INTRAC - Programmer's guide

Schönthal, Thomas

1977

Document Version:
Publisher's PDF, also known as Version of record

CODEN: LUTFD2/(TFRT-7128)/1-033/(1977)

INTRAC - Programmer's Guide

Tomas Schöntal

Department of Automatic Control
Lund Institute of Technology
October 1977

Dokumentutgivare
Lund Institute of Technology
Handläggare
Dept of Automatic Control
Författare
Tomas Schönthal

Dokumentnamn
REPORT LUTFD2/(TFRT-7128)/1-33/(1977)
Utgivningsdatum
Oct 1977, 1st printing
Dec 1979, 2nd printing
Ärendebeteckning

10T4

Dokumenttitel och undertitel
INTRAC - Programmer's Guide

Referat (sammandrag)
This report describes in detail how to construct a command driven interactive program package (preferrably in FORTRAN) based upon the interactive module INTRAC. The following problems are discussed:

Designing application routines and interfacing them to INTRAC;

Calling and initializing INTRAC;

How to use INTRAC's auxiliary routines for argument decoding and data exchange between INTRAC and application routines;

Display handling;

A few examples are taken from IDPAC, one of the first program packages to utilize INTRAC, but the methods are quite general.

Referat skrivet av
Author

Förslag till ytterligare nyckelord
Computer aided design, Interactive programs

Klassifikationssystem och -klass(er)
50T0

Indextermer (ange källa)
Computer software (Thesaurus of Engineering and Scientific Terms, Engineers Joint Council, N.Y., USA).

Omfång
33 pages

Övriga bibliografiska uppgifter
56T2

Språk
English

Sekretessuppgifter
60T0

ISSN
60T4

ISBN
60T6

Dokumentet kan erhållas från
Department of Automatic Control
Lund Institute of Technology
Box 725, S-220 07 LUND 7, Sweden

Mottagarens uppgifter
62T4

Pris
66T0

DOKUMENTDATABLAD enligt SIS 62 10 12

SIS-DB1

Blankett LU 11:25 1976–07

INTRAC - Programmer's Guide

Tomas Schönthal
Department of Automatic Control,
Lund Institute of Technology, Sweden

Contents

1) Abstract

This report describes in detail how to construct a command driven interactive program package (preferrably in FORTRAN) based upon the interactive module INTRAC.

The following problems are discussed:

- Designing application routines and interfacing them to INTRAC
- Calling and initializing INTRAC
- How to use INTRAC's auxiliary routines for argument decoding and data exchange between INTRAC and application routines
- Display handling

A few examples are taken from IDPAC, one of the first program packages to utilize INTRAC, but the methods are quite general.

2) Scope

The scope of this report is to define some programming conventions that are necessary in order to interface INTRAC successfully to usersupplied application routines.

The runtime facilities of INTRAC as well as a discussion of the design philosophy behind INTRAC are dealt with in:

"INTRAC - A Communication Module for Interactive Programs - Language Manual" by Hilding Elmqvist and Johan Wieslander, which the reader is assumed to have read, since concepts such as MACRO, global variable etc. will appear in this text without explanations.

The implementation problems are treated in: "Implementation Procedures for INTRAC", by Tomas Schonthal.

Problems arising when using program languages other than FORTRAN are not discussed.

In the first paragraphs the programming methods are merely outlined and examplified, to be followed by appendices containing exact information about programming conventions and data representations, etc.

3) Main Program Outline

Constructing a command driven program package around INTRAC puts some constraints on the programmer.
Among other things the following items are affected:

- The design of the main program
- The integrity of INTRAC's own data areas
- The information exchange between INTRAC and the user's application routines

This does of course in no way limit the choice of problem solving algorithms and data representations.

The following table describes the idea behind the main program. Some paragraphs contain a reference to an appendix with detailed information.

a) Initialize INTRAC (and if necessary, the application section of the program package), i.e. define mode flags, assign default values, etc. (See Appendix A.)

E.g. at this stage IDPAC writes a version message.

b) Obtain one command line, decode it and transfer the table of application command names by a call to SUBR. INTRAC. (See Appendix A.)

Error? If so: Goto d).

Empty line? If so: Goto b).

Was the command STOP? If so: Exit.

INTRAC has now provided an index, which identifies the desired application routine, to which control may be passed e.g. by means of FORTRAN's computed GOTO construct.

c) Pass control to the desired application routine.
Error? If so: Goto d). If not so: Goto b).

d) Write an appropriate error message, e.g. by a call to an application dependent error routine, which, if the error was flagged by INTRAC, should pass control to INTRAC's own error routine (see Appendix A), then goto b).

4) Error Routine Outline

The convention for signalling error conditions is to assign a global error indicator provided by INTRAC a positive value. (See Appendix G.)

E.g. in IDPAC this error indicator assumes values in several intervals of length 100, each one representing a unique category of error cases. E.g. the interval [901,999] is reserved for Least Squares Identification errors, and to be more specific, error case 916 corresponds to: MODEL ORDER TOO HIGH - TOO MANY PARAMETERS

IDPAC's error routine is therefore organized in two levels:

a) Overall error routine, which is called whenever an error message is desired. The overall routine merely identifies the error category.

   The error interval [1,49] is reserved for INTRAC itself (See Appendix A.)

b) Error category routine, several identical routines, insofar as program logic is concerned. These routines handle the individual error cases, within the scope of one error category.

This organization has the advantage of being easily understood and updated, since its only elements are:

- A computed GOTO statement on the error indicator
- A WRITE statement and a FORMAT key for each error case
- A call to INTRAC's own error routine from the overall error routine (See Appendix A.)
- An echoing logic to remind the operator of an erroneous command if the operation mode is MACRO with no automatic echoing (See Appendix D.)

5) Functional Description of INTRAC

In order to further explain the behaviour of the program package,
a simplified flow table of INTRAC is provided.

a) Is the operation mode MACRO? If so: Take administrative measures for MACRO execution/generation (could be affected by the error indicator).

b) Clear the error indicator.

c) Read a free format command line from appropriate device (keyboard or mass storage, respectively, depending on operation mode).

d) Divide the command line into typed (classified) data items (arguments), the first of which will be referred to as the command word (command identifier). Were the arguments syntactically correct? If not so: Set the error indicator and return.

e) Are INTRAC's own commands desired and does the command word match any of their names? If so: Execute the appropriate INTRAC command, then goto c) if no error occurred, otherwise return.

f) Substitute actual values for variable names in the command line, if any. (See Appendices B and C.)

g) Are any application commands desired and does the command word match an entry in the table of application command names? If so: compute an application selector needed by the main program as described in Section 3, then return.

h) Should the command line be treated as data? If so: Return.

Was it desired to execute a MACRO? If so: Look up a text file, whose name equals the command word and whose first line is a legal MACRO statement. O.K? If so: Enter MACRO execution mode, then goto c).

If not so: Flag the error case INVALID COMMAND, then return.

Comments

- Note that INTRAC is the main routine for its own commands, so that control will not be returned until a

command line is identified either as an application command, a data line or an empty line.

- The flow table does not explain the echoing feature as well as some details concerning MACROs. However, these features are with one exception (to be explained in Section 6) transparent to the application routines.

- Note the scan order between INTRAC's own commands, application commands and MACROs.

- INTRAC merely flags error cases, control to INTRAC's own error routine should be passed via the calling program's error routine.

- The error indicator should be cleared by INTRAC and not by the application routines.

6) Designing Application Routines

The design of the application routines leaves the programmer with the greatest creative freedom. It is therefore impossible to set up general rules, so instead a hypothetical application routine in the spirit of IDPAC is discussed in order to:

- Give a general idea of how an application routine might look.
- To demonstrate the usefulness of INTRAC's auxiliary routines.

This type of application routine normally consists of six sections:

a) Initialization, in which parameters are assigned default values, flags are cleared, etc.

b) Argument decoding, in which the unique syntax of the command is tested in detail. INTRAC's syntax control may be thought of as a first, application independent pass, the argument decoding as a final, application dependent one.

The argument decoding is carried out by means of special LOGICAL FUNCTIONs, which access and correctly interprete INTRAC's data areas.
(See Appendix B.)

INTRAC has an operation mode:
"Define a MACRO but do not execute it",
in which the program package as a whole acts as an interactive compiler, verifying the formal correctness of the command line before it is added to the MACRO, but no more.

For this purpose INTRAC provides a flag. If that flag is set, then the application routine should pass control back to the main program after the argument decoding, except for what is said at e) about command logging.
(See Appendix G.)

Note

This is the only way that INTRAC's MACRO facility affects the application routines, and it has to be considered, if the program package as a whole is to function properly. Otherwise the program package might attempt to access undefined data or alter existing data, etc. Whether to perform the syntax control or not in this mode

is up to the application programmer.

c) Data input (typically in the form of files on mass storage).

d) Evaluation of input data; usually accomplished by calls to prewritten scientific library routines.

e) Data output, very similar to c). However, the programmer should keep in mind that after entering e) there is no "going back". If, for some reason, the results from d) are partly erroneous, or the application routine is unable to produce all the results, then it is recommended that either the output should be completely inhibited or some action be taken to prevent the output from being misunderstood or, perhaps even better: the operator should be allowed to take some remedial action.

If the command was successfully carried out, it might be desired to log it on the listing device. That will be accomplished by a call to a special INTRAC routine (See Appendix D). Just note that if the application routine outputs to the listing device, then the logging should precede that output.

f) Error messages
If an error occurs between a) and e), then INTRAC's global error indicator should be set to an appropriate value and control be transferred back to the main program (See Appendix A).

In some cases, however, extra measures need to be taken, e.g. still open input files should be closed, temporary files should be deleted, etc. In short the runtime environment should to the largest possible extent be reset to its state at the time before the attempt to execute the erroneous command was made.

It is good practice to collect all the error case assignments in a tail at the end of the application routine, each one preceeded by a comment line.

Formally, the application routines are SUBROUTINES lacking arguments, implicitly communicating with INTRAC via argument decoding functions and COMMON blocks. In a general sense, however, they may be regarded as separate programs, interchanging data via mass storage files, reserved memory areas, etc.

Subcommands

The application routine may be regarded as a miniature program package itself, with a command table of its own, so called subcommands. In this fashion it is possible to program an hierarchy of commands. The application routines call SUBR. INTRAC, test the error flag and, if necessary, call the error routine in the same way as the main program, except for the initialization call which should be made just once during the entire run of the program package.

A good example of the use of subcommands is IDPAC's data editor PLMAG.

Its main module accesses and prepares a data file, while the subcommands perform (not seldom a great number of) manipulations of its contents. For efficiency reasons it pays well to leave the data file open between these simple manipulations.

To ensure only one exit point from the program package, the INTRAC command STOP should be flagged as erroneous at subcommand level (See Appendix A).

7) Argument Decoding, a Programming Example

Consider the hypothetical application command:
(Square brackets are not part of the command line,
they merely enclose optional arguments.)

COM [OUTPUT =] INPUT [PARAM]

COM	- command word
OUTPUT	- name of output data set
=	- assignment operator
INPUT	- name of input data set, if OUTPUT is missing, then INPUT will be overwritten by the output data
PARAM	- floating point parameter, if missing, then the value 3.2 is assumed

INTRAC stores the values of the arguments and their types (e.g. NAME, INTEGER, FLOATING POINT, etc.).

In this case four types of arguments are present:

- NAMEs (OUTPUT and INPUT), which are accessed by means of the LOGICAL argument decoding FUNCTION LHNAME

- DELIMITER (=), which is accessed by means of the LOGICAL argument decoding FUNCTION LHOLL

- FLOATING POINT NUMBER (PARAM), which is accessed either by the LOGICAL argument decoding FUNCTION LREAL or LNUMB, the latter preferrable, since it also interpretes INTEGERs as FLOATING POINT NUMBERs.

- LINE TERMINATOR or end of command, which is accessed by means of the LOGICAL argument decoding FUNCTION LTERM

In order to obtain the arguments, the user has to supply a pointer (here named ICNT) to the argument he tries to access.

Examples

- Assume that: ICNT=3, R=3.2 and that INTRAC has processed the command line:

ARG1 ARG2 * ARG4

The FUNCTION CALL: LOG=LNUMB(ICNT,R) then yields:

LOG=.FALSE. , ICNT=3, R=3.2

LOG indicates failure, since * is no legal NUMBER,
which leaves ICNT and R unaffected.

- Assume the same initial values for ICNT and R as above,
but consider the following command line instead:

ARG1 ARG2 12.34 ARG4

In this case the above call to LNUMB would yield:

LOG=.TRUE. , ICNT=4, R=12.34

LOG indicates success, since 12.34 is a legal number, so
ICNT has been auto-incremented, i.e. made to point at
the argument immediately after 12.34,
and R has assumed the value 12.34 .

Assuming that SUBR. INTINI was called with
ARRLG=.FALSE. (see Appendix A),
the hypothetical command may now be decoded by
the following FORTRAN logic:

Table 1-1. 1

```
...
      LOGICAL LHNAME,LHOLL,LNUMB,LTERM, LOUT,LOG
C
C     HINPUT AND HOUTP HOUSE THE 8-CHAR. NAMES
C     INPUT AND OUTPUT, RESPECTIVELY
C     IN THE FORMAT (4HXXXX,4HXXXX)
      DIMENSION HINPUT(2),HOUTP(2),HEQUAL(2)
C
      COMMON /COMINF/ MACLOG,IERR,ICNR,NRL,NRR
C
      DATA HEQUAL
     1    /4H=   ,4H    /
...
C
C     INITIALIZE THE ARGUMENT POINTER AND
C     VERIFY THE PRESENCE OF ARGUMENTS
      ICNT=2
      IF(LTERM(ICNT)) GOTO error
            (command too short)
C
C     ASSIGN THE DEFAULT VALUE TO PARAM
      PARAM=3.2
C
C     TEST THE PRESENCE OF OUTPUT
      LOUT=NRL.GT.3
```

```
      IF(.NOT.LOUT) GOTO 100
C
C     OBTAIN OUTPUT
      IF(.NOT.LHNAME(ICNT,HOUTP)) GOTO error
                        (OUTPUT not recognized)
C
C     OBTAIN =
      IF(.NOT.LHOLL(ICNT,HEQUAL)) GOTO error
                              (= not recognized)
C
C     OBTAIN INPUT
  100 IF(.NOT.LHNAME(ICNT,HINPUT)) GOTO error
                           (INPUT not recognized)
C
C     LET OUTPUT ASSUME ITS DEFAULT VALUE, INPUT,
C     IF OUTPUT IS NOT PRESENT IN THE COMMAND LINE
      IF(.NOT.LOUT) CALL HSTORV(HINPUT,HOUTP,2)
C
C     OBTAIN PARAM
      LOG=LNUMB(ICNT,PARAM)
C
C     VERIFY END OF COMMAND
      IF(.NOT.LTERM(ICNT)) GOTO error
                     (command too long)
C
...
```

Note

- The text string format and SUBR. HSTORV are defined in Appendix F.

- The argument decoding functions (See Appendix B) may be combined to form multi-argument decoding functions.

  E.g. the construct NAME1[(NAME2)] appears frequently in IDPAC, where it is decoded by a call to the LOGICAL FUNCTION LSYNAM, which is based upon the argument decoding functions LHNAME and LHOLL.

- The meaning of the parameter NRL in COMMON /COMINF/ is explained in Appendix G.

- If INPUT and OUTPUT represent file names, then they should be decoded by LOGICAL FUNCTION LFINAM rather than by LHNAME (See Appendix B).

8) Appendix A - Initializing and Calling INTRAC, INTRAC's Error Routine

INTRAC is initialized by a call to SUBR. INTINI, whereby:

- Command decoding mode is specified
- INTRAC's global error indicator is cleared
- INTRAC's internal registers are initialized

- -

SUBROUTINE INTINI(MDEV1,MDEV2,ARRLG,CMLOG)

Arguments, (I) and (O) denote input and output arguments, respectively

MDEV1,2 - logical unit numbers for MACRO I/O (I)
Note: MDEV1 may not coincide with any of the parameters in COMMON /DEVICE/. MDEV1,2 should be file oriented. (See Appendix G.)

ARRLG - LOGICAL mode switch, if .TRUE. , then < is treated like any other character, else < will act as an argument separator, i.e. the numbers of arguments to the left and to the right of it will be held in different registers, and the < itself will not be accessible from the application routines (I) (See Appendix G.)

COMLG - LOGICAL switch controlling the decoding mode for the comma sign, if .FALSE. , then comma signs are decoded just like any other characters, else they will be replaced by the argument at the same position in the previous command line, if possible, and if not possible, then the global error indicator will be set (I)

- -

```
 SUBROUTINE INTRAC(CTAB,NCTAB,NSPACE,IREADY,
1                  LINSW,MACSUS,STOLOG)
```

Arguments (same conventions as for SUBR. INTINI)

CTAB - matrix containing the table of application commands, size (2,NCTAB), dimensioned (2,.) the j:th command name (0 < j < NCTAB+1) is stored with its first four characters in CTAB(1,j) and with its last four characters in CTAB(2,j) (I)
For this reason the j:th command name will occupy the j:th column in CTAB and thus its eight characters may be accessed as one unit. (See Appendix F.)

NCTAB - number of command names in CTAB (I)

NSPACE - number of spaces to the left of the prompter (I)

IREADY - prompter (I)
=0: no prompter
=1: >
=2: #
=3: <
=4: ?
=5: space

LINSW - facility switch (I)
= -1: read the command line into INTRAC's buffer
= 3: (-1) + divide, if possible, the command line into arguments
= 0: (3) + scan INTRAC's own table of commands
= 1: (0) + scan the table of of application commands + if necessary, look up a MACRO, whose name equals the line's first argument
= 2: decode the first argument only
= 4: same as (1) except that no attempt is made to look up a MACRO
= 5: same as (1)

Note
----

(1) is most frequently used, when the command line should be interpreted as an executable statement

(3) is used if the command line should be interpreted as data

(2) can be used to implement a text editor with MACRO facility

MACSUS - LOGICAL flag, if .TRUE. , forcing the current MACRO to be suspended,
MACSUS is set by the calling routine in order to transfer a break signal, e.g. a hardware condition such as the occurence of a certain switch register combination.
always returned .FALSE. (I/O)

STOLOG - returned .TRUE. , when the INTRAC command STOP has executed (O)

- -

INTRAC has an error routine of its own, SUBROUTINE INTERR, which in case of error, writes error messages on logical unit LTO in COMMON /DEVICE/ (See Appendix G.)
Control should be passed to SUBR. INTERR via the program calling SUBR. INTRAC if the error flag was set.

Note

- INTERR has reserved the error cases 1 - 49

- When operating in MACRO mode with no automatic echoing, INTERR itself echoes the command line on logical unit LTO in COMMON /DEVICE/.

9) Appendix B - Argument Decoding Functions

To decode the command line previously processed by INTRAC there exists a set of argument decoding functions with the following properties in common:

- They are all LOGICAL FUNCTIONs, returned .TRUE. on success, otherwise .FALSE.

  Note: If operating in 'Define but do not execute a MACRO' mode and e.g. an integer argument is expected, the corresponding function (LINT) is returned .TRUE. if a legal variable name is found instead, since an integer variable with that name could exist at the time the MACRO is going to be executed.

  Consider the command string DATA(I). DATA is some vector and I is an index. When a MACRO is executed, the local variable I, will be replaced by its value, at the moment the application routine is entered. The substitution is made automatically by INTRAC, so from the application routine's point of view the command strings DATA(I) and DATA(3) are equivialent, assuming that the value of I is 3. When operating in 'Define but do not ..' mode the substitution is not carried out, so in order to be able to verify the formal correctness of a command line, the argument decoding functions will be returned .TRUE. if the arguments follow the syntax for a variable name, i.e. VARNAME[.[EXTENSION]]

- They all have the first argument ICNT, which is an argument pointer, that will be auto-incremented on success, and remains unaffected in case of failure, except for LFORML and LTERM.

- With the exception of LFORML, LHOLL and LTERM, the command line argument is returned as one of the function's actual arguments, which in case of failure will remain unaffected.

  Arguments (I) and (O) denote input and output arguments, respectively

---

LOGICAL FUNCTION LDELIM(ICNT,DELIM)

Returns a delimiter

ICNT - argument pointer (I/O)
DELIM - delimiter (I/O)
(See Appendix F.)

- -

LOGICAL FUNCTION LFINAM(ICNT,HFINAM)

Returns a legal file name, i.e. a name consisting of at most a system dependent number of characters (which itself must not exceed 8).

ICNT - argument pointer (I/O)
HFINAM - file name, size (2) (I/O)
(See Appendix F.)

- -

LOGICAL FUNCTION LFORML(ICNT)

Verifies the presence of a formal argument(*), the value of which has to be obtained by means of one of the other argument decoding functions.

Note: the argument pointer is never affected by LFORML

ICNT - argument pointer (I)

(*) Otherwise stated: LFORML detects that the value and type of an argument is currently unknown, implying that the syntax control cannot be carried out in full detail, yet. Intention: To resolve syntax ambiguities in 'Define a MACRO but do not execute it' mode.

- -

LOGICAL FUNCTION LHNAME(ICNT,HNAME)

Returns a name consisting of up to eight characters

ICNT - argument pointer (I/O)
HNAME - name, size (2) (I/O)
(See Appendix F.)

- -

LOGICAL FUNCTION LHOLL(ICNT,EXPHOLL)

Compares the current argument with EXPHOLL

ICNT - argument pointer (I/O)
EXPHOLL - expected text string, size (2) (I)
(See Appendix F.)

- -

LOGICAL FUNCTION LHOLLS(ICNT,POSSIB,NR,IHOLL)

Compares the current argument with a set of strings

ICNT - argument pointer (I/O)
POSSIB - matrix containing comparison strings, size (2,NR), dimensioned (2,,) (I) each string occupies two consecutive components of POSSIB (See Appendix F.)
NR - number of strings in POSSIB (I)
IHOLL - returned containing a pointer to POSSIB for the matching string (I/O)

- -

LOGICAL FUNCTION LINT(ICNT,INT)

returns an integer argument

ICNT - argument pointer (I/O)
INT - returned integer (I/O)

- -

LOGICAL FUNCTION LNUMB(ICNT,R)

Returns a number, either real or integer, as a real number

ICNT - argument pointer (I/O)
R - returned real number (I/O)

- -

LOGICAL FUNCTION LREAL(ICNT,R)

Returns a real number

ICNT - argument pointer (I/o)
R - returned real number (I/O)

- -

LOGICAL FUNCTION LTERM(ICNT)

Verifies line terminator

ICNT - argument pointer (I)

10) Appendix C - Accessing Global Variables

The global variables act as a scalar data base, which may be either accessed automatically by INTRAC, when the global variable references are substituted in the command line. They may also be accessed explicitly from the application routines.

In the latter case the folowing two groups of INTRAC routines should be used:

a) Fetch Routines

FIDENT - fetches an identifier
FINT - fetches an integer
FNUMB - fetches a number, either integer or real, the result is always converted into a real number
FREAL - fetches a real number

Arguments

(I) denote input argument, (O) denote output argument, and * indicates that the type of the argument depends upon which routine it belongs to.

ARG - the global variable's name, vector size (2) (I) (See Appendix F.)

EXT - the global variable's extension, same format as ARG (I)

VALUE* - the global variable's actual value (O)
FIDENT: same format as ARG
FINT : integer
FNUMB : real
FREAL : real

IND - indicator (O)
=0: success
=1: the global variable ARG.EXT is undefined
=2: the type of the global variable ARG.EXT differs from the expected (e.g. integer in the case of FINT)

b) Deposit Routines

DIDENT - deposits an identifier
DINT - deposits an integer
DREAL - deposits a real number

Arguments

Same conventions as in a).

ARG - see a)

EXT - see a)

VALUE* - value to be assigned to the global variable ARG.EXT (I)
DIDENT: same format as ARG
DINT : integer
DREAL : real number

IND - indicator (O)
=0: success
=1: no room for the global variable ARG.EXT
=2: unused
=3: attempt made to change the type of the global variable ARG.EXT

11) Appendix D - Command Logging and Echoing

The command lines may be logged on the listing device (Logical unit LLP, see Appendix G.) by a call to the INTRAC routine LPCOM, provided that INTRAC's global error indicator is zero (See Appendix G.) and INTRAC's switch LOG is ON
(See "INTRAC - a Communication Module for Interactive Programs - Language Manual")

Note

For application and MACRO calls actual values will appear instead of variable names, which will not be the case for INTRAC's own commands.

SUBROUTINE LPCOM(NSPACE,IREADY)

Arguments: (I) denotes input argument

NSPACE - number of spaces to precede the line (I)
IREADY - character to immediately precede the line (I)
=0: no character
=1: >
=2: #
=3: <
=4: ?
=5: space

Examples:

Assuming the command line:
ARG1 ARG2 * ARG4

CALL LPCOM(0,1) yields:
>ARG1 ARG2 * ARG4

and CALL LPCOM(5,2) yields:
#ARG1 ARG2 * ARG4

- -

The command line may be echoed onto the terminal by the application error routine by a call to SUBR. ECHBUF, no arguments.

The command line will be echoed on logical unit LTO (See Appendix G.) provided that INTRAC's automatic echoing feature is disabled and the operating mode is MACRO.

SUBR. ECHBUF was designed merely in order to relieve the application error routine from accessing INTRAC's internal data areas. It should be called unconditionally, since the echoing condition is tested internally.

12) Appendix E - Display Handling

An interactive dialogue consists of prompting and error messages and command input lines plus very often of graphical output.

The organization of this I/O depends to a great deal on the chosen terminal/display configuration.
The simplest case is that in which command dialogue and graphical output appear on separate physical units.

If the graphical and command output are to appear on the same physical devices, however, then problems will arise about how to divide the screen, place the text cursor, etc.

To minimize the programming job, it is possible to define the proper configuration characteristics by a number of calls to a set of SUBR. collectively called the DISHDL.

All the logic governing the dialogue layout is isolated to the DISHDL, so a change in terminal configuration will automatically be taken into account as soon as the contents of COMMON /DEVICE/ is modified.
(See Appendix G.)
In other words, the DISHDL makes the program package independent of the chosen terminal configuration.

(I) and (O) denote input and output arguments, respectively.

SUBROUTINE DISHDL

Handles text I/O on a display terminal

DISHDL operates in two different modes depending on the states of the LOGICAL flag IAXES in COMMON /DISCOM/
(COMMON /DISCOM/ IAXES,NR,NC must be memory resident throughout the entire run of the program package.)

=.FALSE. - TEXT mode, no special positioning of the text lines is made

=.TRUE. - PLOT mode, employed when the display is used for both plotting and text I/O
Text and graphics output will not interfere
the text starts in the HOME position.
(I.e. the upper left corner of the screen.)
several output lines may appear on the

same line on the screen (tabulated)
a carriage return, line feed will
be issued only when that line is full

DISHDL can be used in different I/O device combinations, defined by COMMON /DEVICE/ (See Appendix G.), even if no display unit is available.

Routines

- -

SUBROUINE PLMODE(LMOD)

Returns current display mode

LMOD - display mode (=IAXES), LOGICAL (O)

- -

SUBROUTINE PLSET(ILOG)

Makes a conditional change of display mode if a visual display mode is present

ILOG - logical mode flag (I)
=.TRUE. : erases the display and sets plot mode
=.FALSE.: erases the display and changes to text mode, provided that plot mode was set and the terminal is a visual display unit

- -

SUBROUTINE EJECT(LUN)

Performs page ejection or corresponding operation (e.g. erase for a visual display unit)

LUN - logical unit number (I)

- -

SUBROUTINE RESET

Resets the cursor to the text position on the display, provided that LDIS.ne.LTO (See Appendix G, COMMON /DEVICE/.)

- -

SUBROUTINE IWRITE(LUN,NLINE)

A call to SUBR. IWRITE is compulsory when the user wants to write a number of text lines on the display in order for the DISHDL to position the text cursor correctly.

E.g. to output the following two lines of text on the display:
RESULT
******

The corresponding WRITE-statement:

```
      WRITE(LDIS,1000)
 1000 FORMAT(' RESULT'/' ******')
```

Should be preceded by CALL IWRITE(LDIS,2), since FORTRAN's WRITE is not "felt" by the DISHDL.

The result of this is that the cursor will be placed after the text (i.e. to the right of it or at the beginning of the next line, depending on DISHDL's operating mode).

LUN - logical unit number (I)
NLINE - desired number of lines, NLINE = 0 forces DISHDL to start on a new line (I)

13) Appendix F - Text String Format

Since FORTRAN IV has no data type TEXT, it has been necessary to define rules for handling text strings in INTRAC.

a) Text strings are REAL, containg four characters per variable.

b) Names begin with a letter followed by an optional mixture of alphanumeric characters, the maximal number of characters being eight.
A name thus occupies two real variables.

c) Delimiters consist of one non-alphanumeric character, thus occupying one real variable.

d) A text string should be right filled with spaces.

Examples

The strings 'LUND', 'MONKEY' and '*' may be declared as:

```
DIMENSION HLUND(2), HMONK(2)

 DATA HLUND, HMONK, HSTAR
1   /4HLUND,4H    , 4HMONK,4HEY   , 4H*   /
```

e) String assignments and comparations are carried out by means of the INTRAC routines HSTORV and LCOMPV.

(I) and (O) denote input and output arguments, respectively.

- -

```
SUBROUTINE HSTORV(HSOUR,HDEST,NR)
```

HSOUR - source string, vector size (NR) (I)
HDEST - destination string, vector size (NR) (O)
NR - string size expressed in terms of real variables, i.e. the number of characters in the strings is obtained as 4*NR (I)

- -

```
LOGICAL FUNCTION LCOMPV(HSTR1,HSTR2,NR)
```

LCOMPV - assumes the value .TRUE. if the strings HSTR1 and HSTR2 are equal, otherwise .FALSE.

HSTR1,2 - strings to be compared with each other, vectors size (NR) (I)

NR - string size expressed in terms of real variables (I)

Note

Since the routines HSTORV and LCOMPV are capable of handling strings of arbitrary length, the restrictions on string lengths in b) and c) need only be considered when the application routines exchange data with INTRAC (See Appendices B and C).

14) Appendix G - INTRAC's Data Areas

(I) and (O) denote input and output arguments, respectively.

a) COMMON /COMINF/

Contains a few parameters to be used by the application routines.

MACLOG - MACRO operation mode (I)
= -1: current operation mode is:
'Define a MACRO but do not execute it'
MACLOG assumes other values as well, but these are of no interest to the application programmer

IERR - global error indicator (O)
= 0: no error
= 1, 2, .. : error cases
the error cases are assigned by the application routines, but
IERR is cleared internally by INTRAC

ICNR - pointer to the table of application commands (I)
used by the main program to pass control to the appropriate application routine
ICNR = 0 indicates data line/empty line

NRL - number of arguments to the left of the left arrow, if SUBR. INTINI was called with ARRLG = .FALSE. , (See Appendix A.)
else NRL = the total number of arguments in the command line (I)

NRR - number of arguments to the right of the left arrow, provided that SUBR. INTINI was called with ARRLG = .FALSE. , else NRL always = 0 (I)

Note

- /COMINF/ must reside in memory throughout the entire execution of the program package

b) COMMON /DEVICE/

Contains all the logical I/O numbers for the application routines.

LKB - Keyboard input

LTP - Tele printer output (= LTO normally)

LLP - Line printer (listing device)

LDIS - Display output (If no display present, set LDIS = LTO)

LTO - Standard terminal output

LPLOT - Put LPLOT non-zero, if plotting on the the device associated with LDIS is possible, otherwise put LPLOT = 0

LXXX - Not used

LDK1,2,3,4
Used for mass storage I/O

Note

- /DEVICE/ is READ-ONLY from the programmer's point of view, and it must reside in memory throughout the entire execution of the program package.
- /DEVICE/ may be initialized by means of a BLOCK DATA program
- LPLOT is no logical unit number, but merely a flag indicating whether the terminal is a visual display unit or not
- logical unit numbers for MACRO handling are provided by a call to SUBR. INTINI (See Appendix A.)

c) COMMON /GLOBAL/

Holds the global variables

MINPNT - Number of reserved global variables (I)

IPNT - Total number of global variables that have been assigned values, initially

IPNT = MINPNT (I/O)

MAXPNT - Max number of global variables that may be allocated (I)

G - Matrix to hold the global variables, dimensioned (7,MAXPNT) (I/O)

G(1,j),G(2,j)
the j:th global variable's identifier
(See Appendix F.)

G(3,j),G(4,j)
the j:th global variable's extension,
in the same format as the identifier

G(5,j)
the j:th global variable's type
=1: Name, (See Appendix F.)
=2: Integer
=3: Real
=4: Delimiter

G(6,j)
value of the j:th global variable,
or, if identifier, its four leftmost characters

G(7,j)
holds, in case of identifier, the rightmost four characters of the j:th global variable's value (See Appendix F.)

Notes

- /GLOBAL/ may be initialized by a BLOCK DATA program
- /GLOBAL/ must reside in memory throughout the entire run of the program package
- The global variables should be accessed via the routines in Appendix C, not via direct references to /GLOBAL/

Example

Initialize /GLOBAL/ with three reserved variables:

```
NPLX. = 123
```

```
DELTA.SAMPLE = 3.2
DEVICE.LISTING = LP
```

and a total of 60 global variables:

Table 1-2. 2

```
C GLOBIN
C
  BLOCK DATA
C
  COMMON /GLOBAL/ MINPNT,IPNT,MAXPNT,G(7,60)
C
  EQUIVALENCE (I51,G(5,1)), (I52,G(5,2)),
 1            (I53,G(5,3)) (I61,G(6,1))
C
  DATA MINPNT,IPNT,MAXPNT
 1    /3,3,60/
C
  DATA G(1,1),G(2,1),G(3,1),G(4,1),I51,I61
 1    /4HNPLX,4H    ,4H    ,4H    ,2,123/
C
  DATA G(1,2),G(2,2),G(3,2),G(4,2),I52,G(6,2)
 1    /4HDELT,4HA   ,4HSAMP,4HLE  ,3,3.2/
C
  DATA G(3,1),G(3,2),G(3,3),G(3,4),I53,G(6,3),G(7,3)
 1    /4HDEVI,4HCE  ,4HLIST,4HING ,1,4HLP  ,4H    /
C
  END
```

Note

The above BLOCK DATA program enables data of different types to be stored at the same memory locations and it does not make use of the numbers of machine cells allocated to scalar data items of each type.

d) Other COMMONs

INTRAC has a number of internal COMMONs. These must reside in memory throughout the entire program run, and they may not be be accessed by anyone else then INTRAC.

They are all declared and initialized in the standard BLOCK DATA program INTTAB.