Identification

describe

M. Padlipsky

Purpose

The describe command affords the console user quick reference to descriptions of system commands. Information available for each command - selectively or in toto - includes calling sequence, Multics System Programmers' Manual (MSPM) and/or Multics Users Manual (MUM) reference, argument descriptions, list o~~r~~f error conditions, and a brief statement of the command's purpose. Non-system ~~commands~~ procedures may also be dealt with by describe, if their authors furnish information segments according to the conventions established below; system commands must be furnished with such information segments.

Usage

```
     describe (comname -C- -R- -A- -E- -P-)
```

where comname is the name of the command to be described (in full or abbreviated form), and the optional arguments are as follows:

C	print calling sequence
R	print reference
A	print argument descriptions
E	print error information
P	print purpose

More than one optional argument may be given (in any order) at a given invocation of the command but the printed descriptions will not necessarily appear in the order dictated by the order of the optional arguments.

Method

Associated with each system command (e.g., describe) is an ASCII file (created by use of edit, BX.9.01) whose name is the commands' name concatenated with ".info" (e.g., describe.info). The contents of the file follow the format shown in the example below. The describe command treats an "info" file as a single character string and extracts the relevant section(s) for printing according to which optional argument(s) it has been called with, or simply prints the entire file if none of the optional arguments has been given. (Note that the "info" file must be known by both the full and the abbreviated version of the command name, as is the command; see also BX.0.01).

No adjustment to the user's searching rules is made. Thus, if a non-system procedure ~~command~~ is given as comname, and if a suitable file named comname.info is accessible to the user, describe will function normally.

Conventions

1. The "info" file contains fi~~v~~e sections, in the following order and with the following headings: Call, Ref., Arg., Errors, Purpose. [illegible] spell out

2. The sections of the "info" file are delimited by lines containing only an asterisk (*), with the heading of the following section appearing on the next line. No other occurrences of the sequence of characters "<NL> * <NL>" ~~new line - asterisk - new line~~ are allowed in the file.

3. In the calling sequence section, the following considerations apply: start new line
   a) Non-literal arguments are in lower-case letters. [space] b) Literal arguments are in upper-case letters, ~~or, if not capitalizable (e.g., numerals) are~~ (Literals are written in their literal form and are

indicated as such by "(lit)" at the beginning of their descriptions in the argument description section.) ~~In either case literals are in their correct form in and indicated by "(lit)" in~~

c) Optional arguments are surrounded by minus signs. In the event of exclusive choices of arguments, the calling sequence shows the arguments separated by vertical bars; e.g., ON|OFF.

d) Command language syntax is used for indicating lists; see BX.1.00

e) The full name of the command is shown in the calling sequence; the abbreviation is given on the final line of the Call section of "info" file (see below).

(although for describing describe is probably recursively silly)

Example

Describe's own "info" file would look like this:

*

Call: DESCRIBE (comname -C- -R- -A- -E- -P-)

Abbrev.: DSB

*

Ref.: MSPM BX.11.02, MUM Cx.xx.xx

*

Arg.:	$\overset{lc}{\cancel{C}}$omname	name of command to be described
	C	print calling sequence ~~only~~
	R	print references ~~only~~
	A	print argument description ~~only~~
	E	print error description ~~only~~
	P	print purpose ~~only~~

*

Error: ~~Missing file, access problems~~
~~from System.~~ "xSECTION NOT FOUND IN comname.INFO"
implies error in creation of "info" file.

*

Purpose: Print command description

*

Note that the delimiting asterisks appear around each section. The colons after the section headings are not scanned for, but may be used or not depending on the aesthetic inclinations of the creators of the various "info" files. Also up for aesthetic grabs is the issue of whether to give the heading a separate line.

Implementation

Figure 1 presents a block diagram of describe. Note that the command relies heavily on the strip library routine, BY.8.03. The logic is as follows: If no optional arguments are specified, call the print command (BK.9.02) for "comname.info" and return. Otherwise, proceed as follows:

To allow for more than one optional argument's having being specified, an array called mask is set up to contain the section heading for each requested section and a blank for each non-requested section, in order. The relevant "info" file is fo~~rme~~und by a call to generate_ptr ~~get_seg~~ (BY.), and the character count by a call to get_char_ct (BY.2.02). Then, a call is made to an internal routine called inner, so that the "info" file may be (more efficiently) treated as an adjustable character string rather than a varying one. Inner is passed a pointer to the "info" segment and a pointer to the mask array, with which information it can preform the remainder of describe's processing. For each non-blank member of the mask array, strip (BY.8.03) is invoked for the current remainder of the original string to extract that heading's section of the "info" file, the resulting string is printed, and the mask (i) is set to blanks. Provided that the "info" file was created in the proper order, the command is finished when the mask array has been looped through; however, to allow for possible mistakes, if any member of mask is not blank further processing is performed. This time, strip is invoked for the entire string for each heading not found previously. (This approach is not taken originally as it is inefficient, and unnecessary if the file is in order.) If any requested section is still not found, a message is printed to the effect "so and so SECTION NOT FOUND IN comname.INFO". (Other errors, especially missing "info" files or access problems, are left to be reflected by the system.)

~~need better statement~~

Other errors are not provided for and ~~and~~ are handled by the Shell as described in BX.2.00)

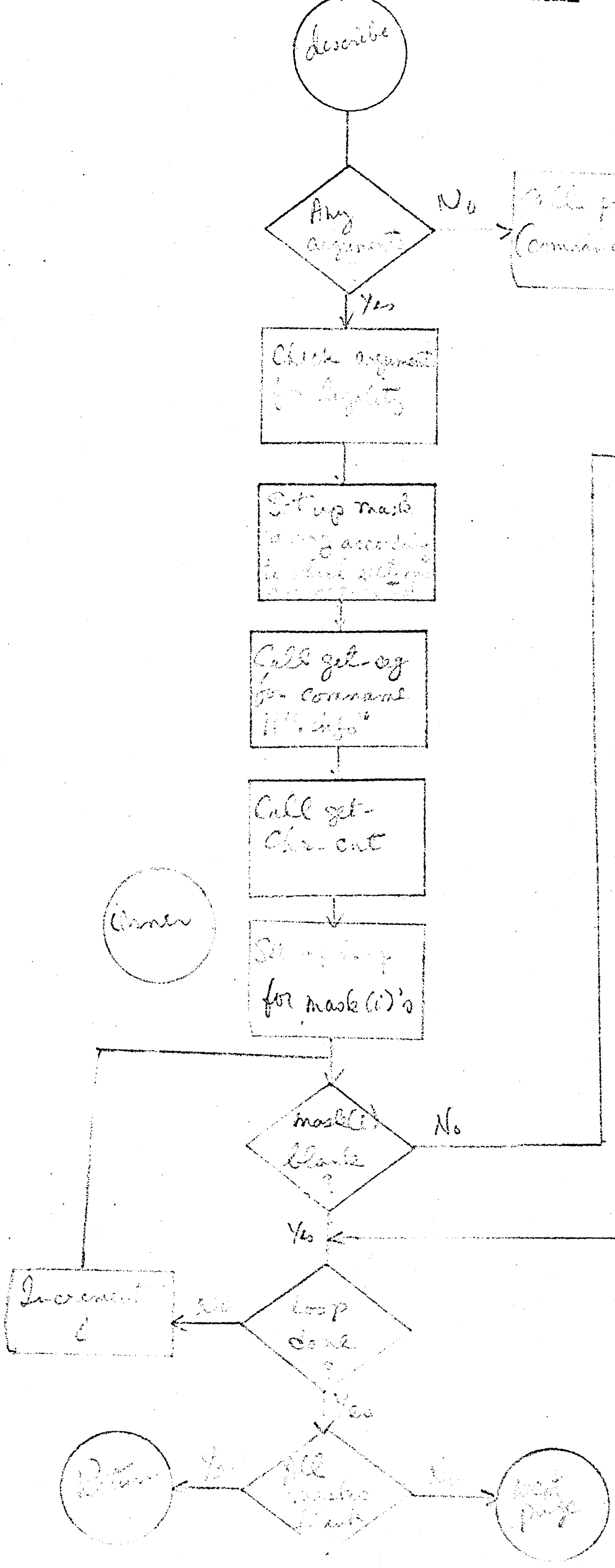
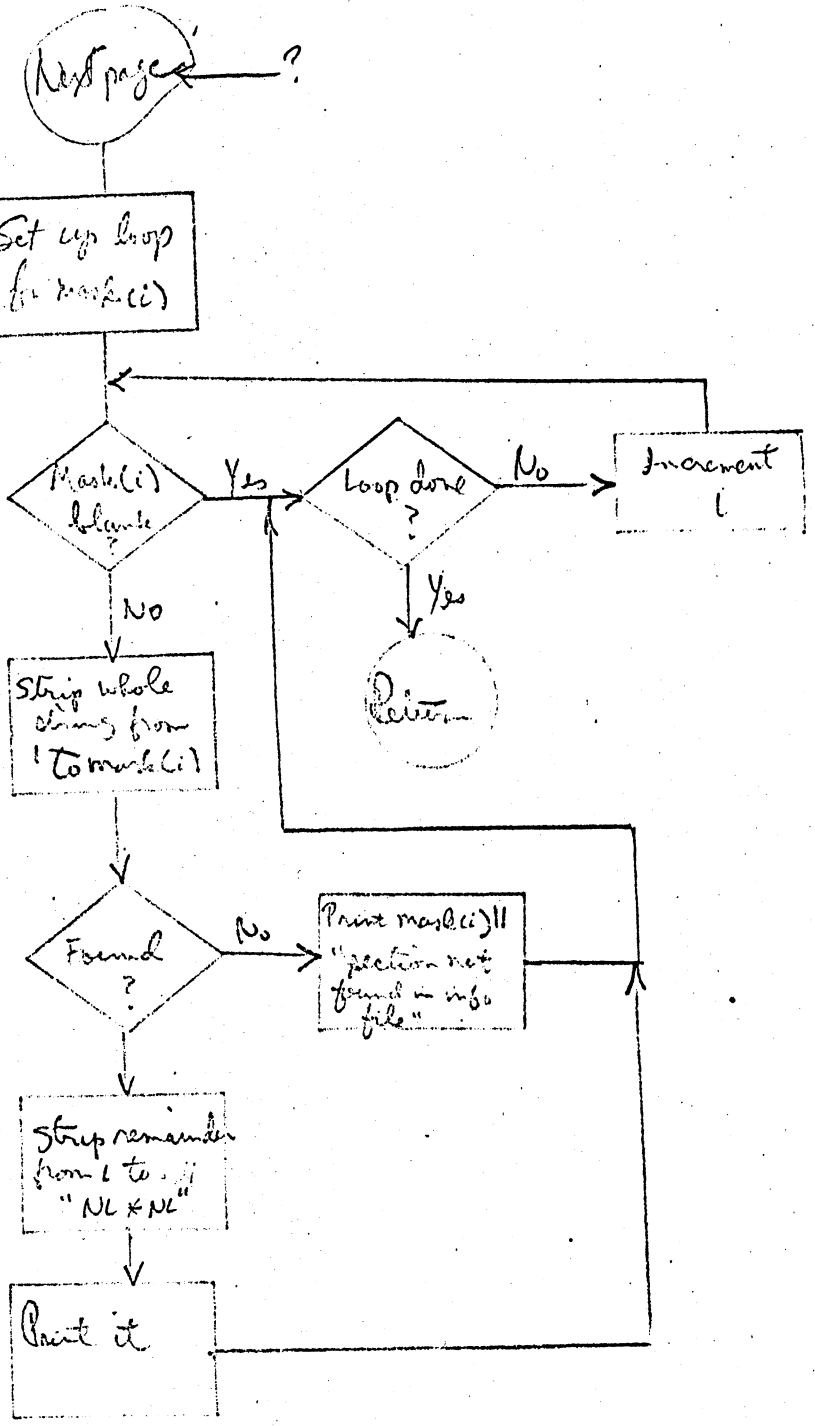

describe
Any argument
No
Call print (command.info)
Return
Yes
Check argument for legality
Set up mask array according to [illegible]
Call get-arg for command ||".info"
Call get-chr-cnt
[illegible] for mask(i)'s
Strip from 1 [illegible] mask(i)
Strip from 1 [illegible]
Print mask(i) || edstring
mask(i) = blanks
mask(i) blank ?
No
Yes
loop done ?
No
Increment i
Yes
All masks blank
Yes
No
Return
next page

Figure I


Next page
?
Set up loop for mask(i)
Mask(i) blank ?
Yes
Loop done ?
No
Increment i
Yes
Return
No
Strip whole string from ' to mask(i)
Found ?
No
Print mask(i)|| "section not found in info file"
Strip remainder from i to "NL*NL"
Print it

Don Wong
3/7/66

A Proposal for On-line Command Documentation

Purpose

One apparent weakness in current command languages is the lack of documentation available on an on-line basis. This paper discusses two possible techniques for improving documentation.

.sp 1

Discussion

When the average user invokes a command which is new or somewhat strange to him, it occasionally happens that he is uncertain as to the exact order and meaning of the arguments to be supplied. Moreover, in CTSS, it frequently is the case that the commands, after detecting the anomalous or missing arguments simply announce that "all is not well". This places the user in roughly the same position as one who gets back a batch job with the notation, "JOB DIDN'T WORK".

A simple aid to the user in cases of doubt or confusion would be the inclusion of an additional meta argument recognized by the SHELL. This argument, called the INFO REQUEST, might possibly be indicated by the notation %info.

Upon detecting this string in an argument line, the SHELL would assume that the user wishes to read a short description (say, one paragraph) of the usage of the command. Therefore, instead of invoking the command itself, the SHELL would call on the print command to print a file whose name was similar to "command.info".

.sp 1

A second difficulty when dealing with commands occurs whenever source files that make up the command are misplaced (lost). It is occasionally been the case that certain CTSS commands were either completely re-written or discarded due to the disappearance of source files.

One simple (albeit somewhat crude) remedy to enable orderly handling of commands is to simply append <u>all</u> of the relevant source files <u>including</u> library programs to the end of the text file (which is the command that the user's invoke). Due to paging, it will be the case that these appended source files will never be loaded during the command's execution. However, a suitable (and simple) program could easily extract the source files for editing or examination.