Subject: MCR10177, Fix bugs in archive.pl1 and its subroutines

Author: Jim Lippard

Date: September 25, 2026

Multics Ticket: https://multics-trac.swenson.org/ticket/412
https://multics-trac.swenson.org/ticket/413
https://multics-trac.swenson.org/ticket/414
https://multics-trac.swenson.org/ticket/415
https://multics-trac.swenson.org/ticket/416
https://multics-trac.swenson.org/ticket/417

Problem Description

The archive command and its supporting procedures have a number of bugs, several of which destroy data: an archive that cannot be read can be replaced by a new one holding only the components being added (bug 1); extracting with force over a link deletes the link's target elsewhere in the hierarchy (bug 5); a component name that is too long selects, and can delete, a different component (bugs 7, 28); a global extract-and-delete can remove components that were not extracted (bug 8); and the copy of an updated archive saved after a record quota overflow can be destroyed by the next update (bug 4). Others leave the user with access they did not have (bugs 2, 3), leak segments (bug 10), print misleading messages (bugs 11, 12), or accept archives that `archive_` rejects (bugs 20–23), so that the command and the subroutine disagree about what an archive is. This MCR proposes fixing all of them, along with a number of minor bugs found by reading the code, and corrects archive.info.

Bugs are numbered across all five programs. Line numbers are those of the MR12.8 source and indicate the start of the problem area. "Test" names the exec_com in the test kit that shows the bug before the fix and its absence after (see Testing); "Inspection" means the bug was found by reading the code and has no reliable visible symptom. Item 6 is not a bug but documented behavior missing from archive.info; it keeps its number so that the numbers match the test kit and the bug reports.

archive.pl1 (the archive command)

Bug 1: an archive that exists but cannot be read is treated as missing, line 460

For the replace and append keys (`r`, `rd`, `a`, `ca`, ...), any failure of `initiate_file_` on the archive, not only "entry not found", was taken to mean that the archive did not exist. The command then tried to create a new archive, got "name duplication", and called `nd_handler_`, which asks whether to delete the existing archive. Answering yes replaced the whole archive with one holding only the new components. Now only

`error_table_$noentry` lets the command create a new archive; any other error is reported and the command stops. Test: bug01.ec.

Bug 2: updating a protected archive deletes the user's own ACL entry instead of restoring it, lines 672 and 717

When the user lacks `w` on the archive, the command asks whether to update it, gives the user's process group ID (e.g. `Person.Project.a`) `rw`, and afterwards deletes that ACL entry. If an entry for that exact name already existed, it was overwritten and then deleted, not restored. The command now uses `update_handler_`, which records any existing entry, and `update_handler_$reprotect`, which restores it. The command's own query procedure (`ask_question`) is removed. Test: bug02.ec.

Bug 3: forced access is left on a protected archive after a failure or an interruption, lines 702 and 792

The ACL entry added for bug 2 was removed only on the successful path. After a record quota overflow, or if the command was interrupted and released, the user was left with `rw` on the archive. Access is now restored on the error path (`MOVE_ERROR`) and by the cleanup handler. Test: bug03.ec.

Bug 4: after a record quota overflow, the saved copy of the updated archive is misreported and then destroyed, line 705

When an update fails for lack of quota, the updated archive (built in the process directory as `archive_temp_.archive`) is renamed to the archive's name, and the user is told where to find it. If that name was already taken in the process directory (e.g. by a copy saved by an earlier failure), the message named `[pd]>temp.archive`, which does not exist, and the segment stayed the command's temporary, so the next update in the process truncated it to 0 records, destroying the only updated copy. Now, if the rename fails, the copy is renamed to a unique name, the message gives that name, and the segment is no longer used as the temporary. The record quota probe itself (contents_overlay and the record_quota_overflow handler) is unchanged; only the recovery after it is changed. Tests: bug04.ec, bug04d.ec.

Bug 5: extracting with force over a link deletes the link's target instead of the link, line 855

When the destination entry exists and is a link, `x` and `xd` ask (through `nd_handler_`) whether to unlink it, and `copy -force` unlinks it with the message "Name duplication. Unlinking the old link ... (Target ... exists.)"; in each case the target is not touched. `xf` and `xdf` instead called the command's own `delete_seg`, which chased the link and deleted its target (elsewhere in the hierarchy), then created a new segment through the link in the target's directory. No segment was created in the working directory, the target lost its other names and its ACL, and no message was printed. For a link, `xf` and `xdf` now call `nd_handler_$force`, so they unlink it with the same message as `copy -force` and extract the component in the working directory; an existing segment is still deleted as before.

This follows the MPM, which says that the extract operations handle an existing entry by the copy command's duplicate name convention; archive.info is corrected to say so. Test: bug05.ec, which also runs `copy -force` and `ac x` over a link for comparison.

6: Not a bug: deleting a linked source after rd, ud or ad deletes the link's target, which archive.info does not say

This is documented behavior (MPM Commands, archive, Notes: "For segment deletions after replacement requests, if the specified component name was a link to a segment, the segment linked to is deleted. The link is not unlinked."), so the code is not changed. But it is unexpected, and archive.info does not mention it; archive.info now includes the MPM's statement.

Bug 7: component names longer than 32 characters are truncated for table and delete, line 535

For `t`, `tl`, `tb`, `tlb`, `d` and `cd`, a component name argument was assigned to a 32-character variable without a length check, so a longer name matched the component named with its first 32 characters: `ac d` deleted a component that was not named. Such names are now rejected with `error_table_$entlong`, as they already were for the other keys by `expand_pathname_`. Test: bug07.ec.

Bug 8: a global extract-and-delete can remove a component that was not extracted, line 835

In `xd` with no component names, if creating the segment for a component failed (e.g. a directory of that name exists), whether the component was kept in the archive depended on a stray word of storage (see bug 9), so it could be deleted from the archive without being extracted. It is now always kept. Test: bug08.ec.

Bug 9: a failed extraction with no component names writes outside the argument array, line 835

In the same case as bug 8, and for `x` with no names when a segment cannot be created (e.g. no append permission), the command read and wrote element 0 of the argument array, which does not exist, overwriting a word of the stack frame before the array. These references are now made only when a component name was given. Inspection (exercised by bug08.ec).

Bug 10: a global delete-type update of more than 102 components creates, and leaks, a segment for each further component, line 990

The component name arrays were kept in stack_space, a 1024-word automatic array, and moved to a made segment when they outgrew it. In a global update with deletion, the test was repeated for every further component, so a new segment was made, and the previous one leaked, for each component after the 102nd. The arrays are now always in a temporary segment from get_temp_segment_, released by the cleanup handler. Their dimensions fill a segment: 4,926 elements for component names given as arguments, and 26,112 for a global operation, more than an archive can hold. More than 4,926 component names gets error_table_$too_many_args. This

removes stack_space (a page of stack initialized on every call), the 113-entry threshold, the element-size constants, and the code that moved the arrays. Test: bug10.ec.

Bug 11: the global update messages name the archive's directory instead of the working directory, line 778

"No matching segments in DIR" and "...no components were updated from DIR" named the directory containing the archive, but the source segments for a global update come from the working directory. They now name the working directory. Test: bug11.ec.

Bug 12: with the copy keys, messages name the original archive, which is unchanged, lines 760 and 769, and an error setting the bit count of the copy named the original archive, line 688

For `cr` and its variants, "appended to" named the original archive rather than the new copy in the working directory. The "Archive segment overflow" message had the same error. Both now name the archive that was changed. Test: bug12.ec.

Bug 13: an archive overflow while copying leaves an incomplete new archive, line 1087

If a copy key (`cr`, `ca`, `cd`, ...) ran out of room while copying components into the new archive in the working directory, the command stopped and left the partial archive with a bit count of 0. It is now deleted. Inspection (needs an archive near the maximum segment length). When a new archive is created, nothing is copied, so this overflow cannot occur. An append that overflows skips the segment that does not fit and leaves a valid archive holding the others. This was tested by appending bound_pl1_.2.s.archive and bound_pl1_.4.s.archive to a new archive, with the old and new archive commands; both report the overflow and keep the first.

Bug 14: segments are left initiated on error paths, lines 851, 975, 1021 and 1030

When a source's bit count is inconsistent with its length, and when adding a source would overflow the archive, the source segment was not terminated. When `xf` replaced an existing destination entry, the segment returned by the first `hcs_$make_seg` was not terminated. Inspection.

Bug 15: a global replace sets a flag through a stale index, line 935

If the archive itself matched a component name in a global replace, the command set `flags (i)` using the index of the last named component, which in the global case is stale and points into the deletion list. The assignment was redundant for named components and is removed. Inspection.

Bug 16: an error from fs_util_$get_max_length is ignored when a new archive is created, line 1120

The code tested the wrong variable (`code` instead of `error`) after the call. Inspection.

Bug 17: an unexpected error from archive_util_$first_element is ignored, line 473

If the archive's status could not be obtained, `first_element` returned a system error code, which the command treated as success. It is now reported. Inspection.

Bug 18: the archive is not terminated when the command stops on an early error, line 467

When the archive's mode or maximum length could not be obtained, the command returned without its normal cleanup. It now goes through cleanup. Inspection.

Bug 19: word counts are computed with 17-bit precision, lines 816, 1008 and 1078

`divide (..., 17, 0)` was used for component word counts, which can exceed 131,071 for large components. Changed to 19 bits, the precision of the variable. Tests on MR12.8 showed no failure, because the values are held in full words. Inspection (cleanup).

archive_util_.pl1 (used by the archive command to read archives)

Bug 20: headers beginning with the obsolete archive_data_$header_begin are accepted, line 71

`archive_` rejects such archives, so the command and the subroutine disagreed on what an archive is. Only `archive_data_$ident` is now accepted. Test: bug20.ec.

Bug 21: the header fence (archive_data_$fence) is not checked, line 71

A header with a damaged fence was accepted. It is now a format error, as in `archive_`. `archive_` compares the fence against `archive_data_$header_end`, an older name for the same storage (`archive_data_.alm` defines header_end and fence at the same location); the comment in `archive_.pl1` explains why archive_ uses that name. It is left unchanged. Test: bug21.ec.

Bug 22: the bit count field is not validated, lines 61 and 121

It was converted with `cv_dec_` with no check, so a field that is not a number was taken as some value (possibly 0) and the scan continued at the wrong place, possibly treating component data as the next header. The field must now be digits (left or right justified, as `archive_` allows), and the component must lie within the archive. Test: bug22.ec.

Bug 23: an archive whose bit count is not a whole number of words is accepted until its end, line 92

The archive's length was rounded up to a word, so its components were listed or processed before a format error was reported at the end. Such an archive is now rejected at the start. Test: bug23.ec.

Bug 24: a header that extends past the end of the archive is read, line 69

When 1 to 24 words remained after a component, the next header was read partly beyond the archive's bit count. The whole header must now lie within the archive. Normally this produced a format error anyway, so the result is the same. Inspection.

Bug 25: offsets are computed with 17-bit precision, lines 36 and 61

Word offsets and the archive length were held as `fixed bin (17)` and computed with `divide (..., 17, 0)`, limiting them to 131,071 words; an archive can be as long as the maximum segment length, sys_info$max_seg_size (261,120 words). An offset derived from a corrupt bit count could also wrap around the segment. Offsets are now computed in 35 bits and checked against the archive length before a pointer is formed. (The overflow checks in archive.pl1 already use the archive's own maximum length and include the component header.) Inspection (tests on MR12.8 with archives over 128K words showed no failure).

Bug 26: first_disected returns component outputs along with an error code, line 112

On a status error or an empty archive, the segment pointer, name and bit count outputs were filled from whatever the header pointer addressed. They are now cleared whenever the code is nonzero. Inspection.

Bug 27: incorrect declarations, lines 34, 40, 103 and 135

`hcs_$status_mins` was declared with `fixed bin (17)` type and code arguments (actual: `fixed bin (2)` and `fixed bin (35)`); the error code parameters were `fixed bin (17)`, though a full `error_table_` code can be returned in them; `cv_dec_` was declared as a subroutine. Corrected; the codes returned by `first_element` and `first_disected` when status cannot be obtained are now documented in the history comment. Inspection.

archive_.pl1 (the archive_ subroutine)

Bug 28: a component name longer than 32 characters is truncated, line 110

`archive_$get_component` and `archive_$get_component_info` truncated the name to 32 characters, so a longer name found the component named with its first 32 characters. They now return `error_table_$no_component`. Test: bug28.ec.

Bug 29: on a format error in list_components, the component list is freed without naming the caller's area, line 266

The list is allocated in the caller's area, but freed with a plain `free`. It is now freed `in (output_area)`, as the cleanup handler already did. Inspection (no symptom was observed

because the area used in the tests permits freeing, where each block identifies its area; in an area that does not permit freeing, a plain free would fail).

Bug 30: a header that extends past the end of the archive is read, line 336

As bug 24, in `archive_`. Inspection.

Bug 31: next_component with a pointer that does not follow a header reports a format error in the archive, line 176

A `component_ptr` into the middle of a component was backed up by a header length and rejected as `error_table_$archive_fmt_err`, blaming the archive for the caller's bad argument. It now returns `error_table_$bad_arg`. Inspection.

Bug 32: missing parameter attribute, line 50

On the `P_n_components` declaration. Cleanup.

archive_aux_.pl1 (archive command utilities)

Bug 33: a directory in the working directory is treated as a source segment, line 102

For a global replace or update (`r`, `u` and variants with no names), a directory, or a link to one, whose name matched a component was taken as the component's source; the command then reported an error for it. Directories are now ignored. Test: bug33.ec (a directory; the link-to-directory case by inspection).

Bug 34: free uses an uninitialized area pointer, line 119

The `free` entry freed the directory listing `in (area)`, where `area` is based on a pointer set only in the invocation of the `listwdir` entry. It now locates the system free area itself. Inspection.

Bug 35: listwdir returns garbage if hcs_$star_list_ fails, line 75

The entry and name pointers were not initialized, and the entry count was updated even when the call failed. They are now null and zero on failure. Inspection.

update_handler_.pl1 (now used by the archive command)

Bug 36: for an entry in the root directory, the protected-segment query shows the path as >>name, line 68

The path was built with `^a>^a`. It is now built with `pathname_`. This affects all callers of `update_handler_`. Inspection.

`archive_sort.pl1` and `reorder_archive.pl1` are also changed, only to declare the access mode arguments of `update_handler_` as bit (36) aligned, as `update_handler_` now does. Both are in `bound_archive_` and are recompiled with it.

archive_util_.pl1

An additional bug was found in the pre-audit:

Bug 37: search uses the archive length left by the previous first_element, line 140

`search` did not find the archive's length itself. It used the value left in internal static by the last call of `first_element` or `first_disected`, which might be for another archive, or 0 in a new process. When that length was too short, `search` found the first component but reported a format error for any later one. The stricter checks of bugs 21–24 would have extended this to the first component as well. Now `search` takes the length from the archive it is given, as `first_element` does, at the cost of a second `hcs_$status_mins` call when the caller has already called `first_element`. `search_return` now also checks the error code, as `disected_return` does (bug 26), since the new path can return an error with the header pointer still set (an empty archive, or a segment whose status cannot be obtained). On every path that existed before, an error code was always returned with a null header pointer, so the added check changes nothing there. Found in pre-audit. The only live caller, `get_archive_file_`, calls `first_element` first, so no installed program is affected. Test: bug37.ec, using search_test.pl1:

Search, after walking a 69-word archive	Old	New
first component	found	found
later component	format error	found
name not in the archive	format error	not found
empty archive	format error	not found

After `first_element`, both versions give the correct result.

archive.info

- `paths`: says what paths are for append, replace, update and extract (the source or destination segment), not only for table and delete.
- `tlb`: said it suppresses "the time portion of dates"; it suppresses the title and column headings, as the MPM says.

- `ca`: "a copy of the new archive segment" corrected; `cud`: "copies, update, and deletes" corrected.
- `x`, `xf`: restores the MPM's statement that an existing entry is handled by the copy command's duplicate name convention, and says that `xf` unlinks a link as `copy -force` does (bug 5).
- Notes: adds the MPM's statement that deletions after replacement requests delete a link's target (item 6); that the copy operations cannot be used on an archive in the working directory; and what happens after a record quota overflow (bug 4).

archive_.info

- `next_component_info`: documents `component_ptr` (Input/Output) in place of a nonexistent `component_name` argument; "a" → "an".
- `next_component`, `next_component_info`: lists `error_table_$bad_arg` and says when it is returned (bug 31; it was already returned for some bad pointers).
- `list_components`: lists `error_table_$unimplemented_version`, and says that the area condition is signalled if the caller's area is too small.
- `get_component`, `get_component_info`: a name longer than 32 characters gives `error_table_$no_component` (bug 28); with duplicate names, the first component is found.
- Punctuation ("one of the following;"), and the dates of the changed entries.

Changes made during audit

- archive_util_: one code parameter for all entries, replacing code4, code5, icode and scode; the entry flag named, with named constants; wordno and addwordno in place of bin (rel ()) and addrel; the bit count converted with convert after checking it is all digits.
- archive_aux_listwdir: the caller is made responsible for freeing before hcs_ star_list_ is called; get_system_free_area_ is declared as a function.
- update_handler_: uses query_info.incl.pl1 and acl_structures.incl.pl1; its access mode parameters are bit (36) aligned, and so are those of its callers.
- archive.pl1: messages about the archive being written name new_archive_dir; unneeded initial values removed from aux_wstructure.
- archive.info and archive_.info: the problems reported by verify_info corrected (ISO dates, the archive and ac names, long paragraphs, section order); archive_component_info_array_ptr refers to the include file and to archive_$get_component_info; the original tabs restored on unchanged lines.

Changes in behavior

- Bug 5: `xf` and `xdf` over a link now unlink it, with the message `copy -force` prints, and extract the component in the working directory.
- Bugs 20–23: `archive_util_` now rejects archives with an obsolete header, a damaged fence, a non-numeric bit count, or a bit count that is not a whole number of words, as `archive_` already did. This affects every caller of `archive_util_`: in the MR12.9 source, archive, hunt, get_archive_file_, extract_message_doc, hunt_dec, upd_name_task_, and save_previous_system (total.crossref lists eleven: xdw_ and macro_ declare archive_util_ entries but never call them, and firmware_util_ and include_cross_reference are in system_library_obsolete). A scan of every archive under >ldd and >sl1 on Gold Hill with the new code found none that it rejects; a few archives could not be read for lack of access (see Testing). No include file changed, so only the five changed programs need recompiling.
- Bug 33: in a global replace or update, directories that match component names are skipped silently.
- Bug 2: archive's protected-segment query is now update_handler_'s, "Do you want to update the protected segment ^R^a^B?", in place of its own "Do you want to update the protected segment ^a ?". The path is highlighted and the space before the question mark is gone. No exec_com in the MR12.9 source matches against this text.
- Bug 4: after a second record quota overflow in a process, the saved copy is in `[pd]>!<unique>.archive` (as the message says) rather than `[pd]>archive_temp_.archive`.
- Bug 37: `archive_util_$search` finds the archive's length itself, rather than using the one left by the last first_element or first_disected.

Known limitation (not changed)

A 32-character component name runs into the date in `tl` output: the long-form row format (`^32a^17a^5a^16a^a`) gives the name exactly 32 columns, so a name that fills them is followed directly by the date. Widening the column would change a listing format that programs may parse.

Duplicate component names (bug 38). The archive command never creates two components with the same name, but a crafted archive, or two archives joined end to end, can hold them. archive.pl1 matches every component against the names requested, so `t` lists both, `d` deletes both, and `r` replaces both with the same segment; `archive_$get_component` and `get_component_info` find only the first. With `xd` or `xdf`, both are extracted to the same entry, so the second overwrites the first (after the name-duplication query for `xd`, answered yes; without a query for `xdf`), and then both are deleted from the archive: the first component is lost. Found by inspection. Not changed, since only a crafted archive can show it; a change (for example, refusing to delete a component whose extracted copy was overwritten, or rejecting duplicate names altogether) can be considered separately.

The most significant issues are bugs 1, 4, 5, 7, 8 and 28, each of which can destroy or silently misidentify data.

Proposed Changes

The proposed changes are shown by the compare_ascii output below. An build script and changed code are in >udd>m>jjl>dev>MCR10177.

```
----------  archive.pl1

compare_ascii >ldd>sss>source>bound_archive_.s.archive::archive.pl1 ==

A14         1) change(87-10-15,TLNguyen), approve(87-10-15,MCR7774),
A15            audit(87-12-16,GWMay), install(88-01-12,MR12.2-1012):
Changed by B to:
B14         1) change(1987-10-15,TLNguyen), approve(1987-10-15,MCR7774),
B15            audit(1987-12-16,GWMay), install(1988-01-12,MR12.2-1012):


A22         2) change(87-10-15,TLNguyen), approve(87-10-15,MCR7776),
A23            audit(87-12-16,GWMay), install(88-01-12,MR12.2-1012):
Changed by B to:
B22         2) change(1987-10-15,TLNguyen), approve(1987-10-15,MCR7776),
B23            audit(1987-12-16,GWMay), install(1988-01-12,MR12.2-1012):


A35         3) change(87-10-15,TLNguyen), approve(87-10-15,MCR7780),
A36            audit(87-12-16,GWMay), install(88-01-12,MR12.2-1012):
Changed by B to:
B35         3) change(1987-10-15,TLNguyen), approve(1987-10-15,MCR7780),
B36            audit(1987-12-16,GWMay), install(1988-01-12,MR12.2-1012):


Inserted in B:
B44         4) change(2026-09-23,Lippard), approve(2026-09-23,MCR10177):
B45            Fix the bugs listed in MCR10177, including data loss on an unreadable
B46            archive, a record quota overflow, xf over a link and over-long names;
B47            keep the name arrays in a temporary segment instead of stack_space.
Preceding:
A44                                                     END HISTORY COMMENTS */


Inserted in B:
B99       dcl  error_table_$entlong external fixed bin (35);
Preceding:
A95       dcl  error_table_$incorrect_access external fixed bin (35);


Inserted in B:
B108      dcl  error_table_$too_many_args external fixed bin (35);
Preceding:
A103
A104      dcl  archive_key_$last_index external fixed bin (17);


A148      dcl (i, j, k) fixed bin (17);
Changed by B to:
B154      dcl (i, j) fixed bin (17);


A159      /* one record may be enough to hold component names. If not, we open a seg
*/
A160
A161      dcl stack_space (1024) fixed bin (35) init ((1024) 0);
A162
```

```
A163      dcl (sp, new_sp) pointer aligned;
A164      dcl (dcount, lcount) fixed bin (17) aligned;
A165
A166      dcl (NONGLOBAL_ELEMENT_SIZE init (53), GLOBAL_ELEMENT_SIZE init (10))
A167                fixed bin int static options (constant);
A168
A169      dcl 1 nonglobal (2500) aligned based (sp),
A170         2 component_name char (32) aligned,           /* if this structure
changes, change NONGLOBAL_ELEMENT_SIZE */
Changed by B to:
B165      /* The component name arrays are in a temporary segment.  Their dimensions
are
B166         as many elements as fit in a segment of sys_info$max_seg_size (261,120
words):
B167         nonglobal elements are 53 words, global elements 10. */
B168
B169      dcl  sp pointer aligned;
B170
B171      dcl 1 nonglobal (4926) aligned based (sp),
B172         2 component_name char (32) aligned,           /* if this structure
changes, change its dimension */


A176      dcl 1 global (2500) aligned based (sp),
A177         2 gcomponent_name char (32) aligned,          /* if this structure
changes, change GLOBAL_ELEMENT_SIZE */
Changed by B to:
B178      dcl 1 global (26112) aligned based (sp),         /* more than the
components an archive can hold */
B179         2 gcomponent_name char (32) aligned,          /* if this structure
changes, change its dimension */


A206      dcl  buffer char (150) varying;
Deleted by B, preceding:
B208      dcl (optr, cptr, p1_orig) ptr init (null);


A221              2 eptr ptr init (null),                          /* for
archive_aux_ */
A222              2 nptr ptr init (null);                          /* " */
A223
A224      dcl  auxw_ptr ptr;
A225
A226      dcl 1 query_info aligned,                                  /* structure for
command query */
A227           2 version fixed bin init (1),
A228           2 yes_or_no_sw bit (1) unal init ("1"b),            /* require yes
or no answer */
A229           2 supress_name_sw bit (1) unal init ("0"b),         /* print name
with question */
A230           2 extra bit (34) unal,
A231           2 status_code fixed bin (35),                       /* set to code
of prompting question */
A232           2 query_code fixed bin (35);
Changed by B to:
B222              2 eptr ptr,                                      /* for
archive_aux_ */
B223              2 nptr ptr;                                      /* " */
B224
B225      dcl  auxw_ptr ptr;
```

```
A240      dcl 1 delete_acl aligned,                                   /* structure for
deleting one acl */
A241          2 userid char (32),
A242          2 status fixed bin (35);
A243
A244      dcl  mustreprotect bit (1) init (""b);                      /* set to true
if archive is protected */
Changed by B to:
B233      dcl  mustreprotect bit (1) init (""b);                      /* set to true
if access to archive is forced */
B234      dcl  access_switches bit (36) aligned;            /* for update_handler_ */
B235      dcl  old_mode bit (36) aligned;                             /* for
update_handler_ */


A322           get_group_id_ entry returns (char (32) aligned),
Deleted by B, preceding:
B313           get_group_id_$tag_star entry returns (char (32) aligned),


A327          (com_err_, command_query_, ioa_, ioa_$ioa_switch) entry options
(variable),
Changed by B to:
B317          (com_err_, ioa_, ioa_$ioa_switch) entry options (variable),


A337           hcs_$delete_acl_entries entry (char (*) aligned, char (*) aligned, ptr,
fixed bin, fixed bin (35)),
Deleted by B, preceding:
B327           hcs_$chname_seg entry (ptr, char (*) aligned, char (*) aligned, fixed
bin (35)),


A340           hcs_$star_list_ entry (char (*)aligned, char (*), fixed bin (3), ptr,
fixed bin, fixed bin, ptr, ptr, fixed bin (35)
\c),
Deleted by B, preceding:
B329           hcs_$status_long entry (char (*) aligned, char (*) aligned, fixed bin,
ptr, ptr, fixed bin (35)),


Inserted in B:
B337           unique_chars_ entry (bit (*)) returns (char (15)),
B338           update_handler_ entry (char (*), char (*), char (*), bit (36) aligned,
bit (36) aligned, fixed bin (35)),
B339           update_handler_$reprotect entry (char (*), char (*), bit (36) aligned,
bit (36) aligned, fixed bin (35)),
B340           get_temp_segment_ entry (char (*) aligned, ptr, fixed bin (35)),
B341           release_temp_segment_ entry (char (*) aligned, ptr, fixed bin (35)),
Preceding:
A349
A350           dl_handler_ entry (char (*) aligned, char (*) aligned, char (*)
aligned, fixed bin (35)),


Inserted in B:
B346           nd_handler_$force entry (char (*) aligned, char (*) aligned, char (*)
aligned, fixed bin (35)),
Preceding:
A353
A354           archive_util_$first_element entry (ptr, fixed bin (35)),
```

```
A367      dcl (addr, addrel, bin, bit, divide, fixed, max, null, ptr, rel, size,
substr) builtin;
Changed by B to:
B361      dcl (addr, addrel, bin, bit, divide, fixed, hbound, length, max, null, ptr,
rel, rtrim, substr) builtin;


A399                sp = addr (stack_space);
Changed by B to:
B393                sp = null;                                                  /* no
temporary segment yet */


Inserted in B:
B453                call get_temp_segment_ (moi, sp, code); /* for the component name
arrays */
B454                if code ^= 0 then do;
B455                     call com_err_ (code, moi, "Getting a temporary segment.");
B456                     go to COMRETN;
B457                end;
B458
Preceding:
A459                call initiate_file_ (archive_dir, archive_name, R_ACCESS, p1,
orig_bc, code);


Inserted in B:
B461                if p1 = null then if code ^= error_table_$noentry then go to
ERROR_RETURN;
B462                                                                    /* only a
missing archive may be created */
Preceding:
A461                if p1 ^= null then do;


A467                          goto RETURN;
Changed by B to:
B469                          goto COMRETN;


Inserted in B:
B476                     if savecode ^= 0 & savecode ^= 1 & savecode ^= 2 then do;
B477                                                                    /* a status
error, not empty (1) or format error (2) */
B478                          code = savecode;
B479                          go to ERROR_RETURN;
B480                     end;
Preceding:
A474                     if savecode = 2 then do;


A508                if lastarg < 3 then if action = 1 then do;
A509                          call hcs_$star_list_ (my_wdir, "**", 2, null, dcount,
lcount, null, null, code);
A510                          if dcount+lcount > 113 then do;
A511                               call hcs_$make_seg ("", "", "", 01010b, sp, code);
A512                               if code ^= 0 then do;
A513                                    call com_err_ (code, moi);
A514                               go to COMRETN; end;
A515                          end;
A516                     end;
A517
A518                if lastarg * NONGLOBAL_ELEMENT_SIZE > size (stack_space) then do;
A519                     call hcs_$make_seg ("", "", "", 01010b, sp, code);
```

```
A520                    if sp = null then do;
A521                         call com_err_ (code, moi);
A522                    go to COMRETN; end;
A523                    do i = 1 to lastarg-2;
A524                         component_code (i) = 0;
A525                         flags (i) = 0;
A526                         ngtype (i) = ""b;
A527                    end;
Changed by B to:
B515               if lastarg - 2 > hbound (nonglobal, 1) then do;
B516                    call com_err_ (error_table_$too_many_args, moi, "At most ^d
component names can be given.",
B517                         hbound (nonglobal, 1));
B518                    go to COMRETN;


Inserted in B:
B527                         if pathlen > length (component_name (last + 1)) then do;
B528                              code = error_table_$entlong;       /* don't
truncate it to another name */
B529                              go to BADARG;
B530                         end;
Preceding:
A536                         component_name (last+1) = patharg;      /* not a
pathname */


A662                         call com_err_ (code, moi, "^a", pathname_ (archive_dir,
archive_name));
Changed by B to:
B657                         call com_err_ (code, moi, "^a", pathname_
(new_archive_dir, archive_name));


A673                    query_info.status_code = error_table_$moderr;
A674                    call ask_question;                          /* find out if
it's ok to change it */
A675                    seg_acl.userid = get_group_id_ ();          /* wants to
update */
A676                    seg_acl.access = "101"b;                    /* give user rw
*/
A677                    seg_acl.ex_access = "0"b;
A678                    call hcs_$add_acl_entries (new_archive_dir, archive_name,
addr (seg_acl), 1, mcode);
A679                    if mcode ^= 0 then go to MOVE_ERROR;
A680                    else mustreprotect = "1"b;
Changed by B to:
B668                    call update_handler_ ((new_archive_dir), (archive_name),
(moi), access_switches, old_mode, mcode);
B669                    if mcode = 1 then go to COMRETN;            /* user answered
no */
B670                    if mcode ^= 0 then go to MOVE_ERROR;
B671                    mustreprotect = "1"b;                       /* access
forced; restore it when done */


A702      MOVE_ERROR:   call com_err_ (mcode, moi, "Archive ^a not updated.",
pathname_ (archive_dir, archive_name));
Changed by B to:
B693      MOVE_ERROR:   call com_err_ (mcode, moi, "Archive ^a not updated.",
pathname_ (new_archive_dir, archive_name));
B694                    call reprotect;                             /* restore
access to the original */
```

```
B695                    if code ^= 0 then call com_err_ (code, moi, "Could not
restore access to ^a",
B696                         pathname_ (new_archive_dir, archive_name));


A705                    if code = 0 then tptr = null;               /* force
temp.archive to be remade */
A706                    archive_dir = get_pdir_ ();
A707                    if code ^= 0 then archive_name = "temp.archive";
A708                    call ioa_ ("A copy of the updated archive can be found in
[pd]>^a", archive_name);
Changed by B to:
B699                    if code ^= 0 then do;                       /* e.g., a copy
saved by an earlier failure */
B700                         archive_name = rtrim (unique_chars_ ("0"b)) ||
".archive";
B701                         call hcs_$chname_seg (tptr, temp_name, archive_name,
code);
B702                    end;
B703                    archive_dir = get_pdir_ ();
B704                    if code = 0 then do;                        /* the copy is
no longer the temp: */
B705                         call hcs_$terminate_noname (tptr, fix17);
B706                         tptr = null;                           /* force the
temp to be remade */
B707                         cleanup_temp = ""b;
B708                         call ioa_ ("A copy of the updated archive can be found
in [pd]>^a", archive_name);
B709                    end;
B710                    else do;
B711                         archive_name = temp_name;
B712                         call ioa_ ("A copy of the updated archive can be found
in [pd]>^a until the next ^a command.",
B713                              temp_name, moi);
B714                    end;


A712               if savecode ^= 0 then call com_err_ (savecode, moi, "^a",
pathname_ (archive_dir, archive_name));
Changed by B to:
B718               if savecode ^= 0 then call com_err_ (savecode, moi, "^a",
pathname_ (new_archive_dir, archive_name));


A717                    delete_acl.userid = seg_acl.userid;         /* delete ACL */
A718                    call hcs_$delete_acl_entries (new_archive_dir, archive_name,
addr (delete_acl), 1, code);
A719                    if code ^= 0 then do;
A720                         call com_err_ (code, moi, "^a", pathname_ (archive_dir,
archive_name));
Changed by B to:
B723                    call reprotect;
B724                    if code ^= 0 then do;
B725                         call com_err_ (code, moi, "^a", pathname_
(new_archive_dir, archive_name));


Inserted in B:
B766                         if copy then dn = new_archive_dir;     /* the copy was
changed, not the original */
B767                         else dn = archive_dir;
Preceding:
A761                         call ioa_ ("^[archive: ^;^9x^]^a appended to ^a",
first_line_sw,
```

```
A763                                pathname_ (archive_dir, archive_name));
Changed by B to:
B770                                pathname_ (dn, archive_name));


A769                        else if iflag = 6                           /* Temp, could
use 2 if error code were available */
A770                        then call com_err_ (0, moi, "Archive segment overflow. Could
not ^a ^a in ^a",
A771                             act_com, pathname_ (component_path (i), component_name
(i)), pathname_ (archive_dir, archive_name));
Changed by B to:
B776                        else if iflag = 6 then do;                  /* Temp, could
use 2 if error code were available */
B777                             if copy then dn = new_archive_dir;
B778                             else dn = archive_dir;
B779                             call com_err_ (0, moi, "Archive segment overflow. Could
not ^a ^a in ^a",
B780                             act_com, pathname_ (component_path (i), component_name
(i)), pathname_ (dn, archive_name));
B781                        end;


A778                             last = 0, archive_dir, pathname_ (archive_dir,
archive_name));
Changed by B to:
B788                             last = 0, my_wdir, pathname_ (archive_dir,
archive_name));


A781                             pathname_ (archive_dir, archive_name), last = 0,
archive_dir);
Changed by B to:
B791                             pathname_ (archive_dir, archive_name), last = 0,
my_wdir);


A793                        if sp ^= addr (stack_space) then do; call hcs_$delentry_seg
(sp, code);
A794                        call hcs_$terminate_noname (sp, code); end;
Changed by B to:
B803                        call reprotect;
B804                        if sp ^= null then call release_temp_segment_ (moi, sp,
code);


A816                    wdct = divide (bcnt+35, 36, 17, 0);
Changed by B to:
B826                    wdct = divide (bcnt+35, 36, 19, 0);


A835                        if nonglobal (i).flags = 1 then do;         /* found an
existing archive component */
A836                             if delete then do;
A837                                  nonglobal (i).flags = 7;           /* indicate that
no delete */
A838                                  call ccmp;                         /* copy this
existing component in temp archive */
A839                             end;
A840                        end;
A841
```

```
A842                    nonglobal (i).component_code = code;         /* save error
code for printing an error message */
Changed by B to:
B845                    if delete then do;                           /* not
extracted, so keep it in the archive */
B846                         if i ^= 0 then nonglobal (i).flags = 7; /* indicate that
no delete */
B847                         call ccmp;                              /* copy this
existing component in temp archive */
B848                    end;
B849
B850                    if i ^= 0 then nonglobal (i).component_code = code; /* save
error code for printing error message */


Inserted in B:
B860                    call hcs_$terminate_noname (cptr, fix17);    /* the existing
entry (or link target) */
Preceding:
A852                    if ^force then call nd_handler_ (moi, initpath, p1 ->
archive.name, code);


A855                         call delete_seg (initpath, p1 -> archive.name, bit
(typef, 2), code);
Changed by B to:
B864                         if code = 0 & typef = 0 then            /* a link:
unlink it, as copy -force does */
B865                              call nd_handler_$force (moi, initpath, p1 ->
archive.name, code);
B866                         else call delete_seg (initpath, p1 -> archive.name, bit
(typef, 2), code);


A935                         flags (i) = 3;
A936                         go to MUSTCOPY;
Changed by B to:
B946                         go to MUSTCOPY;                         /* (MUSTCOPY
sets flags (i) if not global) */


Inserted in B:
B986                         call hcs_$terminate_noname (optr, code);
Preceding:
A976                         go to MUSTCOPY;


A990                              if dlast * GLOBAL_ELEMENT_SIZE > size (stack_space)
then do;  /* need more room */
A991                                   call hcs_$make_seg ("", "", "", 01010b,
new_sp, code);
A992                                   if new_sp = null then do;
A993                                        call com_err_ (code, moi);
A994                                        go to COMRETN;
A995                                   end;
A996                                   do k = 1 to dlast - 1;        /* copy from
stack_space to allocated seg */
A997                                        new_sp -> global (k) = sp -> global (k);
A998                                   end;
A999                                   sp = new_sp;
A1000                             end;
Deleted by B, preceding:
B1001                             gflags (dlast) = 1;
```

```
A1008                    wdct = divide (bcnt+35, 36, 17, 0);
Changed by B to:
B1008                    wdct = divide (bcnt+35, 36, 19, 0);


Inserted in B:
B1022                              call hcs_$terminate_noname (optr, code);
Preceding:
A1022                              go to MUSTCOPY2;


Inserted in B:
B1032                         call hcs_$terminate_noname (optr, code);
Preceding:
A1031                         go to MUSTCOPY2;                        /* Don't update
"flags" */


A1078                    wdct = divide (bcnt+35, 36, 17, 0);          /* convert to
word count */
Changed by B to:
B1080                    wdct = divide (bcnt+35, 36, 19, 0);          /* convert to
word count */


A1087
Changed by B to:
B1089                         if copy then call hcs_$delentry_seg (p2, code); /* don't
leave an incomplete copy */


A1120                         if code ^= 0 then do;
Changed by B to:
B1122                         if error ^= 0 then do;


A1127                              call ioa_ ("archive: Creating ^a", pathname_
(archive_dir, archive_name));
Changed by B to:
B1129                              call ioa_ ("archive: Creating ^a", pathname_
(new_archive_dir, archive_name));


A1204     ask_question: proc;
A1205
A1206     /* Procedure to ask the user whether to update a protected segment */
A1207
A1208                    call command_query_ (addr (query_info), buffer, moi,
A1209                         "Do you want to update the protected segment ^a ?",
pathname_ (new_archive_dir, archive_name));
A1210
A1211                    if substr (buffer, 1, 2) = "no" then goto COMRETN;
A1212
A1213               end ask_question;
Changed by B to:
B1206     reprotect: proc;
B1207     /* Restore the user's access to a protected archive, if update_handler_
forced it.
B1208        Sets code; called on success, on error, and from clean_up. */
B1209
B1210               code = 0;
B1211               if ^mustreprotect then return;
B1212               mustreprotect = ""b;           /* only once */
```

```
B1213               call update_handler_$reprotect ((new_archive_dir), (archive_name),
access_switches, old_mode, code);
B1214
B1215               end reprotect;

Comparison finished: 53 differences, 264 lines.

----------  archive_.pl1

compare_ascii >ldd>sss>source>bound_archive_.s.archive::archive_.pl1 ==

Inserted in B:
B11
B12       /****^  HISTORY COMMENTS:
B13         1) change(2026-09-23,Lippard), approve(2026-09-23,MCR10177):
B14            Return no_component for a name longer than 32 characters rather than
B15            truncating it; free the component list in the caller's area on a format
B16            error; check that a whole header lies within the archive; return
bad_arg
B17            for a component_ptr that does not follow a header.
B18                                                          END HISTORY COMMENTS */
B19
Preceding:
A11       archive_: proc ();


A50       dcl  P_n_components fixed bin;                              /* Output:
number of components in archive */
Changed by B to:
B59       dcl  P_n_components fixed bin parameter;                    /* Output:
number of components in archive */


A84       dcl (addrel, baseno, binary, divide, ltrim, null, pointer, rel, rtrim, size,
string, substr, unspec, verify) builtin;
Changed by B to:
B93       dcl (addrel, baseno, binary, divide, length, ltrim, null, pointer, rel,
rtrim, size, string, substr, unspec, verify) buil
\ctin;


Inserted in B:
B120                if length (rtrim (P_component_name)) > length (component_name)
then
B121                     call FINISH (error_table_$no_component);     /* no component
can have this name */
B122
Preceding:
A111                component_name = P_component_name;


A131                else archive_component_info = comp_info;                    /*
Just fill in the structure from our copy */
Changed by B to:
B143                else archive_component_info = comp_info;          /* Just fill in
the structure from our copy */


Inserted in B:
B189                     if (header_ptr -> archive_header.header_begin ^=
archive_data_$ident)
B190                          | (header_ptr -> archive_header.header_end ^=
archive_data_$header_end) then
```

```
B191                         call FINISH (error_table_$bad_arg);     /* Must follow a
header */
Preceding:
A177                    call GET_COMPONENT_INFO;


A266                    free comp_list;
Changed by B to:
B281                    free comp_list in (output_area);


Inserted in B:
B352                    if binary (rel (header_ptr), 18) + size (archive_header) >
archive_size then
B353                         goto FORMAT_ERROR;                      /* Not room for
a whole header */
Preceding:
A337                    end;

Comparison finished: 8 differences, 25 lines.

----------  archive_aux_.pl1

compare_ascii >ldd>sss>source>bound_archive_.s.archive::archive_aux_.pl1 ==

Inserted in B:
B11
B12       /****^  HISTORY COMMENTS:
B13         1) change(2026-09-23,Lippard), approve(2026-09-23,MCR10177):
B14            listwdir returns no entries if hcs_$star_list_ fails, and makes the
B15            caller responsible for freeing before calling it; free locates the
B16            system free area itself; inwdir ignores directories and links to
B17            directories.
B18                                                        END HISTORY COMMENTS */
B19
Preceding:
A11       /*


A38       dcl       (get_system_free_area_, freen_) entry(ptr);
Changed by B to:
B47       dcl       get_system_free_area_ entry () returns (ptr);


A53                 2 mustfree bit(1),            /* set to one after allocation
complete */
Changed by B to:
B62                 2 mustfree bit(1),            /* set by listwdir if the caller
must free */


A76
A77                 call get_system_free_area_(areap);              /* get area for
star handler */
Changed by B to:
B85                 areap = get_system_free_area_ ();               /* get area for
star handler */
B86                 eptr, nptr = null;                              /*
hcs_$star_list_ has allocated nothing yet */
B87                 ecount, lcount = 0;
B88                 mustfree = "1"b;                                /* the caller
frees whatever is allocated */
```

```
A80                 ecount = ecount + lcount;               /* Update by number of
links, to get total entries */
A81                 if ecount > 0 then mustfree = "1"b;
Changed by B to:
B91
B92                 if eptr = null then do;                           /* nothing
allocated (an error, or no entries) */
B93                      mustfree = ""b;
B94                      ecount = 0;
B95                 end;
B96                 else ecount = ecount + lcount;                    /* Update by
number of links, to get total entries */


A96
Changed by B to:
B111      dcl DIRECTORY bit(2) internal static options(constant) init("10"b);


A102                     if type then dtm = eptr->branches.dtm(k);             /* Is
branch (really should check non-dir) */
Changed by B to:
B117                     if type = DIRECTORY then go to nfnd;         /* Not a segment
*/
B118                     if type then dtm = eptr->branches.dtm(k);    /* Is a segment
*/


Inserted in B:
B122                          if stat.type = DIRECTORY then go to nfnd; /* Link to a
directory */
Preceding:
A106                          dtm = stat.dtm;                         /* Set date-
time-seg updated */


Inserted in B:
B137                areap = get_system_free_area_ ();                 /* where
listwdir allocated */
Preceding:
A120                if eptr ^= null then free eptr -> branches in (area);

Comparison finished: 9 differences, 34 lines.

----------  archive_sort.pl1

compare_ascii >ldd>sss>source>bound_archive_.s.archive::archive_sort.pl1 ==

Inserted in B:
B11
B12       /****^  HISTORY COMMENTS:
B13         1) change(2026-11-02,Lippard), approve(2026-11-02,MCR10177):
B14            Change update_handler_ declarations to use bit (36) aligned.
B15                                                        END HISTORY COMMENTS */
B16
Preceding:
A11       /* ARCHIVE_SORT - Archive Segment Sorting Program.


A66            update_handler_ entry (char(*) aligned, char(*) aligned, char(*)
aligned, bit(36),bit(36),fixed bin(35)),
A67            update_handler_$reprotect entry (char(*) aligned, char(*) aligned,
bit(36),bit(36),fixed bin(35)),
Changed by B to:
```

```
B72            update_handler_ entry (char(*) aligned, char(*) aligned, char(*)
aligned, bit(36) aligned,bit(36) aligned,fixed bin(
\c35)),
B73            update_handler_$reprotect entry (char(*) aligned, char(*) aligned,
bit(36) aligned,bit(36) aligned,fixed bin(35)),


A74       dcl access_switches bit (36),                              /* returned by
update_handler_:
Changed by B to:
B80       dcl access_switches bit (36) aligned,                                /*
returned by update_handler_:


A77            old_mode bit (36);                                    /* previous mode
if any */
Changed by B to:
B83            old_mode bit (36) aligned;                                   /*
previous mode if any */

Comparison finished: 4 differences, 14 lines.

----------  archive_util_.pl1

compare_ascii >ldd>sss>source>bound_archive_.s.archive::archive_util_.pl1 ==

Inserted in B:
B11
B12       /****^  HISTORY COMMENTS:
B13         1) change(2026-09-23,Lippard), approve(2026-09-23,MCR10177):
B14            Validate headers as archive_ does (fence required, obsolete
B15            header_begin rejected, bit count checked, whole header and component
B16            within a word-aligned archive); widen offsets; correct declarations;
B17            clear first_disected's outputs on failure; search finds the archive
B18            length itself; return hcs_$status_mins's full error code.
B19                                                    END HISTORY COMMENTS */
B20
Preceding:
A11       archive_util_$next_element: proc (header_ptr, code);


A15          expect pointer to archive segment in header_ptr
A16          return code = 0 and header_ptr set to next element
A17          unless at end of segment or if a format error
A18          where header_ptr is left as given and code set to 1 if at end of segment
A19          or to 2 if a format error is discovered.
A20
A21          first_element or first_dissected must be called first.
A22          thereafter, next_element or next_dissected must be called.
Changed by B to:
B25          expect pointer to an archive header in header_ptr
B26          return code = 0 and header_ptr set to next element,
B27          or code = 1 at the end of the archive, or code = 2 if a format error
B28          is discovered, or a status code if the archive's status cannot be
B29          obtained.  On a nonzero code, next_element and first_element set
B30          header_ptr to null; search leaves it as given.
B31
B32          first_element or first_disected must be called first.
B33          thereafter, next_element or disected_element must be called.
B34          search finds the archive's length itself.


A29       dcl  header_ptr ptr;
A30       dcl  header_save ptr;
```

```
A31
A32       dcl  header_length_bits init (900) fixed bin static;
A33
A34       dcl (code, scode, stype, icode) fixed bin (17),
A35            bitcnt fixed bin (24),
A36            highoffset internal static fixed bin (17);
A37
A38       dcl  flag fixed bin (17);
A39
A40       dcl  cv_dec_ entry (char (*) aligned, fixed bin (24)),
A41            hcs_$status_mins external entry (ptr, fixed bin (17), fixed bin (24),
fixed bin (17));
A42
A43       dcl  n fixed bin (24);
Changed by B to:
B41       dcl  header_ptr ptr parameter;
B42       dcl  code fixed bin (35) parameter;
B43       dcl  segptr ptr parameter;
B44       dcl  segname char (32) aligned parameter;
B45       dcl  bit_count fixed bin (24) parameter;
B46
B47       dcl  header_save ptr;
B48
B49       dcl  header_length_bits fixed bin internal static options (constant) init
(900);
B50
B51       dcl  entry_type fixed bin;                                  /* which entry
was called, for its return protocol */
B52       dcl (ELEMENT init (0), DISECTED init (1), SEARCH init (2))
B53                 fixed bin internal static options (constant);
B54
B55       dcl  stype fixed bin (2),
B56            bitcnt fixed bin (24),
B57            highoffset internal static fixed bin (19);             /* archive
length in words */
B58
B59       dcl  hcs_$status_mins external entry (ptr, fixed bin (2), fixed bin (24),
fixed bin (35));
B60
B61       dcl  (n, next_offset) fixed bin (35);


A48       dcl  archive_data_$header_begin ext char (8) aligned;
A49
A50       /* builtins */
A51
A52       dcl (addrel, bin, divide, null, rel, size) builtin;
Changed by B to:
B66       dcl  archive_data_$fence ext char (8) aligned;
B67
B68       /* builtins */
B69
B70       dcl (addwordno, convert, divide, ltrim, null, ptr, rtrim, size, verify,
wordno) builtin;


A58                 flag, icode = 0;
A59
A60       start:
A61                 call cv_dec_ (header_ptr -> archive_header.bit_count, n);
A62                 next = addrel (header_ptr, divide (n + header_length_bits + 35,
36, 17, 0));
A63                 if bin (rel (next), 17) < highoffset then go to continue;
A64                 if bin (rel (next), 17) = highoffset then do;
```

```
A65                     icode = 1;header_ptr = null; go to comretn;
A66                end;
A67                icode = 2;header_ptr = null;go to comretn;
A68
A69           continue:
A70                header_ptr = null;                                  /* initialize
for end of archive state */
A71                if next -> archive_header.header_begin = archive_data_$ident then
header_ptr = next;
A72                else if next -> archive_header.header_begin =
archive_data_$header_begin then header_ptr = next;
A73                if header_ptr ^= next then icode = 2;               /* format error
*/
A74
A75           comretn:
A76                if flag = 1 then go to disected_return;
A77                if flag = 2 then go to search_return;
A78                code = icode;
Changed by B to:
B76                entry_type = ELEMENT;
B77                code = 0;
B78
B79           start:
B80                n = BIT_COUNT (header_ptr);
B81                if n < 0 then go to format_error;
B82                next_offset = wordno (header_ptr) + divide (n + header_length_bits
+ 35, 36, 35, 0);
B83                if next_offset = highoffset then do;
B84                     code = 1;header_ptr = null; go to comretn;
B85                end;
B86
B87           continue:                                                /* next_offset
is the offset of the next header */
B88                if next_offset + size (archive_header) > highoffset then go to
format_error;
B89                next = ptr (header_ptr, next_offset);
B90                header_ptr = null;                                  /* initialize
for end of archive state */
B91                if next -> archive_header.header_begin = archive_data_$ident then
B92                     if next -> archive_header.header_end = archive_data_$fence
then do;
B93                          n = BIT_COUNT (next);                     /* the component
must be there too */
B94                          if n >= 0 then
B95                               if next_offset + size (archive_header) + divide (n
+ 35, 36, 35, 0) <= highoffset then
B96                                    header_ptr = next;
B97                     end;
B98                if header_ptr ^= next then code = 2;               /* format error
*/
B99                go to comretn;
B100
B101          format_error:
B102               code = 2;
B103               header_ptr = null;
B104
B105          comretn:
B106               if entry_type = DISECTED then go to disected_return;
B107               if entry_type = SEARCH then go to search_return;


A83                icode, flag = 0;
Changed by B to:
B112               entry_type = ELEMENT;
```

```
B113                code = 0;


A87                 call hcs_$status_mins (header_ptr, stype, bitcnt, scode);
A88                 if scode ^= 0 then do;
A89                      icode = scode;
A90                      go to comretn;
A91                 end;
A92                 highoffset = divide (bitcnt+35, 36, 17, 0);
A93                 if highoffset = 0 then do;
A94                      icode = 1;
Changed by B to:
B117                call hcs_$status_mins (header_ptr, stype, bitcnt, code);
B118                if code ^= 0 then go to comretn;
B119                highoffset = divide (bitcnt, 36, 19, 0);
B120                if highoffset * 36 ^= bitcnt then go to format_error; /* an
archive is a whole number of words */
B121                if highoffset = 0 then do;
B122                     code = 1;


A97                 next = header_ptr;
A98                 go to continue;
A99       \014
A100      disected_element: entry (header_ptr, segptr, segname, bit_count, code5);
A101
A102      dcl  segptr ptr,
A103           code5 fixed bin (17),
A104           segname char (32) aligned,
A105           bit_count fixed bin (24);
A106
A107                flag = 1;
A108                icode = 0;
Changed by B to:
B125                next_offset = wordno (header_ptr);
B126                go to continue;
B127      \014
B128      disected_element: entry (header_ptr, segptr, segname, bit_count, code);
B129
B130                entry_type = DISECTED;
B131                code = 0;


A112                if header_ptr = null then
Changed by B to:
B135                if header_ptr = null | code ^= 0 then


A117                     code5 = icode;
A118                     return;
A119                end;
A120
A121                call cv_dec_ (header_ptr -> archive_header.bit_count, bit_count);
A122                segptr = addrel (header_ptr, size (archive_header));
A123                segname = header_ptr -> archive_header.name;
A124                code5 = icode;
A125                return;
A126                                                                  /* \013 */
A127      first_disected: entry (header_ptr, segptr, segname, bit_count, code5);
A128
A129                flag = 1;
A130                icode = 0;
A131                go to first_elt2;
A132      \014
```

```
A133      search:   entry (header_ptr, segptr, segname, code4);
A134
A135      dcl  code4 fixed bin (17);
A136
A137                flag = 2;
A138                icode = 0;
A139                header_save = header_ptr;
A140                next = header_ptr;
A141                go to continue;
A142
A143      search_return:
A144                if header_ptr = null then do;
A145                     segptr = null;
A146                     code4 = icode;
Changed by B to:
B140                     return;
B141                end;
B142
B143                bit_count = BIT_COUNT (header_ptr);
B144                segptr = addwordno (header_ptr, size (archive_header));
B145                segname = header_ptr -> archive_header.name;
B146                return;
B147                                                                  /* \013 */
B148      first_disected: entry (header_ptr, segptr, segname, bit_count, code);
B149
B150                entry_type = DISECTED;
B151                code = 0;
B152                go to first_elt2;
B153      \014
B154      search:   entry (header_ptr, segptr, segname, code);
B155
B156                entry_type = SEARCH;
B157                code = 0;
B158                header_save = header_ptr;
B159                go to first_elt2;                                 /* set
highoffset from this archive */
B160
B161      search_return:
B162                if header_ptr = null | code ^= 0 then do;
B163                     segptr = null;


A152                segptr = addrel (header_ptr, size (archive_header));
A153                code4 = icode;
Changed by B to:
B169                segptr = addwordno (header_ptr, size (archive_header));


Inserted in B:
B172      \014
B173      /* The bit count field of a header, or -1 if it is not a number.  Only
B174         digits (with surrounding blanks) are accepted, as archive_ requires;
B175         convert alone would also accept a sign, a decimal point or an exponent.
*/
B176
B177      BIT_COUNT: proc (P_header_ptr) returns (fixed bin (35));
B178      dcl  P_header_ptr ptr parameter;
B179      dcl  digits char (8) varying;
B180      dcl  result fixed bin (35);
B181
B182                digits = ltrim (rtrim (P_header_ptr -> archive_header.bit_count));
B183                if digits = "" then return (-1);
B184                if verify (digits, "0123456789") ^= 0 then return (-1);
```

```
B185                return (convert (result, digits));      /* at most 8 digits, so it
always fits */
B186
B187           end BIT_COUNT;
Preceding:
A156
A157           end                                                    /* archive_util_
*/;

Comparison finished: 12 differences, 238 lines.

----------  reorder_archive.pl1

compare_ascii >ldd>sss>source>bound_archive_.s.archive::reorder_archive.pl1 ==

A16         1) change(87-10-07,TLNguyen), approve(87-10-07,MCR7773),
A17            audit(87-12-03,GWMay), install(88-01-12,MR12.2-1012):
Changed by B to:
B16         1) change(1987-10-07,TLNguyen), approve(1987-10-07,MCR7773),
B17            audit(1987-12-03,GWMay), install(1988-01-12,MR12.2-1012):


Inserted in B:
B21         2) change(2026-11-02,Lippard), approve(2026-11-02,MCR10177):
B22            Change update_handler_ declarations to use bit (36) aligned.
Preceding:
A21                                                    END HISTORY COMMENTS */


A98       dcl  access_switches bit (36),                           /* returned by
update_handler_
Changed by B to:
B100      dcl  access_switches bit (36) aligned,                   /* returned by
update_handler_


A101           old_mode bit (36);                                   /* previous mode
if any */
Changed by B to:
B103           old_mode bit (36) aligned;                                  /*
previous mode if any */


A140           update_handler_ entry (char (*), char (*), char (*), bit (36), bit
(36), fixed bin (35)),
A141           update_handler_$reprotect entry (char (*), char (*), bit (36), bit
(36), fixed bin (35));
Changed by B to:
B142           update_handler_ entry (char (*), char (*), char (*), bit (36) aligned,
bit (36) aligned, fixed bin (35)),
B143           update_handler_$reprotect entry (char (*), char (*), bit (36) aligned,
bit (36) aligned, fixed bin (35));

Comparison finished: 5 differences, 14 lines.

----------  update_handler_.pl1

compare_ascii >ldd>sss>source>bound_archive_.s.archive::update_handler_.pl1 ==

Inserted in B:
B11
B12       /****^  HISTORY COMMENTS:
B13         1) change(2026-09-23,Lippard), approve(2026-09-23,MCR10177):
B14            Query uses pathname_, so an entry in the root is not shown as >>name;
```

```
B15            use the query_info and acl_structures include files; access_switches
B16            and old_mode are bit (36) aligned.
B17                                                        END HISTORY COMMENTS */
B18
Preceding:
A11       /* Procedure to interrogate user if he wishes to un-protect a protected
segment for updating purposes.


A21            access_switches bit (36),                                  /* whether
access was forced */
A22            old_mode bit (36),                                         /* user's
previous mode to restore */
Changed by B to:
B29            access_switches bit (36) aligned,            /* whether access was
forced */
B30            old_mode bit (36) aligned,                                 /* user's
previous mode to restore */


A25                                                                          = 1 if no
delete,
Changed by B to:
B33                                                                          = 1 if user
answered no,


A28       dcl access_forced bit (1) def (access_switches) pos (1);  /* whether access
was forced */
A29       dcl old_acl_entry bit (1) def (access_switches) pos (2);    /* whether an
entry for the user already existed */
Changed by B to:
B36       dcl 1 switches aligned based (addr (access_switches)),      /* the bits of
access_switches */
B37           2 access_forced bit (1) unaligned,                      /* whether
access was forced */
B38           2 old_acl_entry bit (1) unaligned,                      /* whether an
entry for the user already existed */
B39           2 pad bit (34) unaligned;


A37            hcs_$list_acl entry (char (*), char (*), ptr, ptr, ptr, fixed bin,
fixed bin (35));
A38
A39       declare 1 query_info aligned,                                /* Structure for
command_query_ */
A40               2 version fixed bin init (1),
A41               2 yes_or_no_sw bit (1) unal init ("1"b),             /* require yes
or no */
A42               2 suppress_name_sw bit (1) unal init ("0"b),         /* print name
with question */
A43               2 status_code fixed bin (35),                        /* set to code
of prompting question */
A44               2 query_code fixed bin (35) init (0);
A45
A46       dcl 1 seg_acl aligned,                                       /* structure
used to add entry to acl */
A47           2 userid char (32) aligned,
A48           2 mode bit (36) aligned,
A49           2 ex_mode bit (36) aligned,
A50           2 status_code fixed bin (35);
A51
A52       dcl 1 del_acl aligned,                                       /* structure
used to delete entry from acl */
```

```
A53           2 userid char (32),
A54           2 status_code fixed bin (35);
Changed by B to:
B47            hcs_$list_acl entry (char (*), char (*), ptr, ptr, ptr, fixed bin,
fixed bin (35)),
B48            pathname_ entry (char (*), char (*)) returns (char (168));
B49
B50       dcl 1 seg_acl aligned like segment_acl_entry;     /* structure used to add
entry to acl */
B51       dcl 1 del_acl aligned like delete_acl_entry;               /* structure
used to delete entry from acl */


Inserted in B:
B63                 query_info.version = query_info_version_6;
B64                 query_info.yes_or_no_sw = "1"b;                  /* require yes
or no */
Preceding:
A66                 query_info.status_code = error_table_$moderr;


A68                 "Do you want to update the protected segment ^R^a>^a^B?", path,
name);
Changed by B to:
B67                 "Do you want to update the protected segment ^R^a^B?", pathname_
(path, name));


A78                 seg_acl.userid = get_group_id_ ();
Changed by B to:
B77                 seg_acl.access_name = get_group_id_ ();


A90                 seg_acl.ex_mode = "0"b;
Changed by B to:
B89                 seg_acl.extended_mode = "0"b;


A104                     del_acl.userid = get_group_id_ ();
Changed by B to:
B103                     del_acl.access_name = get_group_id_ ();


A109                     seg_acl.userid = get_group_id_ ();
A110                     seg_acl.mode = old_mode;
A111                     seg_acl.ex_mode = "00"b;
Changed by B to:
B108                     seg_acl.access_name = get_group_id_ ();
B109                     seg_acl.mode = old_mode;
B110                     seg_acl.extended_mode = "00"b;


A115                                                              /* remove user
from acl */
A116
A117                return;
Changed by B to:
B114
B115                return;
B116
B117      %include query_info;
B118      %include acl_structures;

Comparison finished: 12 differences, 67 lines.
```

```
----------  archive.info

compare_ascii >doc>info>archive.info ==

A1        03/12/84  archive, ac
Changed by B to:
B1        :Info: archive: ac:  2026-09-23  archive, ac


A20          operations.  The star and equal conventions cannot be used.
Changed by B to:
B20          operations.  For append, replace, update, and extract operations,
B21          each path is the pathname of a segment whose entryname is the name
B22          of the component.  The star and equal conventions cannot be used.


A35          operates like tl, but suppresses the column headings and the time
A36          portion of dates.
Changed by B to:
B37          prints the table of contents in long form, briefly; operates like
B38          tl, except that the title and column headings are suppressed.


A55          of the new archive segment created in your working directory.
Changed by B to:
B57          of the archive segment created in your working directory.


A111         copies, update, and deletes; operates like ud, placing an updated
Changed by B to:
B113         copies, updates, and deletes; operates like ud, placing an updated


A135         directory.  The archive segment is not modified.
Changed by B to:
B137         directory.  The archive segment is not modified.  If an entry with
B138         the name of the component already exists where the segment is to be
B139         created, x follows the duplicate name convention of the copy
B140         command: you are asked whether to delete it or, if it is a link,
B141         whether to unlink it.
B142
B143


A144         created.
Changed by B to:
B152         created, as copy -force does.  A link found there is unlinked; its
B153         target is not affected.


A153      with either the append or replace operation.  In each of the
A154      operations that add to or replace components of the archive, the
A155      original segment is copied and the copy is written into the archive,
A156      leaving the original segment untouched unless deletion is specified as
A157      part of the operation.
Changed by B to:
B162      with either the append or replace operation.  In each of the operations
B163      that add to or replace components of the archive, the original segment
B164      is copied and the copy is written into the archive, leaving the
B165      original segment untouched unless deletion is specified as part of the
B166      operation.
```

```
Inserted in B:
B173      The copy operations (ca, cr, cu, cd and their variants) cannot be used
B174      when the archive segment is in your working directory, since the copy
B175      would have the same name as the original.
B176
B177      For segment deletions after replacement requests (ad, rd, ud and their
B178      variants), if the specified component name is a link to a segment, the
B179      segment linked to is deleted.  The link is not unlinked.
B180
B181
B182      If an operation that changes an archive segment is stopped by a record
B183      quota overflow in the directory containing it, the archive segment is
B184      not changed.  A message gives
B185      the name of a copy of the updated archive segment in the process
B186      directory; it is the only copy of the update, and is replaced by the
B187      next archive command in the process unless renamed.
B188
Preceding:
A164      Each component of an archive segment retains certain attributes of


Inserted in B:
B194
B195
B196      :hcom:
B197      /****^  HISTORY COMMENTS:
B198        1) change(2026-09-25,Lippard), approve(2026-11-02,MCR10177):
B199           Add explanation of what paths are for append, replace, update,
B200           and extract, table and delete; correct tlb description, minor
B201           wording errors. Add MPM description on behavior of x/xf and
B202           statement that after replacement requests delete a link's target,
B203           copy operations can't be used on archive in working dir, and
B204           what happens after a record quota overflow.
B205                                                        END HISTORY COMMENTS */
B206
At end

Comparison finished: 10 differences, 64 lines.

----------  archive_.info

compare_ascii >doc>info>archive_.info ==

A1        09/22/86  archive_
Changed by B to:
B1        :Info: archive_:
B2        2026-09-23  archive_


A4        archives, list the components of an archive, and obtain information
Changed by B to:
B5        archives, list the components of an archive, and obtain information


A12       :Entry: get_component:  03/18/81  archive_$get_component
Changed by B to:
B13       :Entry: get_component:  2026-09-25  archive_$get_component


A37         32 characters long.  (Input)
Changed by B to:
B38         32 characters long; for a longer name, error_table_$no_component is
B39         returned.  If the archive contains more than one component with this
B40         name, the first is found.  (Input)
```

```
B41
B42


A56       :Entry: get_component_info:  03/18/81  archive_$get_component_info
Changed by B to:
B61       :Entry: get_component_info:  2026-09-25  archive_$get_component_info


A83          32 characters long.  (Input)
Changed by B to:
B88          32 characters long; for a longer name, error_table_$no_component is
B89          returned.  If the archive contains more than one component with this
B90          name, the first is found.  (Input)


A102      Structure:
A103      The archive_component_info_ptr points to the following structure
A104      (described in the archive_component_info.incl.pl1 include file):
A105
A106           dcl 1 archive_component_info aligned based
A107                 (archive_component_info_ptr),
A108                 2 version fixed bin,
A109                 2 comp_bc fixed bin (24),
A110                 2 comp_ptr ptr,
A111                 2 name char (32) unaligned,
A112                 2 time_modified fixed bin (71),
A113                 2 time_updated fixed bin (71),
A114                 2 comp_lth fixed bin (19),
A115                 2 access bit (36) unaligned;
A116
A117
Deleted by B, preceding:
B109      List of structure elements:


A158      :Entry: next_component:  03/18/81  archive_$next_component
Changed by B to:
B149      Notes on the archive_component_info structure:
B150      The archive_component_info_ptr points to the following structure
B151      (described in the archive_component_info.incl.pl1 include file):
B152
B153           dcl 1 archive_component_info aligned based
B154                 (archive_component_info_ptr),
B155                 2 version fixed bin,
B156                 2 comp_bc fixed bin (24),
B157                 2 comp_ptr ptr,
B158                 2 name char (32) unaligned,
B159                 2 time_modified fixed bin (71),
B160                 2 time_updated fixed bin (71),
B161                 2 comp_lth fixed bin (19),
B162                 2 access bit (36) unaligned;
B163
B164
B165      :Entry: next_component:  2026-09-25  archive_$next_component


A197         is a standard system status code, one of the following;   (Output)
A198                 error_table_$not_archive
A199                 error_table_$archive_fmt_err
A200
A201
A202      :Entry: next_component_info:  03/18/81  archive_$next_component_info
Changed by B to:
```

```
B204         is a standard system status code, one of the following:  (Output)
B205                  error_table_$bad_arg
B206                  error_table_$not_archive
B207                  error_table_$archive_fmt_err
B208         error_table_$bad_arg is returned if component_ptr is not null and
B209         does not point to the base of a component in the archive.
B210
B211
B212      :Entry: next_component_info:  2026-09-25  archive_$next_component_info


A207      returns a pointer to the next component in the archive and fills in a
Changed by B to:
B217      returns a pointer to the next component in the archive and fills in an


A230      component_name
A231         is the name of the component to be searched for.  It may be up to
A232         32 characters long.  (Input)
Changed by B to:
B240      component_ptr
B241         on input, this is a pointer to the previous component in the
B242         archive, or null to indicate that the next component should be the
B243         first component in the archive.  On output, this is a pointer to the
B244         next component in the archive, or null if there are no components
B245         remaining after the one it pointed to on input.  (Input/Output)


Inserted in B:
B256
B257
Preceding:
A243      code


Inserted in B:
B261                  error_table_$bad_arg
Preceding:
A246                  error_table_$not_archive


A249
A250
A251      :Entry: list_components:  03/18/81  archive_$list_components
Changed by B to:
B265         error_table_$bad_arg is returned if component_ptr is not null and
B266         does not point to the base of a component in the archive.
B267
B268
B269      :Entry: list_components:  2026-09-25  archive_$list_components


A286         archive.  (Input)
Changed by B to:
B304         archive.  If the area is too small for the array, the area condition
B305         is signalled.  (Input)
B306
B307


A290         the archive.  It should be declared as follows:  (Output)
Changed by B to:
B311         the archive.  The archive_component_info structure is declared in
B312         the archive_component_info.incl.pl1 include file and described under
```

```
B313         archive_$get_component_info.  The array should be declared as
B314         follows:  (Output)


A293                  like archive_component_info based
Changed by B to:
B317                  like archive_component_info based


A301         is a standard system status code, one of the following;   (Output)
Changed by B to:
B325         is a standard system status code, one of the following:  (Output)


Inserted in B:
B328                  error_table_$unimplemented_versio
B329
B330
B331      :hcom:
B332      /****^  HISTORY COMMENTS:
B333        1) change(2026-09-23,Lippard), approve(2026-09-23,MCR10177):
B334           Correct next_component_info's arguments; explain error_table_$bad_arg
B335           from next_component(_info); list error_table_$unimplemented_version for
B336           list_components; document error_table_$no_component for over-long names
B337           and that the first of duplicate names is found. Fix typos.
B338                                                      END HISTORY COMMENTS */
B339
At end

Comparison finished: 19 differences, 116 lines.
```

Testing

A test kit, >udd>m>jjl>dev>MCR10177>ac_test>tests.archive, holds an exec_com for each bug with a visible symptom (bugNN.ec), the test archives they use (valid, and crafted with each format error), and archive_test.pl1, which calls the `archive_` entries directly. Each exec_com prints what should happen before and after the fix, then runs the commands. `bugtests.ec` runs those that need no setup; `bugtests_rqo.ec DIR` runs bugs 3 and 4, which need an empty directory with its own 60-record quota. All were run on MR12.8 in a new process with the installed bound_archive_, and again in a new process with the fixed version ahead of it in the search rules (`where archive` confirms which is in use).

Bug	MR12.8	Fixed
1	Query offers to delete the existing (write-only) archive	"No read permission on entry"; no query
2	`Person.Project.a` ACL entry deleted	Entry restored with its original `r`
3	`rw` entry for `Person.Project.a` left after a record quota overflow	Only the original entry remains

4	Second overflow: copy said to be in `[pd]>temp.archive`, which does not exist; next update truncates `archive_temp_.archive` to 0 records ("Format error")	Copy in `[pd]>!<unique>.archive` as reported; intact (73 records) after a further update
5	`xf` over a link: no message; link remains; target lost its extra name and ACL entry	"Name duplication. Unlinking the old link ..." as `copy -force`; target unchanged
7	`t` lists and `d` deletes the component with the 32-character prefix	"Entry name too long"; nothing deleted
8	"All components ... have been deleted"; the unextracted component is lost	Unextracted component kept in the archive
10	7 segments leaked in `[pd]` by a global `rd` of 110 components	None
11	Messages name the archive's directory	Messages name the working directory
12	"appended to" names the original archive	Names the copy in the working directory
20	Obsolete header listed without error	"Format error", nothing listed
21	Damaged fence listed without error	First component listed, then "Format error"
22	Non-numeric bit count listed as `x`	"Format error", nothing listed
23	Components listed, then "Format error"	"Format error", nothing listed
28	`get_component` and `get_component_info` find the 32-character prefix	"Component not found in archive"
33	"This operation is not allowed for a directory"	No message; the component is left unchanged
37	search_test program returns format error on component after first.	No format error.

In bug05.ec, `copy -force` and `ac x` (query answered yes) over a link behave the same before and after, and `ac xf` now matches them.

The destruction of the saved copy in bug 4 is shown by bug04d.ec, run in a new process: the copy (73 records) is truncated to 0 records, with its bit count unchanged, by an unrelated update, and then lists as a format error. (In the combined bugtests_rqo run on MR12.8, the second-overflow message was wrong as expected, but the truncation was not seen; bug04d.ec isolates it.)

An “ac tb **.archive” was run with the new bound_archive_ across every directory under >ldd and >sl1; no archive produced a format error. A few archives in >ldd>MR12.6>12.6.LDD and >ldd>system_library_auth_maint>log could not be read due to lack of access.

The other bugs have no reliable visible symptom and were found by inspection; the code they touch is on the paths exercised by the tests above, which pass. The exception is the copy-path overflow of bug 13, which needs an existing archive near the maximum segment length and was checked by inspection. The crafted test archives were generated outside Multics with Archive::Multics (tools/gen_test_archives.pl), which reads and writes the same format.

Documentation

archive.info and archive_.info are updated as described under Problem Description. No other documentation changes.

Version History

Date	Revision	Author	Comment
2026-09-25	0.1	Jim Lippard	Pre-release versions of document
2026-09-25	1.0	Jim Lippard	Version released for initial review/ comment.
2026-10-23	1.1	Jim Lippard	Updated descriptions for bugs 25 and 29 in light of MCRB comments. Added bug 37, including changes in behavior and test cases.
2026-11-03	1.2	Jim Lippard	Changes from audit pass one: component name arrays moved to a temporary segment (bug 10); archive_sort.pl1 and reorder_archive.pl1 added; archive_util_, archive_aux_, and update_handler_ changes restructured per auditor ask; info segments corrected for verify_info.