Published: 11/17/67

Identification

Obtain pointer to initialized "scratch" area
get_area
Charles Garman

Purpose

Get_area creates a new segment for a PL/I area of arbitrary size, and returns a pointer to it. The pointer, in conjunction with a dummy based area, may then be used in PL/I allocate statements; or it may be passed as an argument where an area is required in the calling sequence (e.g., in calls to the list_dir and status primitives of Directory Control.)

A second entry point permits the user to re-initialize an area previously obtained, instead of creating a new segment.

Usage and Implementation

```
dcl  p ptr,

     n fixed bin (17);

call get_area (n, p);

call get_area$re_use (n, p);
```

Before calling get_area, the user assigns a value to n; upon return from get_area, p contains a pointer to the base of the area.

For the call to get_area$re_use, p should be the value previously returned by get_area proper; n, as before, is the size of the area, and should have the same value assigned by the user as in the call to get_area.

When get_area is called (at either entry), it performs the following steps:

1. Calculates the minimum number of 1024-word blocks which contains n words. If n is larger than 262144 (2^{18}), the calculation is as if n were 262144, but no further notice is taken of this anomaly.

2. If get_area proper was called (as opposed to get_area$re_use), a call is made to the Segment Management Module primitive smm$set_name_status (BD.3.02). The arguments are such that it creates a new branch in the hierarchy (in the process directory) whose maximum length is that calculated in (1) above, the new segment is initiated and a pointer to it is obtained. The value of this pointer is then assigned to p.

3. The number of blocks calculated in (1) above is multiplied by 1024. This value is used as the "extent" of a based adjustable area, which is passed in a call to the EPL run-time procedure areamk_.

4. After the return from areamk_, the area has been (re-) initialized, and get_area returns to its caller.

Errors

Errors are possible only in step (2) above, if the segment could not be added to the process directory, or some other error was detected by SMM. In this case, the error message is recorded in the standard form provided by BY.11, and the condition get_area_err is signalled. If control returns after the signal, a value of null is assigned to p, and get_area returns to its caller.

Notes

The name of the segment is a 20-character name, the concatenation of a 15-character unique string and the constant string ".area".

When areamk_ is called to initialize the area, only a few words are modified to indicate the existence of the area; the rest of the segment is left untouched. Thus, if get_area$re_use is called, garbage previously stored in the segment is not removed; this should not affect any future operations with respect to allocating and freeing of variables in the area.

The segment will normally remain in the hierarchy until the process is terminated; if it must be deleted earlier, the user must call smm$get_seg_status to find the pathname and entry name for the segment, and then call either delete_entry (BY.2.01); or an equivalent (and cleaner, from the SMM's point of view) sequence, smm$set_del_sw followed by smm$terminate.

If a larger value of n was supplied in the call to get_area$re_use than in the original call to get_area, it will not be noted; instead, at some future time in the life of the process a bounds violation will occur when one of the library procedures attempts to allocate a new item in the segment at a point beyond the stated maximum length as originally computed. Caveat Emptor!

Example

The following program does not do anything in particular; however, it illustrates the use of get_area.

```
use_area:proc;

     dcl (aptr, branchp, xp) ptr,

         b_area area ((32)) based (aptr),

         ret_type fixed bin(2)

         some_string char (19) based (xp),

         wdir ext entry returns (char (511) varying);

     call get_area (16384/* a nice number */, aptr);

     allocate some_string in (aptr->b_area) set (p);

     p->some_string = "something or other";

     call hcs_$status((wdir), p->some_string,

         1, ret_type, aptr->b_area, branchp);

     end use_area;
```

The "32" in the declaration for b_area is strictly nominal, since the space available for allocations depends only on the declaration in effect at the time the areamk_ procedure is called (calculated in step (1) above).