TO: MSPM Distribution
FROM: M. A. Padlipsky
DATE: 02/03/69
SUBJECT: BB.2.01

The attached issue of section BB.2.01 defines a subset of EPL which is more restrictive than the PL/I subset defined in section BB.2 (06/24/66). BB.2.01 supersedes BB.2 on any points where there is conflict. Pending detailed review of BB.2, however, any information which it contains which is not superseded by BB.2.01 (or by BB.2.02) remains valid. (The cover letter to BB.2.02, 02/09/68, which anticipated a section on user interfaces to occupy the BB.2.01 position, may be ignored in that respect.)

Published: 01/31/69

Identification

EPL Subset for System Programming
R.M. Graham and R.C. Daley

Purpose

The current implementation of EPL generates very inefficient code in certain cases. This section describes a restricted subset of EPL. This restricted EPL, REPL, excludes all of those constructions which generate inefficient code. All future use of EPL by Multics system programmers should be limited to REPL except by special permission. There are two aspects to the definition of restricted EPL. The first is the precise definition of the subset. The second is a set of stylistic forms and standard practices. The purpose of these rules is to suggest ways of accomplishing standard functions which result in the most efficient code.

Definition of REPL

REPL is defined to be EPL with the following features excluded.

1. All varying strings.
2. All adjustable strings, arrays, and structures (including the use of asterisk for parameters).
3. All fixed length strings longer than 36 bits or 4 characters.
4. The substr function and pseudo-variable.
5. Blocks.
6. Strings, arrays, and structures as arguments.
7. Fixed binary variables with precision greater than 18 (except as noted in the next paragraph).

The following exceptions to the above rules are automatically approved.

1. The use of a prohibited data type as an argument or a parameter in order to interface with another

module which has not yet been recoded in REPL or one whose interface was engineered for users.

2. Fixed binary variables of precision greater than 18 when the magnitude of the quantity involved must by its definition exceed 18 bits in size. (Actually the object is to avoid, whenever possible, any intermediate results which have fixed precision greater than 35.)

Style and Standard Practice

The following standard practices and stylistic rules have been observed to produce expecially good code and should be used whenever possible. There may, in fact, be equally efficient ways of achieving the same functions; however, the following rules are known to produce as good code as EPL can be expected to produce for any method of carrying out the same function.

1. Any scalar argument which is to be used more than once in the procedure should be copied into the stack immediately upon entering the procedure. Note that REPL does not permit other than non-string scalars as arguments to procedures.

2. To communicate a string which does not exceed 4 characters or 36 bits, to another procedure, pass a pointer to the string. The callee then uses based storage to refer to the string.

3. Whenever strings are referred to using based storage they must be aligned.

4. To communicate arrays to another procedure pass a pointer to the zeroth element of the array. The callee then uses based storage to refer to the array.

5. Arrays should be limited to one dimension and should be declared with lowest subscript equal to 0. This rule is necessary to make rule 4 work. It also establishes a uniform practice.

6. If a called procedure requires the length of an array or string, the length should be passed as a separate argument.

7. To represent a string longer than 4 characters or 36 bits use an array whose elements are 4 characters or 36 bits

or some sub-multiple. For example, the procedure caller is passing a string of characters to the procedure callee,

```
caller:proc;
     dcl x(0:10) char(4),
         n fixed bin(18),
         p ptr;
     p = addr(x(0));
     n = 8;
     call callee (p,n);
end;

callee:proc(p,n);
/* This is an example of word by word accessing */
     dcl (p,q) ptr,
         (i,n,m) fixed bin(18),
         w(0:10) char (4) based(p);
     dcl y(0:10) char (4);
     q = p; m = n;
     do i = 0 to m; y(i) = q -> w(i); end;
end;
```

8. To scan a string character by character, the string should be treated as an array of single character elements using based storage, rather than using the substr function. For example, callee has been passed an argument which is a string of 41 or fewer characters.

```
callee: proc(p,n);
/* This is an example of character by character accessing*/
        dcl (p,q) ptr, (i,n,m) fixed bin(17);
        dcl 1 w based(p),
              2 c(0:40) char (1),
            1 y,
              2 c(0:40) char(1)
        q = p; m = n;
        do i=1 to m; if q -> w.c(i) = "a"
           then y.c(i) = "b";
           else y.c(i) = q -> w.c(i);
end;
```

9. Concatenation should not be used unless the maximum length of all results does not exceed 36 bits.