Subject: MCR10083, translator_temp_$empty_all_segments

Author: Gary Dixon

Date: November 20, 2020

Repository: http://multics-wiki.swenson.org/index.php/MCRs

Multics Change Ticket: http://multics-trac.swenson.org/ticket/214 proposes a new translator_temp_$empty_all_segments entry point that allows an existing translator chain of areas to be emptied and reused. The program that allocated space in the area for previous tasks could quickly free that storage, and then reuse the area for new allocations.

This MCR proposes to add the new entry point.

Proposed Changes

The translator_temp_ subsystem provides an inexpensive temporary storage management facility for translators in the Tools Library. It uses the get_temp_segment_ subroutine to obtain temporary segments in the user's process directory. Each segment begins with a header that defines the amount of free space remaining in the segment. An entry is provided for allocating storage the temporary segment. If the requested space is not available in the existing area, the chain of temp segments is extended; the current and subsequent allocations come from that extension area segment.

Existing entry points include:

- translator_temp_$get_segment: create a new extendable, quick-allocation area.
- translator_temp_$get_next_segment: add another segment to the chain for uses besides extending the current chain of temporary allocation segments.
- translator_temp_$allocate: allocate a fixed amount of storage in the allocation area.
- translator_temp_$release_all_segments: empty the extended area and release all of its temporary segments (back to the temporary segment facility). This ends use of the area.
- translator_temp_$release_segment: release a particular segment created by translator_temp_$get_next_segment.

The new translator_temp_$empty_all_segments would release all of the temporary segments in the current chain except the first, and would reinitialize the storage header in the first segment to reflect an empty area status. Recorded ownership of the chain would be retained by the program that called translator_temp_$get_segment.

Although translator_temp_ was initially deemed a tool, this subroutine was finally documented in the MPM Subroutines manual. No info segment was created for the tool. This change proposes to add >doc>priv>translator_temp_.info to document the existing and new entry points.

The following mbuild installation script shows components of this change.

```
Description:
 A) Add translator_temp_$empty_all_segments as a new entry point.
 B) Add translator_temp_.info to document this subroutine's interfaces
    online.

Installation_directory:  >udd>m>gd>w>ttemp;

 Build_script:             ttemp.mb;

 Bound_obj:                bound_proj_admin_              IN: sss  UPDATE;
     bindfile:             bound_proj_admin_.bind          REPLACE;
     source:               translator_temp_.pl1            REPLACE  compiler: pl1 -ot;

 Info:                     translator_temp_.info          IN: priv.info  ADD;
```

The bind file is being changed only to retain the new $empty_all_segments entry point in bound_proj_admin_, where translator_temp_ currently resides.

Testing

This change has been tested extensively as part of developing a replacement for the help_ subroutine. help_ will call the new entry point as it moves between each info segment.

This testing uncovered a coding flaw in the original implementation of the new entry point, causing one or more temp segments to never be released. This flaw has been corrected and tested in the final implementation.

Documentation

The info segments documenting all entry points of translator_temp_ is shown below.

```
help translator_temp_ -all

>user_dir_dir>Multics>GDixon>w>ttemp>translator_temp_.info   (156 lines; 6 entry points in info)

2020-08-13  translator_temp_

Function:  This subroutine provides an inexpensive temporary storage
management facility for translators in the Tools Library.  It uses the
get_temp_segment_ subroutine to obtain temporary segments in the
user's process directory.  Each segment begins with a header that
defines the amount of free space remaining in the segment.  An entry
is provided for allocating space the temporary segment.  If the
requested space is not available in the temporary segment, the chain
is extended; the current and subsequent allocations come from that
extension segment.

Entry points in translator_temp_:
 2020-08-09  translator_temp_$get_segment          2020-08-09  translator_temp_$get_next_segment
 2020-08-09  translator_temp_$allocate             2020-08-09  translator_temp_$release_all_segments
 2020-11-20  translator_temp_$empty_all_segments   2020-08-09  translator_temp_$release_segment
```

```
Entry:  translator_temp_$get_segment   (22 lines in entry point; 5 other entry points in info)

2020-08-09  translator_temp_$get_segment

Function:  This entry point should be called by each program
activation to obtain the first temporary segment used during that
activation.  Before the activation ends, the program should release
the temporary segment for use by other programs.  (See the
translator_temp_$release_all_segments entry point.)

Syntax:
   dcl translator_temp_$get_segment entry (char(*) aligned, ptr,
          fixed bin(35));
   call translator_temp_$get_segment (program_id, Psegment, code);

Arguments:
program_id
   is the name of the program that is using the temporary segment.
   (Input)  This name is printed out by the list_temp_segments
   command.
Psegment
   points to the temporary segment that was created. (Output)
code
   is a status code. (Output)


Entry:  translator_temp_$allocate   (28 lines in entry point; 5 other entry points in info)

2020-08-09  translator_temp_$allocate

Function:  This entry point can be called to allocate a block of space
within a temporary segment.

Syntax:
   dcl translator_temp_$allocate entry (ptr, fixed bin) returns (ptr);
   Pspace = translator_temp_$allocate (Psegment, Nwords);

Arguments:
Psegment
   points to the temporary segment in which space is to be allocated.
   (Input/Output)  Psegment must be passed by reference rather than by
   value, because the allocation routine may change its value if there
   is insufficient space in the current temporary segment to perform
   the allocation.
Nwords
   number of words to be allocated. (Input)  It must not be greater
   than sys_info$max_seg_size-32.
Pspace
   points to the space that was allocated. (Output)  The returned
   space contains all "0"b bits.

Notes:
A program that must perform many allocations can include the
translator_temp_alloc.incl.pl1.  This include file contains a PL/I
quick internal procedure called allocate that can be called like
translator_temp_$allocate as shown above.
```

```
Entry:  translator_temp_$empty_all_segments   (23 lines in entry point; 5 other entry points in info)

2020-11-20  translator_temp_$empty_all_segments

Function:  This entry point can be called by a program activation to
empty storage allocated in all temporary segments (obtained earlier by
calling translator_temp_$get_segment or $get_next_segment).  All except
the first temporary segment are released back to the get_temp_segment_
pool.  Storage in the given temporary segment is freed, and its
no-free storage area is initialized to all "0"b bits.

Syntax:
   dcl translator_temp_$empty_all_segments (ptr, fixed bin(35));
   call translator_temp_$empty_all_segments (Psegment, code);

Arguments:
Psegment
   points to one of the temporary segments obtained previously.
   (Input/Output)  Psegment must be passed by reference rather than by
   value, because the empty mechanism may change its value if the
   input pointer an extension segment in the list of temporary
   segments.  Upon return, Psegment points to the first segment of the
   translator_temp_ list.
code
   is a status code. (Output)


Entry:  translator_temp_$get_next_segment   (20 lines in entry point; 5 other entry points in info)

2020-08-09  translator_temp_$get_next_segment

Function:  This entry point can be called by a program activation to
obtain an additional temporary segment for allocations.  The new
segment is chained to the initial temporary segment returned by
translator_temp_$get_segment.

Syntax:
   dcl translator_temp_$get_next_segment (ptr, ptr, fixed bin(35));
   call translator_temp_$get_next_segment (Psegment, Pnew_segment,
        code);

Arguments:
Psegment
   points to one of the temporary segments obtained previously.
   (Input)
Pnew_segment
   points to the new temporary segment. (Output)
code
   is a status code. (Output)
```

```
Entry:  translator_temp_$release_all_segments   (19 lines in entry point; 5 other entry points in info)

2020-08-09  translator_temp_$release_all_segments

Function:
This entry point releases all of the temporary segments used by a
program activation for use by other programs.  It truncates these
segments to conserve space in the process directory.  It should be
called by each program activation that uses temporary segments before
the activation is terminated.

Syntax:
   dcl translator_temp_$release_all_segments entry (ptr,
      fixed bin(35));
   call translator_temp_$release_all_segments (Psegment, code);

Arguments:
Psegment
   points to any one of the temporary segments. (Input)
code
   is a status code. (Output)


Entry:  translator_temp_$release_segment   (16 lines in entry point; 5 other entry points in info)

2020-08-09  translator_temp_$release_segment

Function:
This entry point releases one of the temporary segments used by a
program activation.  It truncates the segment to conserve space in the
process directory.

Syntax:
   dcl translator_temp_$release_segment entry (ptr, fixed bin(35));
   call translator_temp_$release_segment (Psegment, code);

Arguments:
Psegment
   points to the temporary segment to be released. (Input)
code
   is a status code. (Output)
```

Version History

Date	Revision	Author	Comment
2020-11-20	1.0	Gary Dixon	Initial version of this MCR.