remaining elements are interpreted as arguments for this procedure. Appropriate data type conversion is applied to these values before the procedure is called.

A number of commands separated by semi-colons and terminated by NL (new line) form a command sequence. Hence a command is terminated by NL or semi-colon, usually NL.

Element

There are four types of element:

1. string
2. literal string
3. list
4. bracketed command

String

The most common type of element is a string. A string is a sequence of one or more ASCII characters not including the punctuation marks:

[ ] ( ) { } " ` ' ; : NL blank

For example, a command consisting only of strings is:

```
delete  file_name1  file_name2
```

The value of a string is the string itself.

Literal String

A literal string is a sequence of characters (not including unmatched accents, ` ') enclosed by a grave accent on the left and an acute accent on the right. Literal strings may contain literal strings. The value of a literal string is the string with enclosing punctuation removed.
For example:

```
delete  `file  name'
```

would cause the Shell to call delete with one argument, file name. The Shell removes the enclosing punctuation before passing the argument. Literal strings may be nested, in which case the Shell removes the outermost enclosing punctuation

```
delete `file `name one''
```

would cause the Shell to call delete with one argument, file `name one'.

List

A list is a sequence of zero or more elements separated by one or more blanks enclosed by (on the left and) on the right. The value of a list is a list (i.e., structure) of the values of the enclosed elements.

For example, suppose the parameter type information on the alpha subroutine says that alpha has two arguments, an array of strings and an integer.

```
alpha (file_name1 `file name2' ) 1
```

would cause the Shell to call alpha with two arguments: 1) the array containing two strings, file_name1 and file name2, and 2) the integer 1.

It is interesting to note that

```
alpha file_name
```

and

```
alpha (file_name)
```

are not equivalent. The first is an example of a command with one argument - a string. The second is an example of a command with one argument - a one-element array of strings.

Bracketed Command

A bracketed command is a command enclosed by { on the left and } on the right. The value of a bracketed command is the value of the command which is enclosed by {}. The value of a command is the value returned by the procedure which is called as a result of interpreting the command. For example,

```
delete {getname oldfile}
```

causes the Shell to call getname with the argument oldfile.

Suppose the value returned by getname is the string, file_name1, the Shell then calls delete with one argument, the string, file_name1.

```
{getname   delete_function} file_name1
```

causes the Shell to call getname with the argument delete_function. If getname returns the string delete, the Shell then calls the delete subroutine with one argument, the string file_name1.

Parameter type information is always applied to a subroutine called by the Shell. For example, suppose compute is a subroutine whose parameter type information indicates that it has two arguments, a floating point scalar and an integer scalar; and the subroutine getval has parameter type information which indicates that it accepts one argument, a string, and has a value which is a string. Then,

```
compute  {getval  temp}  3
```

causes the Shell to call getval with the string,temp; getval returns the string 92. The Shell converts the string 92 to the floating point scalar 92 and the string 3 to the integer scalar 3 and calls compute with the floating point scalar 92 and the integer scalar 3.

Conversion when required is performed on the value of commands, but further evaluation is not done. Consider the example:

```
{getname}  arg
```

If getname returns as its value, the string {subr} , the Shell calls the subroutine whose name is {subr} with one argument, arg.

Additional features

Labels

A command may be labeled. A label is an identifier terminated by a colon (:). The initial Shell ignores labels. When macro facilities are added to the command language, an interpretation will be defined for labels.

Comments

A comment may appear before or after any element in a command. A comment begins with a double quote mark (") and

terminates with a ", NL, or semi-colon. Comments are stripped from the command and discarded by the Shell.

Subroutine Entry Point Names

The value of the first element in a command must be a subroutine entry point name or an ITS pointer to a subroutine entry point. This requirement limits the first element to the following types:

1. string
2. literal string
3. bracketed command

If the value of the first element is a string, the Shell interprets the string as a subroutine entry point name. If the string has the form,

```
seg|symb
```

then seg is assumed to be a segment name and symb is assumed to be the name of an entry point in seg. If the string does not contain a |, but has the form,

```
name
```

then the Shell interprets this as an abbreviation for,

```
name | name
```

If the value of the first element is a pointer, it is assumed to be pointing to a subroutine entry point.

Shell Escape Character

The character % is the Shell escape character. Use of this character suppresses syntactic interpretation, by the Shell, of the immediately following character. The escape character itself is discarded during evaluation except in the inner literal strings of nested literal strings.

For example,

```
delete %{file_name%}
```

causes the Shell to call delete with the argument, {file_name}. Note that,

```
delete %%file
```

causes the Shell to call delete with the argument, %file.

An example of the use of the escape character in nested literal strings is:

```
alpha `a%; `b%; c''
```

causes the Shell to call alpha with a single argument, the string,

```
a; `b%; c'
```

Null-Valued Elements

Most elements may have null values;

{}	is a bracketed null command; the value of the null command is null;
()	is the null list, i.e., a list with zero elements;
`'	is the null string, i.e., a string of zero length.

Evaluation of a command element may produce a null value. If the subroutine com returns a null value, then

```
delete  {com}  a
```

causes the shell to call delete with two arguments, the first null and the second, a.

The Interjected Command

There are certain data bases which the Shell and the system commands use that must be managed by the Shell. When the Shell begins evaluation of a new command (right after NL or ;) or a bracketed command, a copy of the current data bases is put on a stack, i.e., at the end of evaluation of a new command or a bracketed command the data bases are refreshed by popping up the copy on the top of the stack.

Evaluation of an interjected command does not cause pushing down and subsequent popping up of these data bases. Furthermore, an interjected command, although evaluated when encountered by the Shell, has no bearing on other commands being evaluated when it is encountered.

A typical use of the interjected command is the setting of options in Multics (see BX.12.00). There is a group of permanent options associated with each user which the user is free to set according to his own choosing by use of the option command. There also is a group of commands which will reset the current options. The options group is one of those

data bases which is pushed down at the beginning of a new command or a bracketed command and popped up at the end.

If the brief option is off in the user's permanent options and the command brief turns on the brief option in the current copy of the user's options, then, consider the following examples:

```
alpha  a [brief] {beta   b}
alpha  a {beta   [brief] b}
alpha  a {beta   b} [brief]
```

In the first example, both the commands beta and alpha operate with the brief option on. In the second example, only beta operates with the brief option on. In the third example, only alpha operates with the brief option on. In all three examples, the arguments to alpha are a and the value returned by beta after it is called with its one argument, b.

Appendix

Formal Description of the Command Language

The notation used here is an augmented form of BNF. The augmentation permits nested <> bracketing. It also permits a subscript to be attached which indicates bounds on the number of repetitions, i.e.,

$\langle class \rangle_{k}$ means k or more repetitions of <class>

$\langle class \rangle_{k-}$ means k or fewer (including zero) repetitions of <class>

For example,

$$\langle s \rangle ::= \langle\langle a \rangle_{1} \langle b \rangle\rangle_{2-}$$

is equivalent to

$$\langle ab \rangle ::= \langle a \rangle \langle b \rangle \mid \langle a \rangle \langle ab \rangle$$

$$\langle s \rangle ::= \mid \langle ab \rangle \mid \langle ab \rangle \langle ab \rangle$$

in standard BNF.

The syntax is described in terms of the canonical form (see BC.2.02), i.e., a single blank is <SP> and a sequence of more than one blank is <RHT>.

<command sequence> := <<left space> <labeled command>
<terminal fill> | <space>> ; >$_{0}$

<<left space> <labeled command>
<terminal fill> | <space>> <NL>

<blank> := <SP> | <RHT>

<labeled command> := <<label> <separator>>$_{0}$ <command>

<label> := <identifier>

<right space> := <<blank> <fill>>$_{1-}$ <blank>$_{1-}$

<left space> := <blank> <<fill> <blank>>$_{1-}$$_{1-}$

<terminal fill> := <right space> <<blank> <terminal comment>>$_{1-}$

<space> := <blank> $_{1-}$<<fill> <blank>$_{1-}$ >$_{1-}$

<separator> := <blank> <<fill> <blank>>$_{1-}$

<fill> := <<interjected command> | <comment>>
<<blank> <<interjected command | <comment>>>$_{0}$

<command> := <command name> <<separator> <element list>>$_{1-}$

<comment> := " <character except unquoted " ; or <NL>>$_{0}$ "

⟨terminal comment⟩ ::= " ⟨character except unquoted " ; ⟨NL⟩⟩$_0$

⟨command name⟩ ::= ⟨string⟩ | ⟨literal string⟩ | ⟨bracketed command⟩

⟨element list⟩ ::= ⟨element⟩ ⟨⟨separator⟩ ⟨element⟩⟩$_0$

⟨element⟩ ::= ⟨string⟩ | ⟨literal string⟩ | ⟨list⟩ | ⟨bracketed command⟩

⟨string⟩ ::= ⟨character except unquoted punctuation mark⟩$_1$

⟨punctuation mark⟩ ::= [|] | (|) | { | } | " | ` | ' | ; | : | ⟨NL⟩ | ⟨SP⟩ | ⟨RHT⟩

⟨quoted character⟩ ::= %⟨character⟩

⟨string quote⟩ ::= ` | '

⟨literal string⟩ ::= `⟨⟨character except unquoted string quote⟩$_0$ ⟨literal string⟩⟩$_0$

⟨character except unquoted string quote⟩$_0$ '

⟨bracketed command⟩ ::= {⟨⟨left space⟩ ⟨command⟩ ⟨right space⟩ | ⟨space⟩⟩}

⟨interjected command⟩ ::= [⟨⟨left space⟩ ⟨command⟩ ⟨right space⟩ | ⟨space⟩⟩]

⟨list⟩ ::= (⟨⟨left space⟩ ⟨element list⟩ ⟨right space⟩ | ⟨space⟩⟩)