Charles Garman

J SALTZER

Identification

Some Thoughts on Character Handling with EPL Procedures.

Introduction

In the course of preparing some sample EPL programs and working with the implementation specifications for the edit command (BX.9.01), a need arose for a method of describing particular non-graphic ASCII characters, as well as procedures for performing certain operations implicit in the definition of the relative horizontal-and vertical-tab characters for canonical-form character strings.

This document describes two procedures which should be made available and an extendible proposal for symbolic reference of ASCII character-codes by name.

Procedures for Handling RHT and RVT

Below are specifications for two procedures, get_rel_count and put_rel_count, which extract and create, respectively, the binary count character which follows rht and rvt.

get_rel_count: This procedure obtains the (binary) relative count from the character which follows rht or rvt in a canonical form string. It is a procedure which returns as its value the binary number contained in the single ASCII character which was supplied as an argument. It may be programmed in three different ways:

1. assembly-language
2. use of the UNSPEC pseudo-variable
3. mismatching declarations across a CALL (char (1) ↔ bit (9)).

Since all options are implementation dependent, the procedure must obtain the necessary approval; once approved, any procedure should be able to use it

without further approval.

put_rel_count: This procedure is the inverse of get_rel_count; it takes a binary number as its argument and returns as its value a single character for concatenation after the rht or rvt; the same three implementation options hold as for get_rel_count.

Since the dope and specifiers for characters and bit strings are identical, this author recommends option (3) for implementation (see appendixes 1 and 2).

Non-graphic ASCII Characters

Although the data character set for EPL is full 7-bit ASCII, at this writing character-string literals may contain only the characters in the language character set. Presuming that this problem will disappear with later versions of EPL or the appearance of full-scale PL/I, a more fundamental problem presents itself, concerning the appearance of programs which need to work with non-graphic characters. For example, the form-feed character (ASCII 014), if embedded in a literal, would present certain confusing aspects to a person reading a program, creating as it does, a jarring gap in the printout of the program, or else appearing in its escape prepresentation. A more extreme case is that the backspace character is barred from a single character literal by the particular definition of canonical-form.

To remove these problems, the author proposes that all non-graphic characters be referred to in programs symbolically, and that a single data segment (and its associated linkage section) be available to serve as a system-wide source of these particular characters.

For each such character which a program needed, the following declaration would appear:

dcl ctl-char $character_name ext char (1);

where character_name is the ASCII (or Multics) name of the character; the list of available characters should include at least the following items:

name	definition
nl	new line
rht	relative { horizontal } tab
rvt	relative { vertical } tab
ff	form-feed
bs	backspace
rrs	red- } ribbon-shift
brs	black- } ribbon-shift
hlf	half line feed { forward }
hlr	half line feed { reverse }

how many? space?

The set should probably include all the characters in the first two columns (rows) of the ASCII table, plus del (177).

A similar arrangement could be made for the non-alphanumeric graphics, as well as the upper-case alphabetic graphics, at least until these all become available in some version of the EPL-PL/I translators.

The means for building this segment is not precisely clear; it may be possible to define the characters by means of bit-string mis-match declarations in some initialization program, or more likely by hand tailoring the data- and linkage segment after an initial cut by the assembler.

Appendix 1

The programs below show one method of implementing get_rel_count and put_rel_count by means of the ~~UNSPEC~~ UNSPEC pseudo-variable.

```
get_rel_count: proc(in_char) fixed;

/* This procedure is used as a function to obtain the count from
   the character following the relative horizontal- and vertical-tabs
   (rht and rvt), returning a fixed-point value.

   NOTE:  This procedure uses unspec to do its [dirty] work... */

dcl in_char char(1), i fixed;

          unspec(i) = in_char;

          return (i);

end get_rel_count;
```

```
put_rel_count: proc(fixed_in) char(1);

/* This procedure is used as a function to create the
   relative-count character for rht and rvt,
   given a fixed-point argument.

   NOTE:  This procedure uses unspec to do its [unclean] work... */

dcl fixed_in fixed, temp char(1);

          unspec(temp) = fixed_in;

          return (temp);

end put_rel_count;
```

Appendix 2

The programs below are the programs of Appendix 1 rewritten to remove the UNSPEC pseudo-variables, using instead mis-matched declarations across the CALL.

```
get_rel_count: proc(in_char) fixed;

dcl in_char bit(9),
          i fixed;

          i = in_char;

          return (i);

end get_rel_count;
```

```
put_rel_count: proc(in_num) bit(9);

dcl in_num fixed binary(17),
          temp bit(9);

          temp = in_num;

          return (temp);

end put_rel_count;
```