In communicating with CPC, a standard notation is used to refer to such items as parameters of commands. By using a properly formed Command Language expression, a user or a program can refer to an integer datum, a capability for an object, or a location in which one of these items may be stored. The evaluation of an expression is controlled both by its internal form, and by the way in which it is used.

Since the impending discussion of expression evaluation will involve some fairly complicated notions, we digress briefly to more familiar ground where the same notions appear in a simpler form. Consider an assignment statement in a language like FORTRAN or ALGOL. The statement is performed by evaluating the left-hand expression to an address, the right-hand expression to a number, and finally storing the number at the address. Notice that in the statement K=K+1, the K on the left-hand side is evaluated to an address, while the K on the right-hand side is evaluated to a number. Clearly, the same expression may mean different things, depending on the *context* in which it is used. This suggests that evaluation be performed in two stages. First, each expression can be tentatively assigned an *immediate value*, independent of its surrounding context. (In the example, both occurrances of K have an immediate value of type *address*) Second, the constraints imposed by the context in which the expression occurs must be considered in assigning a final value. (In the example, the left-hand K is used in a context where a location is required; this is satisfied by an address, so the immediate value is retained as the final value. The right-hand K is used in a context where a numerical datum is required, so the contents of the address (immediate value) are fetched to produce a final value of type integer)

We now present a more precise definition of our hypothetical assignment-statement language. The same format of presentation will subsequently be used to describe Command Language expressions.

I. Environment: The environment or "universe of discourse" of our simple language consists of:

a) Numeric data (integers < some limit)

b) Memory locations in which data may be stored.

II. Types: The value of an expression is of some type, which may be:

a) Integer

b) Address

III. Contexts: The final value of an expression must satisfy the context in which it appears, as shown in table 1a.

CONTEXT ↓	TYPES OF FINAL VALUE SATISFYING CONTEXT*
datum	integer
location	address

Table 1a

*It will be seen later that, in general, a context may be satisfied by more than one type.

IV. Context Satisfaction Rules: Given the immediate value and context of an expression, table 1b shows the rules for producing a final value satisfying the context.

CONTEXT ↓	TYPE OF IMMEDIATE VALUE	
	integer	address
datum	identity	take contents
location	fail**	identity

Table 1b

** "failure" means the expression cannot satisfy the context; for example, the left hand side of $5 = K+1$

V. Syntactic Forms (grouped by the type of their immediate value)

a) Integer -

1) constant: a string of digits, literally representing an integer

2) sum: E_1 "+" E_2

The subexpressions E_1 and E_2 are to be evaluated in context datum and the results added.

b) Address -

1) Identifier: an alphabetic character, symbolically representing an address.

Note that a compound expression (e.g. sum) specifies the context of evaluation of its subexpressions.

***

We now use the same format to describe the Command Language. To avoid clouding the issue with excessive detail, we have removed the complete descriptions of the context-satisfaction rules and the formal grammar to appendices 1 and 2, respectively (see pg. 17)

I. Environment: The Universe of Discourse of the Command language consists of the set of signed integers ($|n| < 2^{59}$) and the current protection domain of access rights. Within this environment, an expression may refer to a datum, an object, the location of a datum or object, or an identifier.

We summarize the structure of the protection domain; further details are provided elsewhere (primarily in the description of the directory system.) The main component of the protection domain is the current "implicit scan list", which consists of a list of directories which can be interrogated sequentially to "look up" an object by name. Looking up an object returns a capability for the object, with access rights determined by the directory system access control features.

In SERVICES or the debugger, expressions may also reference named locations in CPC called "variables." Each variable may contain a single integer or a single capability.

Finally, in the debugger, the environment is expanded to include the

1) memory
2) C-list
3) central registers

of the "active subsystem" (the subprocess which was running just before the debugger was entered.)

II. <u>Types:</u> The value of an expression is of some type, which may be:

a) <u>string:</u> a string of characters
b) <u>integer</u>: a signed integer ($-2^{59} < N < 2^{59}$)
c) <u>capability:</u> a capability for an object
* d) <u>variable</u>: a named location in CPC
e) <u>indexed-object</u>: a location in a file or a C-list
** f) <u>subprocess-location</u>: a location in the memory or C-list of the active subsystem
** g) <u>register-location</u>: a location in the CPU state (registers) of the [redacted] active subsystem
h) <u>directory-location</u>: a location in a directory (directory; name; access-key)

* May be used only in SERVICES and the debugger
** May be used only in the debugger

III Contexts: The final value of an expression must satisfy the context in which it appears, as shown in table 2a

CONTEXT ↓	TYPES OF FINAL VALUE SATISFYING CONTEXT
identifier	string
datum	integer
object	capability
datum-location	indexed-object subprocess-location Variable register location
object-location	indexed-object subprocess-location variable directory-location

Table 2a

III. Context Satisfaction Rules: given the immediate value and the context of an expression, table 2b shows the rules for producing a final value satisfying the context. The entries are:

I = Identity (immediate value satisfies context)

R_j = Use Rule j to satisfy context (see appendix 1 for details of rules)

F = Failure (expression cannot satisfy context)

3

CONTEXT ↓	TYPE OF IMMEDIATE VALUE string	integer	capability	variable	indexed object	subprocess location	register location	directory location
identifier	I	F	F	F	F	F	F	F
datum	R_1	I	F	R_4	R_6	R_8	R_{10}	F
object	R_2	F	I	R_5	R_7	R_9	F	R_{11}
datum location	R_3	F	F	I	I	I	I	F
object location	F	F	F	I	I	I	F	I

Table 2b

V. Syntactic Forms

Note that the only types of value for which there are simple expressions are <u>string</u> and <u>integer</u>. The remaining types can only be referred to by properly constructed compound expressions. (Also, recall that a compound expression specifies the context of evaluation of its subexpressions. See page 5)

a) <u>String</u> – any sequence of letters, digits, periods, and quoted characters, beginning with a letter or a quoted character. Each quote allows the insertion of one special character into a string.

<u>Examples:</u> SYS1.FOO

REVOLUTION'#9

b) Integer –

1) A literal constant: i.e, a sequence of digits, optionally ending with B or D (octal and decimal, respectively). If no B or D is present, the current "input mode" is used (default = octal).

Examples:

573

6792D

2) An integer expression

$\pm N$ (unary)

$N_1 \pm N_2$ (binary)

$L_1 \backslash N_1, L_2 \backslash N_2, \ldots, L_m \backslash N_m$ (fields)

The subexpressions N_i and L_i are evaluated in context datum, and unary (or binary) plus (or minus), or field packing are applied to the results to obtain the appropriate integer value. Note that the fields (unlike those of a COMPASS VFD) are right adjusted in the word if $\Sigma L_i < 60$.

Examples:

-6

2+(3+7)

5\3,18D\3777B

c) Capability – a reference through a scan-list:

S > N

The subexpressions S (scan-list) and N (name to look-up) are evaluated:

S in context <u>object</u>
N in context <u>identifier</u>

The immediate value is the capability found by looking up the name in the scan-list.

<u>Example</u>:

SCANL >LOOKTHISUPFORME

d) Variable – A special named location in CPC:

↑V

The subexpression V (variable name) is evaluated in context <u>identifier</u> (hence must be a <u>string</u>)

<u>Example</u>:

↑BEADHERE

e) Indexed-object — a ________ location within a file or a C-list:

FC#I

The subexpressions FC (file or C-list) and I (index within file or C-list) are evaluated:

FC in context <u>object</u>
I in context <u>datum</u>

The indicated location in the file or C-list is referenced.

<u>Examples</u>:

WORKINGCLIST#3
SCRATCHFILE#27

f) Subprocess-location — a ________ location in the core or working C-list of the active subsystem:

#I

The subexpression I (index) is evaluated in context <u>datum</u>, and the indicated location is reference

<u>Example</u>: #

g) Register Location – a location in the CPU state of the active subsystem:

$REG#I

The subexpression I (index) is evaluated in context <u>datum</u> and the corresponding word of the exchange package is printed.

Examples:

$REG#1 (RA, A1, B1)

$REG#12 (X2)

h) Directory Location – a named entry in a given directory, with an optional access-key.

D:N

or

D:N;K

The subexpressions D (directory), N (entry name), and K (access key) are evaluated:

D and K in context <u>object</u>

N in context <u>identifier</u>

The named entry in the directory is referenced, with access controlled by the access key.

<u>Examples</u>:

TEMPDIR: SCRATCHFILE

FRIEND: SHAREDFILE; MYKEY

Appendix 1: Context Satisfaction Rules

If the immediate value of an expression does not satisfy the context in which it is used, the appropriate context satisfaction rule (if any; see Table 2b, page 10) is applied, in an attempt to produce a satisfactory final value. If no rule exists, or if the rule itself discovers an error, the evaluation of the expression <u>fails</u>.

* R_1: (<u>string</u> in context <u>datum</u>) If the string names a variable which contains an <u>integer</u>, then that integer becomes the final value.
Fails if:
No such variable
Variable is empty
Variable contains a capability

R_2: (<u>string</u> in context <u>object</u>) The string is looked up using the implicit scan-list. If a <u>capability</u> is found, it becomes the final value.
Fails if:
No capability found by look-up

R_3: (<u>string</u> in context <u>datum-location</u>) If the string names a <u>variable</u>, then this variable becomes the final value.
Fails if:
No such variable

* R_4: (<u>variable</u> in context <u>datum</u>) If the variable contains an <u>integer</u>, then that integer becomes the final value.
Fails if:
No such variable
Variable is empty
Variable contains a capability

* R_5: (<u>variable</u> in context <u>object</u>) If the variable contains a <u>capability</u>, then that capability becomes the final value.
Fails if:
No such variable
Variable is empty
Variable contains an integer

R_6: (<u>indexed-object</u> in context <u>datum</u>) If the indexed-object is a file, then the word in the file corresponding to the index (fileaddress) is fetched and becomes the (<u>integer</u>) final value.

Fails if:

- File does not exist.
- Indicated file-block does not exist
- Index is outside address-space of file
- Indexed object is not a file

R_7: (<u>indexed-object</u> in context <u>object</u>) If the indexed-object is a C-list, then the <u>capability</u> in the C-list corresponding to the index is fetched and becomes the final value.

Fails if:

- C-list does not exist
- Index is outside C-list
- Indexed object is not a C-list.

** R_8: (<u>subprocess-location</u> in context <u>datum</u>) The Contents of the indicated word in the memory of the active subsystem is fetched and becomes the (<u>integer</u>) final value.

Fails if: address falls outside memory of active subsystem.

** R_9: (subprocess-location in context object) the capability in the indicated slot in the C-list of the active subsystem is fetched and becomes the final value.

Fails if: Slot falls outside C-list of active subsystem.

** R_{10} (register-location in context datum) the indicated word in the CPU state (exchange package) of the active subsystem is fetched and becomes the (integer) final value.

Fails if: Index is outside exchange package (Index < 0 or Index > 17B)

R_{11} (directory-location in context object) The indicated name is looked up in the given directory. If the name is found, the access key is used to obtain the capability from the directory entry. That capability becomes the final value.

Fails if: Original capabilities were not for a directory and an access key.
Directory does not exist
Name not found in directory
Access key does not unlock directory entry

* applicable only in SERVICES or debugger

** applicable only in debugger