DRAFT
date 7/9/68

Identification

An alternative command language

W.H. Southworth, M.A. Padlipsky

Purpose

In order to offer the Multics user a compact command language with a built-in "macro" facility, a procedure named egg has been developed. As the name suggests, the Egg is closely related to the standard Multics command language interpreter, the Shell (BX.1.00). The syntax recognized by the Egg is quite similar to that of the Shell, although it has been chosen to allow for a somewhat more consistent format and implementation. For users familiar with CTSS, it should be pointed out that the Egg is not unrelated to the familiar "dot" (.) subsystem: a user-oriented interface which provides convenience in using the time-sharing system, particularly in regard to the typing of commands.

Introduction

Being a command language interpreter, the Egg has as its primary role the invoking of procedures as dictated by character string input. As is usual in Multics, communication between procedures is performed by means of closed subroutine calls (cf. BD.7.00). Hence, the Shell is issued a command, "egg", causing the procedure of that name to be called; then the Egg is prepared to accept commands as specified herein, of the general form (but not format)

v= f(i1, i2, i3,...)

That is, a single-valued function of an arbitrary number of input arguments, all the arguments and the value being character strings. An advantage of this approach is that the input arguments may be treated as fixed-length

character strings, thus facilitating implementation.

The basic format of an Egg command is

commandname arg1 arg2 arg3 ...

That is, typing the name of a procedure and its arguments (or causing the Egg to be called with arguments from another procedure, or through the Shell, or as an absentee user) causes that procedure to be called (i.e., "the function is applied") with the specified argument list and the "value" or return argument as an implicit last argument. The significance of the value returned by a command should become clear in the examples below.

Defined syntactic elements in the command language allow the performance of additional services, beyond the simple invocation of a command: "iteration" may be caused, with a given subset of command elements being applied repeatedly to other elements; an "active function" may be specified, its value being obtained directly for the rest of the command to operate on; concatenation of arguments is performed as desired; and so on. The syntactic elements are defined below, followed by a discussion of conventions and the presentation of examples. Finally, we turn to the macro facility mentioned above.

Syntactic Elements

The Egg recognizes the following syntactic elements; essentially, they are character strings and various special-purpose delimiters (pointed brackets are used in the Backus Normal Form sense):

characterstring — A character string comprises any ASCII characters except the defined delimitors. The first string in a command is the command name, subsequent strings

are either arguments or, if appropriately delimited, constituent commands.

space — The space is the basic argument delimiter; arguments must be separated by spaces. The defined delimiters need not be separated from arguments by spaces, however; i.e., (arg) is equivalent to (arg).

{...} — Braces delimit an active function; the value of the enclosed function should be obtained and inserted into the current command line. Active functions are re-evaluated if the obtained value is other than an ordinary character string (i.e., if it in turn contains special-purpose delimiters).

[...] — Square brackets delimit a neutral function; the value of the enclosed function should be obtained and inserted into the current command line, but should not be re-evaluated, even if it contains special-purpose delimiters.

'...' — Single quotation marks delimit a literal string; the value of the neclosed string should be inserted directly into the current command line, regardless of the presence or absence of special-purpose delimiters.

(...) — Parentheses delimit a group of iteration elements, the iteration elements themselves being separated by spaces; the command line is evaluated for each element of the enclosed group of elements.

; — The semicolon denotes the end of a command; it causes termination of all function delimiters to the left

Should not begin first command if terminated by semicolon; is a continuation of typing character.

except literal string, and implicitly begins a new active function. (The new line character has the same effect as a semicolon.)

Afte the Egg has evaluated a command line until no delimiters except space remain, the command is in the basic format shown above and is itself evaluated (i.e., the named procedure is called, unless the command is a macro, as discussed later).

Conventions

The Egg itself deals only with character strings. Thus, all input and output arguments of called procedures must be converted to/from character strings for interfacing with the Egg. (Argument conversion may be added subsequently;. see below.)

Arbitrary nesting of syntactic elements is permitted.

A command implicitly begins with a left brace, {, and is terminated by a semicolon, which is equivalent to a right brace followed by a left brace, }{, in this context.

Elements of a command which are not separated by spaces are concatenated. E.g.,

```
print {getname}.text  ;
```

would cause getname to be called (active function), and the value returned by it to be concatenated with ".text", the result being passed to print as its argument. Assuming that getname returned the string "srpkg" and that print, plausibly enough, causes the printing of a file at the console, then "srpkg.text" would be printed. (Had there been a space between the } and the., however, print would have been called with two arguments, "srpkg" and ".text".) The concatenation may contain more than two components. E.g.,

```
print {pdir}>x>{getname}  ;
```

with what warning?

The Shell escape character, %, may be used.

Examples

1) Suppose procedure b returns "x" as its value; then

print {b} ;

would cause file "x" to be printed, as would

print [b] ;

2) Suppose, however, that procedure b returns "{x}" as its value and that procedure x returns "y" as its value; then

print {b} ;

would cause file "y" to be printed, whereas

print [b] ;

would cause file"{x}" to be printed, if such a file exists, as would

print '{x}' ;

Examples 1 and 2 point up the difference between active and neutral functions.

3) As an example of iteration, consider the command

print srpkg(.text .link .symbol) ;

which would cause files "srpkg.text", "srpkg.link", and "srpkg.symbol" to be printed, by virtue of the concatenation of the iterated elements with the string "srpkg".

4) A further example of iteration is

(print delete) (srpkg func1) ;

which would cause files "srpkg" and "func1" to be printed and then deleted.

Macro Facility

The Egg offers both a standard macro facility and an interface for an optional user-furnished special interpretation of command lines, which may be a special macro facility or some other procedure such as an argument abbreviation interpreter. If the user wishes to have his own procedure called before (or instead of) the invocation of a command, he issues the following command:

```
egg$set_interpreter procedurename;
```

This command causes the Egg to call procedurename when the current command has been reduced to basic format. The calling sequence is

```
call procedurename (commandname, valuestring, errcode,
argcount, arg1, arg2, arg3, ...);
```

with declarations

```
dcl (commandname, arg1, arg2, arg3, ...)
                char (*),
                valuestring char (*) varying,
                (errcode, argcount) fixed bin (17);
```

In essence, then, the user procedure is called with the current command line being made available to it; for convenience, a count of the number of arguments is also furnished. The errcode argument is to be set by the user procedure as follows: if errcode is set to 0, the Egg is to consider the command to have been evaluated and proceed with its post-evaluation processing; if errcode is non-zero, the Egg continues its processing by evaluating the command. Note that only errcode and valuestring may be written into by the user procedure; errcode must be set, but valuestring is null to begin with and if no value is intended to be returned it may be left alone. A non-standard macro processor would set errcode non-zero if commandname is not found to be a macro, thus allowing the Egg to call the command as a Multics procedure; an argument abbreviation interpreter, however, would have to set

errcode to zero, because it is not permitted to write into the (fixed-length) argument strings and must invoke the command itself.

The standard macro facility is accomplished by the insertion of an additional step in the logic of the Egg before the command in question is actually called: A "table of contents" is checked, to determine whether or not the command name is in reality a macro name. If it is not, the name is that of a standard Multics procedure, which is called in the usual fashion. If, however, the current command name is a macro name, the table of contents indicates a string which is to be evaluated in turn as a command, instead of directly calling a procedure of the given name. The evaluation of the macro string also permits the possibility of arguments to the macro, by scanning for special markers in the string and substituting arguments from the current command line as appropriate. The following commands exist, then, to establish macro definitions, to allow for macro arguments to be specified, and to cause the removal of macro definitions.

To establish a macro definition, the command employed is def, which takes as arguments the macro name and a literal string defining the macro. A simple example would be

```
def a '{list **}' ;
```

which would result in the definition of the "command" a as the performance of the list command for all files. A rather more elaborate example is

```
def . '{{read_string}} {.}' ;
```

which, assuming that read_string is a procedure which reads a character string from the user_input stream, would have the following effect: read_string is called (active function); the string read is processed as a command (active function again); when that string has been processed, the (macro) command "."

is called, and ... the process is repeated. The upshot of it all is that wait and ready messages are suppressed, just as they would be in the Egg's ancestor the CTSS "dot" command, because command level is never again reached until explicit action to do so is taken. Note that macro definitions for the same name may be "pushed down". To establish arguments within a defined macro, the command employed is arg, which takes as arguments the macro's name and whatever strings are to be specified as arguments. Suppose a macro named "p" had been established by

```
def p '{list x y  x}' ;
```

then the effect of the following sequence

```
arg p x y ;
p ab ac ;
```

would be the execution of the command

```
list ab ac  ab ;
```

To display the current set of defined macros, the command employed is dis. Finally, to remove a macro definition, the command employed is rem, which takes the macro's name as argument. E.g.,

```
rem a ;
```

would remove the macro definition associated with a.

Implementation

The implementation of the Egg is based upon the TRAC algorithm, as described in the Communications of the ACM, (Mooers, C., "TRAC, A Procedure-Describing Language for the Readtive Typewriter, Vol. 9, No.3, March, 1966).