Subject: MCR10113, Shared template_slt_.alm Sources

Author: Gary Dixon

Date: January 31, 2022

Multics Change Ticket #180 reports that the template_slt_.alm source is included in two different bound segments, and that slightly different versions of this source are used in each bound object. The two objects are: the hardcore bound_bootload_0 segment; the tools bound_checker_ object which contains the check_mst and compare_mst commands.

The actual ALM source lines are identical in both versions of the segment. The hardcore version was updated in 1987 to add a Honeywell Bull copyright notice, and to add additional comment lines describing the purpose of each section of source in this multi-purpose setup module. No history comment was added attributing these source comments to an author.

Both bound_bootload_0 and bound_checker_ are tasked with loading a Multics System Tape. During system bootload, the bound_bootload_0 version actually loads each file from tape into memory in order to start Multics operation. The check_mst program goes through many of the same loading steps to get a list of programs loaded from the MST tape for two purposes.

- Ensuring that all modules needed for each boatload collection appear in proper order on the tape, verifying that inter-module links will be snapped correctly during the real bootload operation.
- Printing a detailed hardcore_checker_map listing all segments loaded with their external entry points, segment numbers, names, ACLs, ring brackets and other attributes. The functions of the template_slt_.alm source are needed for both operations.

bound_bootload_0 is not retained after the bootload operation completes: thus, there is no >sl1>bound_bootload_0 segment in the hierarchy whose template_slt_ subroutine could be called from bound_checker_. So template_slt_ must be included explicitly in each bound segment.

Arguments could be given in an atypical bind command line to bind a single standalone object component with components of an object archive to form each bound object segment. For example:

```
bind bound_checker_.archive template_slt_
```

tells the binder to group the object components in bound_checker_.archive with an unarchived template_slt_ object while following instructions in a bound_checker_.bind file contained in the archive. A similar command could be given to create bound_bootload_0. But there is no way to specify this unusual archive-with-standalone-object binding using only the syntax of a .bind file.

To avoid using such atypical bind commands, the developers decided to just include an identical source file and its object in the archives of each bound segment. But the developers failed to note this duplication in source comments, relying instead on undocumented knowledge held within the hardcore development team to keep these two source components synchronized.

As Multics development efforts wound down, this team-specific hardcore development lore dissipated, and the two versions diverged slightly in content. This divergence should be corrected now and avoided in any future changes.

Proposed Changes

Since the hardcore version of template_slt_.alm contains a superset of information in the two versions, that version will be used as a starting point.

A warning comment will be added near the top of this module to warn future developers that both versions must be updated at the same time.

```
" ********************************************************************************
"
" WARNING: This source program is included in 2 different bound objects.  Any
"          changes must be duplicated in the shared source file which appears
"          in the source archives:  >ldd>hard>s>bound_bootload_0.s.archive
"                                   >ldd>tools>s>bound_checker_.s.archive
"          Please update both of these component objects whenever any change is made
"          to this shared source program.
"
" ********************************************************************************
```

Because the same object component appears in two different bound segments, mbuild requires each bound object to be built in a separate installation directory. The same template_slt_.alm source will appear in each directory; but each could have different fields in its history comment.

NOTE: It would be useful if the installer ensures all fields in the new history comment remain identical in both versions.

The build script for the bound_checker_ change is shown below.

```
Description:
 1) Merge comments in the two versions of template_slt_.alm to form a
    unified source file that is incorporated into two bound objects:
     >ldd>hard>o>bound_bootload_0.archive  and >ldd>tools>o>bound_checker_.archive
 2) This installation updates the bound_checker_ object segment.

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

 Build_script:             tslta.mb;

 Bound_obj:                bound_checker_           IN: tools  UPDATE;
     source:               template_slt_.alm         REPLACE            compiler: alm;
```

The build script for the bound_bootload_0 change is shown below:

```
Description:
 1) Merge comments in the two versions of template_slt_.alm to form a
    unified source file that is incorporated into two bound objects:
     >ldd>hard>o>bound_bootload_0.archive  and >ldd>tools>o>bound_checker_.archive
 2) This installation updates the bound_bootload_0 object segment.
```

```
NOTE: Because this source exists in two different bound segments, when processing the
      hardcore change, you must invoke mbuild in a special fashion to avoid seeing
      references to both source locations.  The sequence is:
          mbuild -no_scan -request "read; az"

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

 Build_script:             tsltb.mb;

 Bound_obj:                bound_bootload_0        IN: hard  UPDATE;
     source:               template_slt_.alm         REPLACE           compiler: alm;
```

Testing

Because this change involves only comments in template_slt_.alm, testing focused on ensuring the content of the object segment matched the original template_slt_ component in bound_checker_.archive; and ensuring the same entry points and links were included in that object. This was done by using compare_object against the original and new version of template_slt_; and by comparing the print_link_info outputs from that original and new object component. Only length of symbol sections differed between the two versions. This occurs because each version was assembled in a different directory, and that source pathname is included in the symbol section of each object component.

Documentation

There are no interface changes to document.

Version History

Date	Revision	Author	Comment
2022-01-31	1.0	Gary Dixon	Initial draft of this MCR.
2022-01-31	1.1	Gary Dixon	Changes suggested by auditor.