bcc	title THE COMMAND PROCESSOR	prefix/class-number.revision CMP/W-6	
checked [illegible]	authors Larry L. Barnes Larry L Barnes	approval date	revision date
checked [illegible]		classification Working Paper	
approved Mel		distribution Company Private	pages

ABSTRACT and CONTENTS

This document describes the structure of command lines and the operation of the command processor. It also describes the STATUS command.

TABLE OF CONTENTS

1. STRUCTURE OF COMMAND LINES

The criteria for a satisfactory command structure are that it be adaptable to the needs of novice to expert users, efficient in operation, and reasonably simple to implement. The first criterion is the most important, and potentially the most difficult to satisfy. The novice prefers a system which is "forgiving," "helpful," and consistent, and which contains few unexpected system comments. Further, he tends to be explicit in his requests. The expert has compactness as his primary criterion; the less typing, the better. As a final point, the command structure must adapt as a user becomes more sophisticated; he should not have to unlearn habits in order to progress.

One proposal which appears to satisfy these requirements is the following: A command line consists of a command name followed by an optional list of parameters, separated by terminators and ending with carriage return. A terminator is normally a space or comma. If the parameters are not supplied in the command line, it is left to the command to collect any necessary parameters. One example of these two processing modes is:

```
@COPY F1, F2 ¢
```

or

```
@COPY ¢
FROM FILE: F1 ¢
TO FILE: F2 ¢
```

(¢ stands for carriage return)

The messages typed in the second case are dependent upon the actual command that is invoked, but the message should indicate the type of parameter expected. Implementing this second form in a command is one way to make the system more helpful.

To summarize, the syntax of command lines is given below.

```
    line = file:name $ (terminator string) "¢"
    terminator = ($" " ","/ " ") $ " "
    string = string:char $ string:char
    string:char = -("," / " ") character
```

We call attention to the fact that parameter strings normally have syntactic restrictions not given here. See the File-Naming System specifications for the syntax of file:names.

Implicit in the line-oriented processing is the criterion of forgiveness. We assume that the command line will be collected with the system line-edit facility, however it is implemented. Thus, the user can correct typing errors at any point before the carriage return is typed. If the command name or parameters contained errors, the system will allow the user to correct the erroneous line if he wishes by editing the previous command line. This facility has proven valuable in many interactive languages and should be made universal in System I. In order to adapt to the needs of the expert, the user should be able to abbreviate names as he wishes. The abbreviation scan is covered in "The File-Naming System."

2. INTERNAL OPERATION OF THE GENERAL COMMAND PROCESSOR

In the previous section we discussed the collection of a command string. We now assume that the string is collected and has been passed to the dispatcher which will create a sub-process to perform the desired action.

The dispatcher collects characters up to the occurrence of a terminator or carriage return, whichever occurs first. This command name is then used to find the name of a file with the same main-name and type SAVE. If the name has a user-spec, then only the designated directory is searched (see File-Naming). Otherwise, the user's file directory is searched, and if that fails, a system directory is searched. If the search procedure fails, an error is returned.

If the search procedure succeeds, the file is ATTACHED to the sub-process structure. Then the sub-process is called at the initial entry point with no arguments. Command files of type 'TSAV' are destroyed when they return. Type 'PSAV' commands are not destroyed until the next 'permanent' command is typed (see the next section for more information).

Multiple entry points are implemented by making one command name the file name and the other command names links to the principal name. The command program must then look up the actual command name specified in the input line using a private table to determine the proper entry point.

3. EXTERNAL OPERATION OF THE COMMAND PROCESSOR

The command processor is capable of operating recursively through the use of escape. For the present we will discuss operation at a single level. There are two types of commands: permanent and temporary. A permanent command resides in a sub-process until a new permanent command is executed. In addition to the current permanent command, the user may execute temporary commands, that is, ones which exist only during their execution.

There is a system-wide convention that all commands which take sub-commands shall use "FINISHEDȻ" as the sub-command which returns to the calling sub-process (normally the command processor).

Thus the following scenario is typical of the operation of the command processor where SPL is an example of a permanent command and STATUS, of a temporary one.

```
@STATUS SUB-PROCESS *  ◄-
1 BCC-UTILITY Ø: 1*
@SPL                   ◄-
>sub-commands to SPL   ◄-
  .
   .
    .
>FINISHED              ◄-
@STATUS  S   *         ◄-
1 BCC-UTILITY Ø: 1*,2*
2 SPL 1: 2*
@CONTINUE              ◄-
SPL
>more sub-commands     ◄-
  .
   .
    .
```

We first listed the sub-processes (input lines are marked with ◄–). Then we executed SPL and returned, listed the sub-processes and continued executing SPL. This sequence is typical. We turn now to a description of the commands for manipulating the sub-process structure.

@permanent name...¢

This type of command RELEASEs the current permanent command, creates a new sub-process for the new name and attaches then calls it. Upon return the sub-process, its memory and files are retained until the next permanent-name is input.

@temporary name...¢

This type of command attaches the named file for the duration of its execution after which it is RELEASEd. Execution of temporary commands does not affect the current permanent command.

The following commands are calls on the utility and are treated specially.

@CONTINUE¢

causes the current permanent command to resume execution.

@SAVE-CURRENT¢

causes the current permanent command to be placed in save status so that the processing of subsequent permanent commands does not cause this command to be RELEASEd. This command responds "command-name IS sub-process number," <u>eg</u> "SPL IS 2."

@RELEASE sub-processØ

This command destroys the sub-process structure which is designated either by sub-process number or by sub-process (command) name if the name is unambiguous.

@RESETØ

destroys all sub-processes including saved ones.

@CONTINUE sub-processØ

resumes execution in the designated sub-process without disturbing the current permanent command.

These commands describe the use made by the command processor of the sub-process structure. To summarize a user may have a number of SAVEd permanent commands plus a current permanent command. He may also execute temporary commands, and saved permanent commands without disturbing the permanent ones.

The command processor recurses when escape is typed. The recursion level is indicated by the number of spaces printed before the herald "@." Each level of command processing is nominally independent. There are two utility call commands to implement returns.

@FINISHEDØ

performs a RESET and returns to the sub-process interrupted by escape. This command is not legal at the top-level.

@QUITØ

performs a RESET (at the current level) and jump returns to the next previous incarnation of the command processor.

This will usually leave the interrupted sub-process in a bad state such that CONTINUE may not work. This command is also illegal at the top-level.

4. THE STATUS COMMAND

Given that the sub-process structure can be rather complex, the commands for printing the state of a process need careful planning. The STATUS command operates in one of two modes: if no parameters are given, then it operates with sub-commands; if parameters are given, they must constitute a sub-command. In the latter case, the STATUS command performs a single sub-command and returns. We will now discuss in detail the first mode of operation.

>FINISHED

returns to the command processor.

>MEMORY ALL, and
>MEMORY *

list all private pages, followed by the file pages grouped by file.

>MEMORY PRIVATE [sub:process]

lists all private pages, or those directly controlled by the designated sub:process if the optional parameter is given. Sub:process is either a sub-process number or a command name (which might be ambiguous).

>MEMORY SUB-PROCESS sub:process

lists all pages directly controlled by the sub:process as in "MEMORY ALL."

>MEMORY FILE file [sub:process]

lists those pages which belong to the file designated by name or OFT number. As in "MEMORY PRIVATE" the listing can be restricted to those pages controlled by a particular sub-process.

All these commands list PMT entries grouped by file and then by PMT index. The format of the listing is:

```
    PRIVATE or file:name or unique:name
```

followed by the byte listing:

```
    Rnnn:pppp* sp*,...
```

'R' is printed if the page is read-only in PMT; otherwise blank is printed. 'nnn' is the PMT index. 'pppp' is the page number of the page in a file; it is not printed for private pages. The '*' following 'pppp' is printed if the page is in DWS. Following this is a list of the sub-processes which can access this entry. The sub-process number can be followed by '*' if it also controls the entry.

```
>OPEN-FILE *
>OPEN-FILE  OFT:index
>OPEN-FILE  file:name
```

list all or one OFT entry in the format:

```
        nn: file:name sp*,...
```

where 'nn' is the OFT index, and 'sp*' has the same meaning given above.

```
>SUB-PROCESS  sub:process [option]
```

lists information about SPT. The sub:process is either '*' for all or a sub-process number or name. The option is one of 'MAP,' 'STATE,' or 'COMPLETE' which combines the previous two. In any case the line:

```
        ss:rr  file:name  ff: as*,...
```

is printed, where 'ss' is the sub-process number and 'rr' is the recursion level at which it was created. The file:name is the name of the ATTACHed command. 'ff' is the father of the sub-process, and the 'as*' list is a list of sub-processes which may be called by this one, marked with '*' if it is controlled. If the 'STATE' was requested, the line:

```
ep:eg tm pak1 pak2 tak sb,...
```

is printed. 'ep' is the entry point; 'eg,' the G-register; 'tm,' the trap mask. 'pak1,' 'pak2,' and 'tak' are the access keys. The status bits 'sb' follow alphabetic abbreviations. If the 'MAP' was requested, it is listed. If the sub-process is a utility at most four lines are printed; otherwise at most eight lines are printed. Each line starts with the map page number (00, 10, ...) followed by four bytes, a '/' and four more bytes. Each byte is printed either as '␣␣-␣' if empty, '␣nnn' if read-write, or 'Rnnn' if read-only. 'nnn' is the PMT index. If any line is completely unused (empty) it is not printed.

There will be a sub-command to print SPCS.

No commands are planned to modify the process state explicitly since the utility system would have difficulties determining their legality.