CERN Program Library Long Writeups L210

COMIS

Compilation and Interpretation System

Reference Manual

Version 2.

Application Software and Databases

Computing and Networks Division

CERN Program Library Office
CERN-IT Division
CH-1211 Geneva 23
Switzerland
Tel. +41 22 767 4951
Fax. +41 22 767 8630
Internet: cernlib@cern.ch

Contact Person: Vladimir Berezhnoi /EP Vladimir.Berejnoi@cern.ch
Cocumentation consultant: Michel Goossens /CN (goossens@cern.ch)

Edition – August 1998

Preliminary remarks

This manual serves at the same time as a **Reference manual** and as a **User Guide** for the `comis` system. Historically the following IHEP (Institute for High Energy Physics, Moscow Region, Russia) people have worked on the `comis` system: V. Bereshnoi, S. Nikitin, Y. Petrovych and V. Sikolenko. At CERN René Brun has contributed to the development of the system.

In this manual examples are in `monotype face` and strings to be input by the user are underlined. In the index the page where a routine is defined is in **bold**, page numbers where a routine is referenced are in normal type.

In the description of the routines a * following the name of a parameter indicates that this is an **output** parameter. If another * precedes a parameter in the calling sequence, the parameter in question is both an **input** and **output** parameter.

This document has been produced using LaTeX[1] with the `cernman` style option, developed at CERN. A compressed PostScript file `comis.ps.Z`, containing a complete printable version of this manual, can be obtained from any CERN machine by anonymous ftp as follows (commands to be typed by the user are underlined):

```
ftp asis01.cern.ch
Trying 128.141.201.136...
Connected to asis01.cern.ch.
220 asis01 FTP server (Version 6.10 ...) ready.
Name (asis01:username): anonymous
Password: your_mailaddress
230 Guest login ok, access restrictions apply.
ftp> cd cernlib/doc/ps.dir
ftp> get comis.ps.Z
ftp> quit
```

[1]Leslie Lamport, "LaTeX, A Document Preparation System, Addison and Wesley, 1986

Table of Contents

Chapter 1: Introduction

1.1 comis - What is it ?

comis (a COMpilation and Interpretation System) is a Fortran interpreter. With comis you can interactively define, edit and execute any Fortran-like routines (e.g., consisting of one simple output statement), without recompiling and relinking. A small user interface system is part of comis, and an interface with the local editor is also provided.

comis is one of the key components of the paw system and is currently implemented on IBM VM/CMS, VAX/VMS, Apollo (Aegis and Unix), IBM RS/6000, DEC Station 3100, Silicon Graphics, Sun, HP/UX and MSDOS. It can also be used as an interactive compiler and interpreter from inside a Fortran 77 program. If the source of your program to be compiled is in a file IFORT.FOR then you should use the following code:

Example of calling comis from within Fortran

```
    PROGRAM MAIN
    CALL HLIMIT(10000)
    CALL CSINIT(2000)
  1 CALL CSPAUS('IFORT')
    GO TO 1
    END
$ Fortran IFORT
$ link IFORT,'LIB$'
$ run IFORT
```

You can run the simple example below by typing the lines below in response to the comis prompt “CS>”:

```
  DO 1 X=3.14, -3.14,-0.7
  PRINT *,X,SIN(X)
1 CONTINUE
  END
```

or the same code in short notation as: `DO X=3.14, -3.14,-0.7 PRINT *,X,SIN(X) OD #`

The sequence `STOP END` will cause exit from comis.

1.2 comis at a glance

comis is a library of routines written in Fortran 77. A few routines exist also in assembler on IBM. comis has two major components:

- A set of subroutines to interactively compile and execute Fortran-like routines.
- A package of subroutines, handling the interface between the user's routines compiled with the normal Fortran compiler and those compiled interactively.

The comis system supports the following facilities:

– The interpretation of `comis` routines entered either interactively or through text files.
– The ability to call any user routine from the `comis` program and vice versa.
– The access to user COMMON blocks data from the `comis` program.
– The ability to redefine any `comis` program.

1.3 `comis` with PAW

A high level interface to `comis` is provided by `paw`. The PAW user creates and edit text files containing Fortran code using the local editor. These files are automatically interpreted and executed by the PAW commands without typing any `comis` commands. From the PAW level `comis` may be invoked in one of the three following ways:

– Using a `comis` command. Example:

```
PAW > comis
CS >   do x=1.,10.
MND>      sq=sqrt(x)*10.
MND>      print *,x,sq
MND>   od
MND>   end
CS > quit
PAW >
```

– Using an APPLication command. Example:

```
PAW > Vector/Create Y(10) r 1 2 3 4 5 6 7 8 9 10
PAW > APPLication comis QUIT
CS >     subroutine demo
FSD>     vector y, x(10)
FSD>     do 10 i=1,10
FSD>       xx=i
FSD>       x(i)=y(i)*sqrt(xx)*10.
FSD>  10 continue
FSD>     end
CS >     call demo
MND>     end
CS > quit
PAW> Vector/print X
```

– Using a CALL command: `CALL urout`.

  – Case 1:
    `urout` is a routine compiled and linked with `paw`, for example `HPRINT`. Then one can type `CALL HPRINT(10)`
  – Case 2:
    `urout` is the name of a file which can be edited interactively with the `paw` EDIT command. For example if the file `UROUT.FOR` contains:

```
      SUBROUTINE UROUT(N)
```

```
      SUM=0.
      DO 10 I=1,N
        SUM=SUM+I
 10   CONTINUE
      PRINT *,SUM
      END
```

Then one can type `CALL UROUT.FOR(10)`.

PAW users are recommended to use the standard Fortran notation and to read the following chapter to be aware of the few `comis` restrictions.

Chapter 2: The `comis` compiler

This chapter gives a brief description of the `comis` syntax. Only the main differences between the `comis` syntax and standard Fortran 77 will be discussed.

The `comis` system language is an almost full implementation of Fortran 77. As the user is working in a realtime environment, some simplifications of the language syntax (in addition to the standard one) are also available: short forms for key-words (`CHAR` instead of `CHARACTER` and so on) and a free source code format.

The main differences between the `comis` syntax and standard Fortran77 are:

- `comis` does not accept:
  - character functions;
  - statement functions;
  - INTRINSIC statements;
  - ENTRY statements;
  - BLOCK DATA statements.
- `comis` does not allow a character type variable to be equivalenced with another character variable.
- A type declaration statement, which define the data type of common block name's, must precede the COMMON statement.
- The logical operators .EQV. and .NEQV. are not included in `comis`.
- `comis` does not allow a character expression and an alternate return specifiers as actual arguments.

2.1 The source format

The `comis` source code is essentially free format. Extensions are:

- Statements may start at any position.
- More than one statement may be given per source line. In this case they should be separated by blank(s), or optionally by ';'.
- A statement may occupy more than one source line without any special marks.
- Source lines may be up to 80 characters long. Thus, source line numbering in columns 73-80 is not available.
- A symbolic name is a string of letters, digits and underscore (_). The symbolic name may have any length, but significant characters are the first eight only.

Note

- Blanks are meaningful in `comis`.
- Comment lines are marked by `'*'` or `'C'` in column one. An exclamation mark (!) starts an inline comment, except when it appears as the first non-blank character on a line.
- Some extentions may cause a conflict with the conventions of Fortran 77 (see the `!`FORTRAN directive in Chapter 4).

2.2 Data types and constants

The following data types are supported:

```
INTEGER, REAL, DOUBLE PRECISION, LOGICAL, CHARACTER, COMPLEX
```

There are seven types of constants: Integer, Real, Double precision, Logical, Complex, Character string and Hollerith.

Constants may be placed directly in the source code, or they can be accessed by names assigned to them with PARAMETER statements.

2.3 Declarations

Data type declarations

```
full form              short form
---------              ----------
INTEGER                INT
REAL                   REAL
DOUBLE PRECISION       DOUBLE
LOGICAL                LOG
CHARACTER              CHAR
CHARACTER*(*)          CHAR*(*)
CHARACTER *n           CHAR *n
COMPLEX                COMPLEX
```

DIMENSION statement

In F77 the upper bound of the last dimension may be specified as '*'. `comis` does not allow it.

EQUIVALENCE statement or EQU

The short form of this statement is EQU.

In F77 a character type variable may be equivalenced with another character variable. `comis` does not allow this.

DATA statement

In F77 named common blocks may be initialized by means of a DATA statement in a BLOCK DATA subprogram. `comis` does not support BLOCK DATA subprogram, but DATA statement may be used to initialize common blocks in any `comis` routines.

Note that `comis` does not support an implied DO list in a DATA statement to initialize the elements of an array.

IMPLICIT statement or IMP

This statement is the same as the standard one. `comis` supports the `IMPLICIT NONE` statement also.

SAVE statement

All data are saved in `comis` .

In F77 a named common block name (preceded and followed by a slash) is allowed in SAVE lists. `comis` does not allow this.

INTRINSIC statement

This statement is at present not supported by `comis`.

COMMON statement or COM

This statement is the same as the standard one.

The COMMON statement defines the position and the order of the user's variables and/or arrays in storage . The common block with the same name must be present in the application program (see also section 3.3).

USE statement

```
USE list
```

In any `comis` routines one can have access to variables' names, defined in COMMON declarations in the main `comis` program (see section 2.5.1). This can be done by the USE statement, in which one should declare the names of a corresponding COMMON block. The name `BLANK$` should be declared in the list for access to variable names of a non labeled COMMON.

VECTOR statement

This statement can be used in a `paw` version of `comis` only.

The declaration `VECTOR vector_name` may be used inside a `comis` routine to address a `kuip` vector. If the vector does not exist, it is created with the specifications provided by the declared dimension. The vectors `x` and `y` defined in the example on page 2 show how this works.

2.4 Other language elements

2.4.1 Arithmetic, logical and character string expressions

The following operations are available:

```
 +  -  *  /  **  //
.LT.   or   <
.LE.   or   <=
.EQ.   or   ==
.NE.   or   /=
.GE.   or   >=
.GT.   or   >
.OR.  .AND.  .NOT.
```

The logical operators `.EQV.` and `.NEQV.` are not included in `comis`.

2.4.2 Control statements

Loops

DO loops with index and DO WHILE (logical expression) constructions are provided. Loops may have the DO-ENDDO or DO-OD forms or may be labeled in a Fortran-like manner. For the indexed loops the index may be of an integer or a real type only.

2.4.3 IF statements

comis supports both the standard F77 syntax and a form ending in FI for the IF statement.

```
IF...[ELSEIF]...[ELSE]...ENDIF
```

or

```
IF...[ELESIF]...[ELSE]...FI
```

GOTO statements or GO

Jumps to constant labels, assigned GOTOs in connection with ASSIGN statements, and computed GOTOs are provided.

F77 allows the use of a list with optional labels on an assigned GOTO statement; this facility is not supported in comis.

CALL statement

This statement calls a user or a comis routine.

comis supports three forms of the CALL statement:

```
1.   CALL subr_name [( arg_list)]
2.   subr_name ( arg_list)
3.1  subr_name  arg_list
3.2  subr_name;
```

where subr_name is the name of the subroutine which may be:

- an already defined comis routine;
- a user supplied routine: if no comis routine with the specified name is found, then comis will search for a user routines with that name (see also section 3.3).

The semi-colon character ";" is significant in case 3.2 above!

INCLUDE directive

An INCLUDE directive is available and has one of the following forms:

```
1.   INCLUDE 'name'
2.   INCLUDE  name
```

When `name` is given between quotes (case `1.` above), then the name is taken literally, while in case `2.` the string `name` is converted to uppercase.

Note that `comis` does not allow recursive INCLUDE directives.

2.4.4 Other statements

`CONTINUE`	the short form of this statement is CON
`RETURN`	the short form of this statement is RET
`QUIT`	return from `comis` to application program
`END`	the short form of this statement is "`#`"

2.4.5 Input/output statements

The full set of F77 input/output statements is implemented. `comis` provides four types of input/output statements:

- Sequential input/output statements
- Direct access input/output statements
- List directed input/output statements
- Internal data set input/output statements.

A list of `comis` input/output source statements follows:

```
OPEN, WRITE, PRINT, READ, ENDFILE, BACKSPACE, REWIND, CLOSE, INQUIRE.
```

The `comis` extentions are:

- INPUT statement - input from a terminal in free format.
- TYPE statement - output to a terminal in free format.

FORMAT statement

The FORMAT statement has the form:

```
FORMAT( f1 [,f2 [,...,fn]])
```

where `f1,f2,...,fn` are format codes.

The current version of `comis` does not support `kP, S, SP, SS, BN` and `BZ` edit descriptors.

The length of a format specification cannot exceed 256 characters.

INQUIRE statement

The INQUIRE statement has one of the forms:

```
      INQUIRE(dsns,iflist)
      INQUIRE(  ns,iflist)
```

`dsns` data set name specifier of the form `FILE= dsn`;

`ns` data set reference number of the form `UNIT= ns`;

`iflist` list as defined in Fortran 77.

You cannot omit `UNIT=ns` on a `comis` inquire statement.

INPUT statement or INP

The INPUT statement has the form

```
INPUT list
```

This statement inputs values from the terminal; the user is prompted with the list element name. If the user press a carriage return key, the current value is not changed, otherwise the constant typed in becomes the new value of a list element. If a list element value is a constant then this constant is simply typed out.

TYPE statement

The TYPE statement has the form

```
TYPE list
```

This statement types values of the lists elements in free format.

Example:

If the user runs the next simple `comis` program:

```
      CHARACTER A*4
      J=1
      INPUT 'CHARACTER *4',A, 'J HAS VALUE 1',J,Z
      TYPE A,J,Z
      END
```

The dialogue will be

```
'CHARACTER*4'
*C  A=abcd<cr>
'J HAS VALUE 1'
*I  J=<cr>
*R  Z=3.14<cr>
*T  A = ABCD     J = 1     Z = 3.140000
```

2.5 Programs

2.5.1 Main program

The `comis` main program syntax is:

```
[PROGRAM name]
[Declaration statements]
[Executable statements]
END
```

The main program is executed immediately whenever its definition is finished. The `comis` system "remembers" all declarations of common blocks issued in the main program. These declarations should not be repeated each time the main program is redefined.

Moreover these declarations are valid for other subprograms through the USE statement (see section 2.3) This does not mean that you cannot enter the new COMMON declarations during the main program redefinition.

2.5.2 Functions and subroutines

The colon character "`:`" is the short form of key words FUNCTION and SUBROUTINE. Actual arguments may be constants, variables, arithmetic expressions, arrays, arrays elements or subroutines names.

`comis` does not allow a character expression and alternate return specifiers as an actual arguments.

The `comis` extensions are:

– The number of routine's arguments may vary and can be obtained inside the called routine.
– An argument can be a sequence of statements enclosed in square brackets.
– An argument can be omited.

For details see Appendix B.

`comis` does not accept statement functions, functions of a character type and an ENTRY statement.

The full set of intrinsic functions supplied in the `comis` system excepting `CHAR, LLT, LLE, LGT, LGE`.

Chapter 3: Calling sequences

This chapter describes the `comis` interface with a user and his application program.

3.1 Initialization

The first action to perform is to initialize the `comis` system variables by calling:

CALL **CSINIT** (NWORDS)

where the input parameter `NWORDS` is the size of the system common block

```
COMMON /COMIS/CS(NWORDS)
```

By default `NWORDS=2000`, which is usually sufficient.

3.2 Entry to `comis`

The three routines described in this section can be used for this purpose:

CALL **CSPAUS** (ENTRY_PROMPT)

where `ENTRY_PROMPT` is a character string. In this case `comis` types the `ENTRY_PROMPT` and the dialogue is started.

CALL **CSTEXT** (ENTRY_PROMPT,TEXT)

where input parameters `ENTRY_PROMPT` and `TEXT` are characters strings. In this case `comis` interprets the string "`TEXT`", types `ENTRY_PROMPT` and starts the dialogue.

CALL **CSEXEC** (TEXT,IERR*)

where input parameter `TEXT` is a character string. In this case `comis` interprets the given string "`TEXT`" only and control is returned to the calling routine. The output parameter `IERR` is zero when no errors are encountered, non-zero otherwise.

During the dialogue session `comis` gives the standard prompt "`CS>`" and the user can:

- enter the system directives(see Chapter 4);
- enter the function or the subroutine definition;
- enter the `comis` main program.

All system directives are executed immediately and the prompt "`CS>`" is displayed again.

If the FUNCTION or the SUBROUTINE statement is entered, `comis` changes his prompt to "`FSD>`" until the END statement completes the routine definition. An intermediate code is stored in the internal buffer and `comis` gives the prompt "`CS>`".

During the main program definition `comis` uses the prompt "`MND>`". The `comis` linker is invoked automatically if the compilation was error-free. The linker tries to resolve references in the order: `comis` routines; user's routines. After this stage the main program executes and `comis` types the entry prompt and gives the standard prompt "`CS>`".

A comis built-in editor (see Chapter 5) is automatically invoked when a syntax error is detected. The command "E" of the editor causes the recompilation of the text edited and then the dialogue is continued. The command "Q" of the editor causes the current routine definition to be skipped and comis gives the prompt "CS>".

The exit from the comis dialogue session is performed by the RETURN statement in the main program or by an "empty" (END or #) main program definition. For exiting from any comis routine to an application program the QUIT statement can be used.

3.3 Calling the user routines

In order to invoke the routines compiled by the user comis has to know the address of the called routine. The location of the routine can be passed to the comis interpreter through a call to subroutine CSEXT:

```
CALL CSEXT  ('name1.type,...,nameN.type#',name1,...,nameN)
```

where `name1,...,nameN` should be declared as EXTERNAL in the Fortran program. Possible values for the .type specifier are:

- D for a double precision function;
- I for an integer function;
- L for a logical function;
- R for a real function;
- S for a subroutine;
- X for a complex function.

If the type specifier is omitted, then type subroutine ('.S') is assumed.

Data transmission between Fortran and comis routines is done in the usual way using routine parameters and through COMMON blocks. To handle COMMON blocks the interpreter has to know the address of the first element of each such block. This is specified by using subroutine CSCOM as follows:

```
CALL CSCOM  ('name1,...,nameN#',FE1,...,FEN)
```

`FE1,...,FEN` are the first elements of the COMMON blocks with names `name1,...,nameN`. The name `$BLANK` should be used for the blank COMMON.

The addresses of COMMON blocks which contain character data should be specified with a call to routine CSCOMC, whose calling sequence is similar to CSCOM.

3.4 Calling comis routines from a user program

The user can call a comis routine from a Fortran77 program using the routine's name or address. The second call is faster, but it requires the preliminary calculation of the address by

```
JADP=CSADDR('NAME')
```

Arithmetic parameters

This section describes routines and functions which can be used for calling comis subroutines and functions which have only arithmetic parameters.

Calling a `comis` subroutine

CALL **CSCALL** ('NAME',NPAR,P1,P2,...) and CALL **CSJCAL** (JADP,NPAR,P1,P2,...)

where

`NAME`	is the name of the called `comis` routine;
`JADP`	is the address of the called `comis` routine;
`NPAR`	is the number of arguments;
`P1,P2,...`	are the routine's actual arguments.

Using integer `comis` functions

I = **CSICAL** ('NAME',NPAR,P1,P2,...) and I = **CSIJCL** (JADP,NPAR,P1,P2,...)

Using real `comis` functions

R = **CSRCAL** ('NAME',NPAR,P1,P2,...) and R = **CSRJCL** (JADP,NPAR,P1,P2,...)

To speed up execution three special functions are available for the case of routines with one, two and three arguments:

```
R = CSR1FN(JADP,P1)
R = CSR2FN(JADP,P1,P2)
R = CSR3FN(JADP,P1,P2,P3)
```

Using double precision `comis` functions

D = **CSDCAL** ('NAME',NPAR,P1,P2,...) and D = **CSDJCL** (JADP,NPAR,P1,P2,...)

Using complex `comis` functions

Cx = **CSCCAL** ('NAME',NPAR,P1,P2,...) and Cx = **CSCJCL** (JADP,NPAR,P1,P2,...)

General parameters

This section describes routines and functions which can be used for calling `comis` subroutines and functions which can have any type of parameters.

Calling a `comis` subroutine

CALL **CSSUBR** (STR,P1,P2,...) and CALL **CSJSUB** (JADP,STR1,P1,P2,...)

where `STR` and `STR1` are character strings which specifies the name of called `comis` routines and the type of each parameters in argument list.

Using integer `comis` functions

```
I = CSIFUN (STR,P1,P2,...)   and   I = CSIJFN (J,STR1,P1,P2,...)
```

Using real `comis` functions

```
R = CSRFUN (STR,P1,P2,...)   and   R = CSRJFN (J,STR1,P1,P2,...)
```

Using double precision `comis` functions

```
D = CSDFUN (STR,P1,P2,...)   and   D = CSDJFN (J,STR1,P1,P2,...)
```

Using complex `comis` functions

```
Cx = CSCFUN (STR,P1,P2,...)   and   Cx = CSCJFN (J,STR1,P1,P2,...)
```

The parameters `STR` and `STR1` in these routines have the forms

`STR` `name` (parameter type description)

`STR1` (parameter type description)

where "`name`" is the name of the `comis` routine which is called; "`(parameters types description)`" specifies the type of each parameter in the argument list.

Examples:

```
STR = 'NAME(I,R,D,C,E,*12)'

where  I  stands for INTEGER or LOGICAL
       R         for REAL
       D         for DOUBLE PRECISION
       C         for COMPLEX
       E         for EXTERNAL
       *12       for CHARACTER *n
```

If a double precision `comis` function is declared with:

```
      FUNCTION CSDPF(R,D,C,I)
      DOUBLE CSDPF,D
      REAL R  INTEGER I
      CHARACTER *(*)C
      .....
      END
```

then the call from the user's routine may look like:

```
CHARACTER *12 TEXT
DOUBLE PRECISION D,DP,CSDFUN
.....
D = CSDFUN('CSDPF(R,D,*12,I)', R, DP,TEXT,10)
```

3.5 XYZ-expressions

`comis` supports a special class of XYZ-expressions. The XYZ-expression uses only the variable names X and/or Y and/or Z, e.g.

```
X+Y+Z  or X**2+Z**2  or SIN(Y)**2+COS(Y)**2
```

An XYZ-expression can be translated by

```
CALL CSEXPR  (XYZexpr,JADDR)
```

where `XYZexpr` is an argument of type character, whose contents is an XYZ-expression, `JADDR` is the address of an XYZ-expression stored in the `comis` internal buffer.
The XYZ-expression can be evaluated by

```
VAL = CSRJCL(JADDR,NPAR,ARG1,ARG2,ARG3)
```

using the standard way to call a `comis` real function.
If X is omitted in the XYZ-expression then `ARG1` must be a dummy argument, if Y is omitted then `ARG2` must be a dummy argument, e.g.

XYZ-expression	can be evaluated by
`SIN(Y)/Y`	`V = CSRJCL(JADDR,2,dummy,Y)`
`X+Z`	`V = CSRJCL(JADDR,3,X,dummy,Z)`
`Z**2`	`V = CSRJCL(JADDR,3,dummy,dummy,Z)`
`X**2`	`V = CSRJCL(JADDR,1,X)`

Chapter 4: System directives

Some of the directives listed below have the optional parameter “`LUN`”. It is the logical unit number for input/output streams. If this parameter is omitted, the default value is used (see appendix A). All `comis` directives start with a special character ”`!`” to avoid a conflict with user defined routines.

!HELP

This directive ouputs to terminal short help about directives.

!FILE `[lun,] file_name`

By default the system input device is the terminal. This directive sets the input stream to the file `file_name`. The same action can be performed in a user routine by calling:

`CALL` **CSOFIL** `(lun,'file_name')` and `CALL` **CSFILE** `('file_name')`

The system switches to the terminal again when the file is read or an `!EOF` directive is reached.

!EOF

This directive closes the input file. The next string will be accepted from the terminal.

!LOGFILE `[lun,] file_name`

The transcript of the interactive session will be collected in the file `file_name`. This file can be used in the `FILE` directive later (to repeat the same dialogue session, for example). The same action can be performed in a user routine by calling:

`CALL` **CSOLOG** `(lun,'file_name')` and `CALL` **CSLOG** `('file_name')`

!FORTRAN

This directive sets the mode of compilation to “Fortran”. This mode should be selected avoid syntactic conflicts when you want to process standard Fortran sources with the `comis` compiler. After getting this directive the compiler treats a ’C’ character in the first column as a comment mark and every character in the sixth column as a continuation mark (unlike `comis`’ free format syntax). This mode is the default.

!COMIS

This directive sets the “`COMIS`” mode of compilation.

!SHELL `command`

passes an operand line to the operating system for command processing. Does not cause a break.

!SHOW `memory`

This directive shows the `comis` internal memory usage.

!SHOW `routines`

This directive shows a list of routines currently known to `comis`.

!SHOW `commons`

This directive shows list of common and global blocks currently known to `comis`.

!SHOW `names common_name`

This directive shows declaration of common or global block with name `common_name`.

!REMOVE `cs_routine_name`

This directive removes from the internal `comis` memory the comis routine with name `cs_routine_name`.

!CLEAR

This directive clears the internal `comis` memory: it removes all comis routines and all common/global blocks declarations.

!CHECKB

When the directive `CHECKB` is given `comis` will check during routines interpretation that the evaluated result of a array's subscript expression is greater than or equal to the corresponding lower dimension bound and does not exceed the corresponding upper dimension bound.

!NOCHECKB

The directive `NOCHECKB` causes no check to be made during routines interpretation (default is `CHECKB`).

!PARAM

The directive `PARAM` causes the `comis` compiler to insert additional code to provide all facilities for the treatment of actual arguments.

!NOPARAM

With the directive `NOPARAM` you cannot obtain the argument's text and the Algol like manner of argument processing is not available (default is `NOPARAM`).

Chapter 5: Built-in editor

comis provides a built-in editor which is automatically invoked by the interpreter when an error is detected. The comis built-in editor is similar to the VAX line-mode EDIT/EDT. A line can be referenced by its line number. The INSERT and DELETE commands change a the line numbering. To operate on a set of lines it is possible to specify a line range in the following forms:

N1:N2	specifies the set of lines from N1 to N2, where N1<N2 or N1>N2.
N	specifies N-th line from the beginning of the file.
+N	specifies +N-th line from the current pointer position.
-N	specifies -N-th line from the current pointer position.

The character "F" is recognized as the first line of the routine and the character "L" is recognized as its last line; "W" is recognized as "F:L".

5.1 Commands explanation

T [range]

This command types the lines in the given range. If range is omitted the current line is typed. The leading T can be omitted and only the range specified. In response to a <CR> the next line is typed.

S/old/new/ [range]

This command substitutes the "old" text string by the "new" one for all lines in the range.

D [range]

This command deletes the set of lines specified (the line numbers will be changed).

I [line_number]/ line1<CR> line2<CR>...<lineN>/

This command inserts the given text lines after the line specified by the line_number. To insert new lines at the very beginning you should specify line_number=0 (the line numbers will be changed).

EXIT

This command cause exits from the editor. The text is processed by the comis compiler automatically.

QUIT

This command cause exits from the editor without any compilation.

EDIT

comis invokes the local editor of the operating system. After edit session control is returned to the built-in editor.

HELP

This command types HELP information about the editor commands.

Appendix A: Input/Output unit numbers

For input/output streams `comis` uses the following channels by default:

channel	**usage**
`lunsn =11`	for the system's needs.
`lunfil=12`	for input source code under the `!`FILE directive.
`lunlog=13`	for output into the LOG-file.
`lunmap=14`	for access to the MAP-file.
`lunedt=15`	for the local editor

One can change the default values by

```
CALL CSSETL (lunsn,lunfil,lunlog,lunmap,lunedt)
```

before `comis` system initialization.
In the case of PAW, these defaults have been changed to 81,82,83,84,85 respectively.

Appendix B: Obtaining the values of actual arguments

A group of routines to handle actual arguments is provided by `comis`. These routines allow the user

- to obtain the number of actual arguments;
- to obtain the type and the mode of each argument;
- to get the value or execute argument's code.

These routines can be used at the `comis` level or at the Fortran one, if the user's routine was called from the `comis` level.

The number of actual arguments, `NPAR`, can be obtained with:

```
NPAR  = CSNPAR  (DUMMY)
```

The type and mode of the K-th argument can be obtained by:

```
IT = CSKPAR  (K,MODE)
```

where

```
IT=0    unknown               MODE=0   unknown
   1    integer                    1   expression
   2    real                       2   constant
   3    character                  3   variable
   4    logical                    4   array element
   5    double precision           5   function call
   6    Hollerith                  6   array name
                                   7   external
                                   8   [sequence of statements]
                                   9   omitted (empty) argument
```

The value of the `K`th argument or the execution of its code is obtained by:

```
CALL CSCPAR  (K)
```

In fact this subroutine fills the system COMMON block `CSWPAR`, whose definition is described below:

```
DOUBLE PRECISION DVPAR
COMMON/CSWPAR/LORN,IREP,NPAR,ITPAR,MDPAR,JRESP,JCHPAR,NCHPAR,IVPAR,RVPAR,DVPAR,JCHVP,NCHVP
```

where

`LORN` is set by the user. It specifies the manner of argument processing:

- `0` ordinary Fortran-like processing, i.e. the value of the argument is taken immediately using the argument address.
- `1` ALGOL-like processing "by name": the argument is re-evaluated as if it was in the routine's text at that point.

`IREP` the reply word: 1 - o.k.; less or equal 0 - error.

`NPAR`	the number of actual arguments.
`ITPAR`	the type of the argument.
`MDPAR`	the mode of the argument .
`JRESP`	the address of the argument's value.
`JCHPAR`	the 'character' address
`NCHPAR`	the length of the argument's text.
`IVPAR`	the integer value of the argument.
`RVPAR`	the real value of the argument.
`DVPAR`	the double precision value of the argument.
`JCHVP`	the 'character' address
`NCHVP`	the length of the character type argument.

To obtain the value of the integer type argument the next integer function may be used:

```
I = CSIPAR  (K)
```

To obtain the value of the real type argument the next function may be used:

```
R = CSRPAR  (K)
```

These two latter functions call subroutine CSCPAR internally.

Examples:

Using an Algol-like manner of argument processing.

```
FUNCTION SUM(P1,I,N)
SUM=0.
DO I=1,N  SUM=SUM+CSRPAR(1) OD
END
```

The `comis` main program

```
.....
LORN=1  I=1  S=SUM(A(I),I,N) END   produces   S=A(1)+A(2)+ ... +A(N)
```

while the program

```
.....
LORN=1  K=1  S=SUM(A(I,K)*B(K,J),K,M) END  produces the  (I,J)th  inner product of  A  and  B.
```

The length of the K-th argument's text can be obtained by

```
L = CSLPAR  (K)
```

The K-th argument's text can be moved into a variable of character type by

```
CALL CSTPAR  (K,CHARVAR)
```

The code of the K-th argument with the mode ”sequence of statements” can be stored in the `comis` internal buffer as the `comis` routine with given name `NAME` by

```
CALL CSSPAR  CSSPAR(K,'NAME',IADP)
```

where output parameter `IADP` is the address of the `comis` routine.

Appendix C: `comis` source files and libraries

On the CERN machines, `comis` is provided in the context of PAW.

IBM

```
CERNLIB PAWLIB
LOAD user (NOAUTO
```

VAX

```
CERNLIB PAWLIB
LINK user,'LIB$'
```

Unix

```
f77 user.o `cernlib pawlib`
```

Apollo

```
ld user.bin `cernlib pawlib`
```

Index