Dave

Standard Names

Throughout the Command Processor Complex a standard nameing convention is used to specify data, objects, and the location of data and objects. The Syntax and semantics of the standard naming conventions provide a uniform naming mechanism which can be used to form parameters for commands. Standard names can be used to define a datum, an object, the location of a datum or an object, or an identifier (string).

Standard manes are always evaluated within the context of an environment which controls the set of accessable objects and data. The environment of evaluation, which may change from time to time, determines the "meaning" of the standard name. The environment of evaluation consists of a default scan list and a set of "variables" maintained by the CPC. The scan list specifies a sequence of directories which can be interrogated, in order, to "look up" an object name (identifier). A directory "look up" returns a capability for the object corresponding to the proffered name. The access control features of the directory system provide control on the type of access permitted to the objects in the directories. "Variables" are named objects, within the CPC, each of which contain a single datum or a single capability. For example, a variable named HOWARD may contain a weak capability for a friend's directory, and may be used in a standard name (e.g. to access a file or subsystem in that directory).

In the debugger the environment of evaluation is expanded to include the 1) memory, 2) working C-list, and 3) central registers of the subsystem which was running just before the debugger was entered.

Standard names are written as a sequence of identifiers, integers, and/or punctuation mards. A standard name be either an "atomic" (simple) name or a compound name. The evaluation of a standard name, "atomic" or compound,

returns a value of some value-type. There are 9 value-types which can be produced by the evaluation of a standard name or element of a compound standard name. They are: 1)identifier, 2) datum, 3) object, 4) variable, 5) indexed object (index.obj), 6) scan list reference (scan.ref), 7) directory location (directory.loc), 8) subprocess location (subp.loc), 9) register location (reg.loc).

VALUE-TYPES:

| | |
|---|---|
| identifier | a string of characters |
| datum | a 60-bit word |
| object | a capability for some object |
| variable | one of the set of "variables" maintained by the CPC |
| index.obj | an object and a datum: the object is either a file or a C-list; the datum is an index to select a word from the file or a capability from the C-list |
| scan.ref | a scan list (object) (i.e. a C-list of directory and access key pairs) and a name(identifier) to "look up" in the scan list |
| directory.loc | a directory,(object) name, (identifier), and access key (object) specifying an entry in the directory |
| subp.loc | an index (datum) specifying either a location in the memory (core) or the working C-list of the subprocess calling the debugger...(available only in debugger) |
| reg.loc | an index (datum) selecting a word of the exchange package (registers) of the subprocess calling the debugger...(available only in debugger) |

When evaluating a standard name, or an element of a compound standard name, it must be specified what "kind" of value-type is desired. This is called the "mode" of evaluation. If a result is obtained which is not of a correct value-type, the evaluation procedure has at its disposal a number of type conversion functions which can be used to transform the value to a type matching the current "mode".

There are five modes of evaluation: 1) identifier, 2) datum, 3) object, 4) datum location (datum.loc), or 5) object location (object.loc). The "mode" of evaluation for a standard name may be any of five modes. For parts of a compound name, the "mode" is restricted to identifier, datum and object.

| MODE | VALUE-TYPES(S) |
|---|---|
| identifier | identifier: a string of characters |
| datum | datum: a 60-bit word |
| object | object: a capability for some object |
| datum.loc | the location of a datum<br>a) index.obj: a file and file address<br>b) variable: ~~the datum contents of~~ a variable<br>c) subp.loc: a memory (core) location (in debugger only)<br>d) reg.loc: a register location (in debugger only) |
| object.loc | the location of/for an object<br>a) directory.loc: a directory, name, and access key<br>b) index. obj: a C-list and C-list index<br>c) scan.ref: a scan list and name<br>d) variable: ~~the capability contents of~~ a variable<br>e) subp.loc: working C-list location (in debugger only) |

If the type-value resulting from the evaluation of a standard name or component of a compound standard name is not compatable with the prevailing "mode" of evaluation, the type conversion function corresponding to the value-type of the result and the current mode of evaluation is performed. If no such function exists, the standard name evaluation fails. The type conversion function may also fail in attempting to perform the transformation. The current symptoms of such failures are messages from the CPC of various degrees of obscurity (i.e. F-retrum and error indications).

Type conversion functions

type = identifier: mode = datum

An identifier may be converted to a datum to satisfy the mode of evaluation if the character string of the identifier matches a variable which contains a datum. In this case the new value is the value of the variable and the new value-type is datum.

Failure conditions: 1) no such active variable
2) variable is un-initialized
3) variable contains an object

type identifier: mode = object

An identifier may be converted to an object by "looking up" the name which is the identifier, in the "default" scan list. If the look up succeedes, the new value is the capability returned by the look up and the new value-type is object.

Failure conditions: 1) indicated name not available if "default" scan list.

type = identifer: mode = datum.loc

If the identifier corresponds to the name of an active variable, that variable can become the new value and is of type variable. This conversion function is available only for commands to the GPC (i.e. services and debugger commands).

Failure conditions: 1) no such active variable

type = variable: mode = datum

If the variable contains a datum, that value can be returned to match a datum mode of evaluation.

Failure conditions: 1) no such active variable
2) variable is un-initialized
3) variable contains an object

type = variable: mode = object

If the variable contains a capability, the capability can be returned to match an object mode of evaluation.

Failure conditions: 1) no such active variable
2) variable is un-initialized
3) variable contains a datum

type = index.obj: mode = datum

An index.obj is an object and a datum. If the object is a file (disk file or ECS file) [or a name tag for a file] then an index.obj can be converted to a datum. The contents of the word at the file address corresponding to the value of the datum of the index.obj is returned as the value, and the new value-type is datum.

Failure conditions: 1) object part is not a file [or name-tag for file capability]
2) file does not exist
3) block of file at indicated address does not exist
4) datum value is not legal file address for the file

type = index.obj: mode = object

If the object of an index.obj is a C-list [or a name tag for a C-list] then the index.obj can be converted to an object. The capability at the C-list slot selected by the value of the datum part of the index.obj is returned as the new value and the new value-type is object.

Failure conditions: 1) object part is not a c-list [or name tag for c-list capability]
2) c-list does not exist
3) datum is not legal c-list index

~~of type object.~~

Failure conditions: 1) object is not a C-list [or dynamic name tag for a C-list capability]
2) the C-list is not a proper scan list
3) indicated name (identifier) not available in the scan list.

type = directory.loc: mode = object

directory. loc is a directory (object), name (identifier), and access key

(object2). To convert a directory loc. to an object, the evaluation procedure simply looks up the name in the specified directory using the access key to authorize access to the directory entry. The capability returned by the directory system is the new value of type object.

Failure conditions: 1) object1 is not a directory capability
2) object2 is not an access key capability
3) directory does not exist
4) indicated name (identifier) not in directory
5) name is in directory but access key does not authorize access

type = subp.loc: mode = datum

A subp.loc is an index (datum) which can specify a memory word or working C-list entry in ~~a subprocess which has called the debugger~~. To convert a subp.loc to a datum, the contents of the memory word selected by the index of the subp.loc is returned as the value. This conversion function is ~~alvilable~~ available only when operating in the debugger section of the CPC.

Failure conditions: 1) index (datum) is not within field length of subprocess calling debugger (positive index)
2) index would be in debugger core or below (negative index)

type = subp.loc: mode = object

When operating in the debugger, a supb.loc can be converted to an object.

The object (capability) returned is the one found at the specified index in the ~~working~~ full C-list of ~~subprocess which called the debugger~~.

Failure conditions: 1) index (datum) is not within the full C-list of subprocess calling the debugger (positive index)
2) index would be in debugger C-list or below (negative index)

type = reg.loc: mode = datum

A reg.loc refers to the $20_8$ word exchange package (registers) of the subprocess which has called the debugger. If the index of a reg.loc is within range (0 to 20B) the contents of the word of the exchange package corresponding to the index is the datum value returned.

Failure conditions: 1) index is negative
2) index is greater than $15(17_8)$

"Atomic" (simple) standard names consist of either an identifier or a content.

<ident> ::= <letter> | <char> | <ident> <letter> | <ident> <digit> | <ident> . | <ident> '<char>

Identifiers are composed of letters, digits, periods and quoted characters. An identifier starts with either a letter or a quoted character. A simple quote mark the following single character. It makes that character part of the identifier, any character except carriage return can be quoted.

In particular, a quote mark can be quoted. The value-type of an identifier is identifier.

Example:

| | |
|---|---|
| NAME | How simple can you get. |
| NAME15 | Digits are ok except at the beginning. |
| NAME15.A | Periods are also. |
| DOLLAR'$ | The identifier is DOLLAR$ |
| '$.MONEY10 | Quoted characters can come at the beginning. |
| '$MONEY'''#.47'9 | The string is $MONEY'#.479 (Note: the quoted quotes). |

⟨const⟩ ::= ⟨digit⟩ | ⟨const⟩ digit | ⟨const⟩ B | ⟨const⟩ D

Constants are composed digits, with possibly a trailing 'B' or 'D' with a trailing 'B' it is read in octal, with a trailing 'D' it is in decimal. Otherwise, it is read in the current input mode (default input mode is octal). A constant is an integer with a maximum value of 60-bits.

The value-type of a constant is datum.

Example:

| | |
|---|---|
| 55 | $55_8$ in default input mode. |
| 59 | Illegal in octal mode. |
| 59D | Ok anytime. |
| 5912347D | A bigger octal number. |

⟨integer.prim⟩ ::= ⟨ident⟩

mode: ⟨ident⟩ identifier

result value type: identifier

semantics: identify

⟨integer.prim⟩ ::= ⟨const⟩

mode: ⟨const⟩ datum

result value-type: datum

semantics: value is value of the ⟨const⟩

⟨integer.prim⟩ ::= ↑ ⟨ident⟩

mode: ⟨ident⟩ identifier

result value-type: same

semantics: identity

⟨integer.prim⟩ ::= (⟨std.name⟩)

mode: ⟨std.name⟩ any result value type: same

semantics: identity

⟨integer.term⟩ ::= ⟨integer.prim⟩

mode: ⟨integer.prim⟩ any result value-type: same

semantics: identity

⟨integer.term⟩ ::= ⟨integer.term⟩ * ⟨integer.prim⟩

mode: ⟨integer.term⟩ datum

⟨integer.prim⟩ datum

result value-type: datum

semantics: not get implemented. Probably

48 multiplication

⟨integer.term⟩ ::= ⟨integer.term⟩ / ⟨integer.prim⟩

mode: ⟨integer.term⟩ datum

⟨integer.prim⟩ datum

result value-type: datum

semantics: not yet implemented. Probably

48 bit division

⟨integer.exp⟩ ::= ⟨integer.term⟩

mode: ⟨integer.term⟩ any

result value-type: same

semantics: identity

⟨integer.exp⟩ ::= ⟨integer.exp⟩ + ⟨integer.term⟩

mode: ⟨integer.exp⟩ datum

⟨integer.term⟩ datum

result value-type: datum

semantics: 60-bits addition

<integer.exp> ::= <integer.exp> - <integer.term>

mode: <integer.exp> datum

<integer.term> datum

result value-type: datum

semantics: 60-bit subtraction

<integer.exp> ::= + <integer.term>

mode: <integer.term> datum

result value-type: datum

semantics: 60-bit 1's complement (except for zero ...-0 produces all bits on)

Integer expressions (<integer.exp>) are expressions built from <integer.prim> using binary and unary + and - (*&/ someday). parentheses around any standard name makes it an <integer.prim>. However, unless the <std.name> is of value-type datum (or can be converted to datum) it cannot be used with the arithmatic operators.

Example:

| | |
|---|---|
| 5+10-15D | value is -2 if input mode is octal. |
| 5+7-(10-3) | =5 if input mode -5 octal |
| 5+(FLE1#10D) | value = 5+ contents of the 10th word of file k in "default" scan list. |
| 275B+(#52B+7) | 275B+ contents of cell 6/8 of memory of subprogram calling the debugger |
| TABLBASE+(CORE-FILE#51) | if TABLBASE is a variable containing |

a datum and COREFILE is a variable containing a file capability, we have contents of TABLBASE plus the contents of word 51 in COREFILE.

<word.past> ::= <integer.exp> \ <integer.exp>

mode: <integer.exp> 1 datum
<integer.exp> 2 datum

result value type: datum pair

semantics: The datum pair value is interpreted as a shift count (<integer.exp>1) and 60-bit datum value. This pseudo value type is only used in a <word.exp> and is to simulate a crude sort of COMPASS type VFD facility. (This construct is of doubtful utility).

<word.exp> ::= < word.part>

mode: <wordpart> datum pair

result value-type: datum

semantics: the result value is simply the second datum of that datum pair. The shift count part is ignored.

<word.exp> ::= <word.exp>, <word.part>

mode: <word.exp> datum
<word.part> datum pair

result value type: datum

sematics: the datum value of the <word.exp> is left shifted (end off) by the shift count of the datum pair. The shifted value is then OR'd with the datum value of the datum pair and form the result value. No checking is done to prescribe interesting fields in this pseudo VFD (also built form right instead of left as in COMPASS).

<word.exp> ::= <integer.exp>

mode: <integer.exp> any result

value-type: same

semantics: identity (this is the common parse route for names not using the pseudo VFD datum definition.)

<name.prim> ::= <word.exp>

mode: <word.exp> any result

value-type: same

semantics: identity (this is an extra production, adds nothing to the language).

<name.term> ::= <name.term> : <ident>; <name.prim>

mode: <name.term> object

<ident> identifier

<name.prim> object

result value-type: directory.loc

semantics: the directory.loc consists of the <name.term> as the directory, the <ident> as the entry name, and <name.prim> as the access key.

<name.term> ::= <name.term> : <ident>

mode: <name.term> object
<ident> identifier

result value-type: directory.loc

semantics: the directory.loc value consists of the <name.term> as the directory, the <ident> as the entry name, and the "null" access key as the access key.

<name.term> ::= <name.prim> > <ident>

mode: <name.prim> object
<ident> identifier

result value-type: scan.ref

semantics: the <name.prim> is taken as the scan list (should be a proper a-bit for scan list) while the <ident> is the name to access the scan list.

<name.term> ::= <name.prim>

mode: <name.prim> any

result value-type: name

semantics: identity

<std.name> ::= <std.name> # <integer.exp>

mode: <std.name> object
<integer.exp> datum

result value type: index.obj

semantics: the indexed.object may be either a c-list or a file. The index (<integer.exp>) along with the object (<std.name>) form the index.obj value result.

<std.name> ::=# <integer.exp>

mode: <integer.exp> datum

result value-type: subp.loc

semantics: the <integer.exp> indexes either the c-list of memory of a subprocess. This index is the value of the resulting subp.loc

<std.name> ::= $REG# <integer.exp>

mode: <integer.exp> datum

result value-type: reg.loc

semantics: The index (<integer.exp>) specifies a word in the exchange jump package of the subprocess which called the debugger. This index is the value of the reg.loc result

<std.name> ::= <name.term>

mode: <name.term> any result value-

semantics: identity

Examples:

NAME1

if mode of evaluation is

1) datum: value is datum contents of variable 'NAME1'

2) object: value is capability obtained from 'default' scan list under 'NAME'.

TEMPDIR:NEWFILE

refers to entry 'NEWFILE' in directory 'TEMPDIR'. 'TEMPDIR' is "looked up" in the default scan list. The "null" access key completes the directory.loc

↑ FRIEND:SHARFILE;OWN.KEY

'FRIEND' should be a variable containing a directory capability. The entry named 'SHARFILE' is referended with 'OWN.KEY' providing access authorization. 'OWN.KEY' is looked up in the 'default' scan list and should be an access key entry.

SCANL>DIRECT1:FILE73;↑ SPECKEY

'DIRECT1' in 'SCANL' should be a directory. 'SCANL' is looked up in the "default" scan list and then 'DIRECT1' is looked up in the 'SCANL' scan list. 'SPECKEY' should be an active

variable conating an access key.

PERDIR:DIR1:FILE1;MYKEY#TABASE+72D

'PERMOIR:DIR1' should be a directory gained using the "null" access key in 'PERMDIR', PERMDIR' is located in the "default" scan list. 'FILE1' should be a file in 'DIR1' available under 'MYKEY' (MYKEY' from 'default' scan list). The word at 'TABASE' ('TABASE' is a variable) plus 72 is the location referenced by this standard name (index.obj)

#BASEADR+10B

in the debugger this name references the word (or capability) at 'BASEADR' (BASEADDR' is a variable whose contents are taken to get a datum because of the '+') plus 10 octal.