of the package as well, to support CHARACTER variables instead of INTEGER*2 arrays, and to cope with incompatibilities between HP and IBM systems (e.g. due to the fact one is a 16-bit and the other a 32-bit machine). A complete description of the user's features is provided in Chiappetti et al. (1987, Proc. 3rd Cosm. Phys. Nat. Conf., 289).

The latter modifications hindered the compatibility and the exchange of software between the two systems, which was possible only imposing a set of rules (like it has been done for the IFC temporal analysis package).

Hence the need of rebuilding a new package (the present Version 2), which is truly machine-independent, and is described in the present document. It is hoped this package can be easily transported to other machines (e.g. VAX) withy minimum effort.

1.3 New features

This section is mainly concerned with users familiar with the Version 1 RHPAR package. The functioning of the package, in regards of the users, is unchanged (see the Exosat System Handbook, by L.Chiappetti & B.Garilli, section 2.9, and Chiappetti et al., op.cit. section 3). See my note (CONVERS.IA) of May 5, 1987 for programmers instruction about the use of the Version 1 package.

Under Version 1 programs using a command file called routine FROM, and programs using no command file did not call any routine. Now all programs shall call routine COMFIL. Additionally, HP programs using no command file do not need a $FILES directive any more.

Under Version 1 different routines (RDPAR, RDARR, RDANG) were used to read a single parameter, or an array of parameters of the same type. The calling interface for such routines was slightly different on different machines. An unique routine READIN replaces all the above, enabling to read any number of parameters of mixed types, including characters.

Under Version 1 echo of strings (not read from the terminal) was care of the user program. This is now

automatically handled by READIN. Also all logical unit assignment are concealed to the user.

Under Version 1, in the case the dialogue of a program was depending on some of the preceding answers, it was care of the user program to keep trace of parameter numbering. Now READIN is able to read by default the next parameter, unless otherwise instructed.

Under Version 1 the escape character in command files (used to force terminal input) was a backslash on HP and a slash on IBM. Now both are supported on both machines. A terminal EOF handling for IBM is also provided.

A set of "interrogation" utilities, not available in Version 1, is also provided, together with new character string handling utilities.

1.4 Scheme

A typical program generally requires the following actions to be performed :

a get access to a command file, if any

b1 set default values to parameters
b2 write a prompt to user terminal
b3 get a reply from run string, command file or terminal
b4 read the values from the reply
b5 echo the reply
b6 if an error is encountered act in consequencecontinue processing

the above is done for each read

The above is implemented under Version 2 as follows (steps b1 and b2 are of course care of the user) :

.cp2
a call routine COMFIL to access the command file (if any) or set the appropriate logical unit (if reading from terminal)

b3 call READIN from run string, command file or terminal
b4 do a formatted or free-format internal read from the string generated by READIN (for characters see also 9.2)
b5 echo is handled automatically by READIN (this means the string as read is echoed, not the values !)
b6 obtain information about errors in READIN via routine

COMERR, and about conversion errors in the internal read via IOSTAT keyword, then act in consequence

1.5 HP implementation notes

The software for the Version 2 package (with the exception of RCPAR, see 2.2 below) is provided at IFC in library &LBINT (source) and $LBINT (relocatable).

1.6 IBM implementation notes

The software for the Version 2 package is provided at IFC in library LUCIOLIB LIBRARY (source) and as separate members in LUCIOLIB TXTLIB (relocatable).

2. The basic interface routine RCPAR

The function RCPAR is the (machine-dependent) interface routine which retrieves the command string and parses it in separate parameters. It is the only routine which is fully machine-dependent. A machine-specific copy of it shall be provided on each machine where the package is going to be implemented, and is likely to be radically different on each. All other routines will only require minor tuning. RCPAR in Version 2 replaces the RHPAR parsing routine.

It is likely that the user never calls RCPAR directly. It is called internally by COMFIL and READIN to get a parameter. The basic syntax and some implementation notes follow. The calling sequence is :

```
LENGTH = RCPAR (N,STRING)
```

RCPAR shall be defined as INTEGER tout court. It returns the Nth parameter in the runstring in the CHARACTER variable STRING. It also returns the length of the parameter string read in (it returns 0 if no parameter has been found).

2.1 General implementation notes

The RCPAR function acquires the runstring the first time is called, then keeps it internally. The acquisition is machine dependent (see below). The RCPAR function also parses the runstring in separate parameters. The parameter separator (unique) can be encoded in the RCPAR Fortran code (see below for implementations).

2.2 HP implementation notes

On HP RCPAR is a system-provided routine (see exception below).

On HP RCPAR (see Fortran FTN7X manual) may be called as a function or as subroutine. Always call it as a function for compatibility.

On HP a runstring generated when invoking a program from FMGR is of the form RU,program,p1,p2 ... RCPAR therefore expects two parameters (or at least two commas) before the first parameter. When passing a runstring to a program scheduled via CALL EXEC, the buffer shall begin with two commas.

On HP the parameter separator is a single comma

On HP a parameter may be skipped (null parameter) just using two consecutive commas

In the case one wants to generate the runstring in a program and process it internally (i.e. as TINTE does with the emulation of RHPAR provided in file %RXPAR), one needs to load an emulation of the system RCPAR. The code (untested) for such emulation is provided (commented out). The runstring shall be inserted, care of the user program, in common block /PASS/ (also provided, commented out). The emulation shall be copied in a separate file, compiled and linked explicitly.

2.3 IBM implementation notes

On IBM RCPAR is emulated by dedicated Fortran code.

On IBM RCPAR shall be called as a function only.

On IBM the runstring is generated in the calling EXEC with a statement like :

```
&S = &STRING OF  &1 &2 &3 &4 ....
```

and passed through the console stack as &STACK &S. Note that RCPAR expects one true blank before the first parameter (this is for compatibility with the HP original. When typing in the above statement, two blanks (one separator and one "datum") shall follow the words

STRING OF mandatorily. The console stack (terminal) is expected on logical unit 5.

On IBM the parameter separator is a single blank. The EXEC processor will automatically remove surplus blanks.

On IBM a parameter may be skipped (null parameter) using a placeholder (minus sign). Do not use consecutive blanks.

No applications using common block /PASS/ are foreseen on IBM.

The IBM RCPAR disables with a CALL ERRSET the "end of record" error (error 212), occurring when a string shorter than expected is read in with an A format.

.cp5

3. Command file access through COMFIL

The COMFIL function shall be called at the beginning of a program (replacing the call to FROM and all ECHO settings of Version 1), either if using a command file or not, as follows :

```
FNAME=COMFIL(K)
```

Both FNAME and COMFIL shall be declared CHARACTER*12 (this is likely to be enough on most machines). The K parameters shall be set to 0 if the program is not using any command file. Otherwise K is the parameter number of the command file (generally K=1, since the command file is the first parameter). The function returns the name of the command file (or the word TERMINAL).

The logical unit associated to the command file and to the terminal are those specified in common block /COMCOM/ (see 5 below).

3.1 HP implementation notes

On HP the command file, on LU 99, is opened with a named OPEN. The terminal is accessed directly on LU 1 (hence no $FILES directive is any more required when no command file is used).

3.2 IBM implementation notes

On IBM the command file, on LU 51, is opened with an OPEN by unit. The association with unit 51 shall be done via a FILEDEF in the calling EXEC. The terminal is accessed directly on unit 5 (include a FILEDEF if necessary).

4. Parameter access via READIN

The way to access one or more parameters is here described. All parameters accessed in a single read shall be given on a single line (from terminal or command file, missing ones may be defaulted with commas in the case of free-format). If the first parameter of a set is not present in the runstring, all the following parameters read in in the same read are ignored. If the one of the parameters other than the first is missing in the runstring, it defaults to the previous value. The complete calling sequence to read a list of values, including error handling, is :

```
write a prompt
CALL READIN(N,M,MESSAG)
READ(MESSAG,format,IOSTAT=IERR [,ERR=nnn,END=nnn])list
IF(COMERR(KERR).OR.IERR.NE.0) error action
```

.cp2
The arguments N and M to routine READIN indicate the first parameter and the number of parameters to be read. If N is zero, the next parameter is automatically read in. For clarity, it is recommended to insert at the beginning of the program some statements like :

```
PARAMETER (NEXT=0,ONE=1)
```

then calls like the following will be immediately legible :

```
CALL READIN(NEXT,ONE,MESSAG)
CALL READIN(7   ,ONE,MESSAG)
```

respectively as "read in next parameter" and "read in parameter 7". In all cases the parameters are returned in CHARACTER variable MESSAG (provide adequate length, maximum 130 chars).

The LOGICAL routine COMERR (see 6 below) traps any error occurred in READIN. Conversion errors in the internal READ shall be trapped autonomously (IOSTAT=IERR) and may be processed separately (ERR=nnn etc.). See also implementation notes. Note that MESSAG can be read with any user-defined format, although

one generally will use either free-format (*) for numeric values and maybe for characters (see 9.2), or A-format for characters (one might also use a direct character assignment, which however does not improve legibility of the program).

Note that routine TRULEN (see 9.1) is called internally.

See 9.2 for character input handling.

READIN may generate an internal error 999 (trapped using COMERR) in the case the concatenation of all parameter (strings) read in exceeds the maximum length of the internal work area. Such work area (as well as the maximum length of the runstring, and the maximum length of a single parameter) is currently fixed to 130 characters (this is consistent with IBM CMS limitations).

4.1 HP implementation notes

The current version of READIN (as well of all Version 2 package) presume the FTN7X compiler has been generated in 66 compatibility mode (or option 66 is used).

Under such assumption, a carriage return when reading from the terminal is NOT interpreted as End-of-file, but just terminates current read (default).

A control-D is instead interpreted as EOF. This (or a command file EOF) returns error -1 (when calling COMERR).

There is a difference in the READIN code for HP, due to the inability of FTN7X to handle null strings like A(1:0), and to handle "overlapping" assignments like A(1:K)=A(1:L)//B (even if L<K).

The READ(MESSAG,*) statement, if MESSAG is empty, blank, or does not contain some of the parameters, does not trigger any error.

Format (conversion) errors in READ are trapped via IOSTAT.

4.2 IBM implementation notes

Contrary to IBM standards a carriage return when reading from the terminal is trapped by the IBM version of READIN and NOT interpreted as End-of-file, but just

terminates current read (default).

Since all actual terminal read is handled via A-format, reading in a string, the IBM free-format prompt (?) is no longer displayed.

Since no terminal EOF is therefore available, this is simulated by the character sequence /*. This, when input by a terminal, (or a command file EOF) returns error -1 (when calling COMERR).

The READIN code for IBM presumes the ability to handle null strings like A(1:0), and to handle "overlapping" assignments like A(1:K)=A(1:L)//B (even if L<K).

The READ(MESSAG,*) statement, if MESSAG is empty, blank, or does not contain some of the parameters, triggers error -1 (in IOSTAT).

Most format conversion errors in READ (like reading a numeric value from a character field, which is read as all zeros) do not trigger any error (refer to IBM VS Fortran manual).

5. The common block COMCOM

This common block contains all information used internally by the Version 2 package. The relevant BLOCK DATA contains all logical unit customization. The common block needs not to be included in the user program. A description of the content follows. All values are INTEGER tout court (or LOGICAL tout court), and defaults to 16-bit, 32-bit or otherwise according to each machine's default.

LUCOM is the current logical unit (set to LUIN or LUFIL by COMFIL), from which command input is taken

ECHO (logical) is true if input is coming from a command file, false if coming from a terminal.

LUTERM is the logical unit for messages (terminal output)

LUIN is the logical unit of the terminal (input)

NXTPAR is the next parameter to be read

LERR is the error code of last read

LUFIL is the logical unit for command files

All interrogation routines described below access this common block.

LUFIL,LUIN and LUTERM are the values which may be customized in the block data on different machines. All other values are assigned internally.

.cp5

5.1 HP implementation notes

LUTERM is set to 1, LUIN is set to 1 and LUFIL is set to 99 (therefore such logical unit number is reserved).

Necessary $ALIAS directives are included in all routines.

5.2 IBM implementation notes

LUTERM is set to 61 (since 6, the default for terminal output, is also the destination of error messages). A FILEDEF is therefore necessary. LUIN is set to 5 and LUFIL to 51 (this also requires a FILEDEF).

6. Query last error via COMERR

The logical function COMERR (declare as LOGICAL tout court) is true if an error has occurred during last READIN. The error value is returned in KERR, with the calling sequence e.g.:

```
IF(COMERR(KERR))GOTO 999

....
999 WRITE(luterm,998)KERR
998 FORMAT(' Detected error ',I5)
```

7. Query if command file defined via COMAND

The logical function COMAND (declare as LOGICAL tout court) is true if a command file is in use, and false if the terminal is used as command file. The calling sequence

is e.g.:

IF(COMAND())GOTO label

8. Query next parameter via COMNXT

The integer function COMNXT (declare as INTEGER tout court) returns the number of the next parameter to be read. The calling sequence is :

I=COMNXT()

9. Character handling utilities

9.1 True string length via TRULEN

The integer function TRULEN (declare INTEGER tout court) is used internally, and may be used freely in any program (even if not using the Version 2 communication package) to obtain the true length of a character string, with the following calling sequence :

LENGTH=TRULEN(STRING)

The difference with the standard Fortran call LEN(STRING) is clarified by the following example :

let a variable be defined as CHARACTER*12 STRINGand be assigned a value as STRING ='ABC'

LEN will always return 12 (the length of the allocated memory area, so called "inherited length" in subroutines with declaratives CHARACTER*(*)) , while TRULEN will return 3, that is the length excluding any trailing blank.

9.2 Free-format character input with EXPAND

The way character string input is handled by standard Fortran is not satisfactory, particularly with multiple strings (arrays of strings). Let us assume for example to have an array CHARACTER*6 ARR(3). One can always read in such an array with a fixed format like (3(A6,1X). This implies on most machines (certainly on IBM and HP) that it is care of the user to format the input field :

AAAbbb BBBBbb Cbbbbb

where b indicates a significative blank, will be read in correctly as ARR(1)='AAA ', ARR(2)='BBBB ', ARR(3)='C ', but for instance :

```
AAA BBBB C
```

will not be read correctly (ARR(1)='AAA BB', etc.). Using a generic A-format like (3(A,1X)) does not help in this case. Use of a generic A-format (A) is however effective if one is reading in a single string. The exact number of characters in the string (eventually padded with blanks) is read in.

If one wants to read multiple strings, one may use list-directed (free-format) input, however character free-format expects each value to be enclosed in primes (quotes), like :

```
'AAA' 'BBBB' 'C'
```

separated by blanks or commas. These will be succesfully read in by a statement like READ(in,*)ARR. However it is particularly annoying to have to include quotes, specially when passing parameters in a runstring. To overcome this, routine EXPAND may be used. The calling sequence is :

```
KERR=EXPAND(STRING)
```

where EXPAND shall be declared INTEGER tout court. KERR is always zero, unless an error in EXPAND has occurred. EXPAND will take a string, parse it in fields, each field being separated by a single comma or blank, and rebuild a string containing all fields enclosed in primes and separated by commas. The following examples may help :

```
AAA BBBB C              becomes   'AAA','BBBB','C'
AAA,BBBB,C         also becomes   'AAA','BBBB','C'
AAA  BBB C              becomes   'AAA',' ','BBB','C'
```

Since the replacement occurs in-place, there shall be enough space left in STRING for the quotes (two for each field) to be inserted. If this is not the case KERR is set to a non-zero value. Note improper handling in the case more than one blank are used to separate fields (third example above!).

A string, once reshuffled by EXPAND, may be read with free-format. Summarizing a CALL READIN(NEXT,ONE,MESSAG) may be followed by :

```
READ(MESSAG,'(A)')STRING
```

which will work as well as

```
READ(MESSAG,*)STRING
```

for a single string, but for multiple string one shall use the second form either passing parameters already enclosed in quotes, or as follows:

```
KERR= EXPAND(MESSAG)
READ(MESSAG,*)ARR
```

Note however that any "missing" parameter is set to a null string by READIN, and to blank by EXPAND, therefore the corresponding value is assigned as blank (no default kept), consistently with current character i/o handling. Similarly occurs with an A-format with a single string. With free-format in the case of a single string, or with EXPAND for multiple string, a carriage return, or blank line in command file, will however keep the default. This form is recommended, the other shall be avoided, as well as reading in a mixture of characters and numeric values, like 1.0,ABCD,3.5.

As implementation note, the basic version (IBM) assumes Fortran to be able to handle "overlapping string" assignments like :

```
STRING(1:)=PRIME//STRING(1:LL)
```

On machines (like HP) where this is not possible, alternative coding shall be used (see the example in the HP implementation).