Subject: MCR10140, mbuild Enhancements for MR12.9

Author: Gary Dixon

Date: 2023-10-09

Multics Ticket: https://multics-trac.swenson.org/ticket/317
https://multics-trac.swenson.org/ticket/319

The ticket #317 requests enhancements to the Multics mbuild subsystem to support build and/or install of several programming language source segments and tool scripts. These include:

A. Pascal source segments (suffix: .pascal)

B. LISP Assembly Program segments (suffix: .lap)

C. compose Device Writer Tables (suffixes: .compdv, .comp_dsm and .comp_dsm.alm)

D. compose Device Writer Programs (suffix: .pl1.xdw)

E. compose Device Writer Library Expansion segments (suffix: .xdw)

F. MRDS Database Creation segments (suffix: .cmdb)

G. tss_basic_.archive

Support for building and/or installing all of these segments can be provided using the existing paradigms known to the mbuild system.

Segment types A through D must be compiled to translate their source into a derived object. Both source and object get installed into the Multics Libraries.

Segment type E describes a compose Library Expansion source that gets "called with arguments" by a compose Device Writer Program segment (type D above). These Library Expansion segments are merely installed for later reference by a compose Device Writer Program compilation.

Segment type F and G describe segment types that mbuild should install in the source directory of a library. The tss_basic_.archive should be installed only in >ldd>unbundled>source (as documented and currently installed).

Also, ticket #319 reports a minor mbuild defect which causes a spurious warning message from the mbuild analyze request. This defect will also be corrected.

Design Method

The majority of changes being proposed are enhancements which enable mbuild to build/install new types of segments. For each new segment type, the following steps are followed to design how mbuild can support that type of segment.

1. Get a list of all segments of that type that are already installed in the Multics Libraries (or other relevant library that has been organized to permit mbuild build and install operations).
2. Determine what suffixes identify these segment types, including all intermediate source suffix values, etc.
3. Determine which segment types would follow the **source** paradigm. This paradigm requires:
   - compilation of source segment to produce an object segment; followed by
   - possible archiving and binding of source and object to produce a bound segment; and
   - installation of source or source archive(s); object or object archive(s); and bound or unbound object segment.
4. For each source segment type:
   a. Identify how that segment type is compiled.
   b. Determine characteristics of that compiler and whether it meshes well with the requirements of the mbuild compile request.
   c. Requirements of mbuild compile request include the following.
      - Does it support control arguments for compilers:
        -list (-ls) -no_list (-nls)
        -table (-tb) -brief_table (-bftb) -no_table (-ntb)
        -optimize (-ot)
      - Does it accept an archive component pathname (which would permit compiling a source while it remains as a component in a source archive)?
        - If yes, then mbuild_compile_.pl1 must be modified to know about this capability (adding this compiler to an internal list of compilers supporting archive pathnames).
      - Does it make use of translator search paths to locate any include files referenced by the source; or some other search paths?
      - Does compilation produce an intermediate source file which must also be compiled? If so, does the compiler automatically invoke that second compile when appropriate?
5. For each segment type, layout a plan for integrating the type into the mbuild knowledge base (mbuild_info_.cds and mbuild_request_tables_.alm). The Pascal Source Segments section of this MCR represents this kind of plan.

Proposed Changes

The following changes are need for each of the ticket #317 enhancements (items A through G listed above).

Pascal Source Segments

The pascal compiler has the following compiler characteristics.

- Does support -list and -table control arguments; but uses -table as the default. [Most compilers that support -table do not generate symbol table information by default. For that reason, the mbuild compile request uses -no_table as its default.]
- Does not support -optimize.
- Does not accept an archive component pathname (which would permit compiling a source while it remains as a component in a source archive).
- Does use translator search paths to locate any include files referenced by the source.

To assign a build paradigm to pascal source segments, the following seg_info entry must be added to the mbuild_info_.cds segment. It informs mbuild's analyze request: to invoke the Multics pascal compiler for .pascal source segments; and to follow mbuild's **source** paradigm when archiving and installing pascal source and derived object segments.

The -list, -optimize and -table control arguments are the default compiler characteristics required by mbuild_compile_.pl1. -no_table is added as the default_compile_option to make pascal compilations match the compile request's -no_table default. mbuild_compile_.pl1 will be changed to remove -no_table if a "compile -table" request has been given.

```
B417                info.source_starname(i) = "**.pascal";       /* Pascal */
B418                info.description(i) = "Pascal Source File";
B419                info.mbuild_type(i) = "source";
B420                info.build_paradigm(i) = PDM_source;
B421                info.compiler(i) = "pascal";
B422                info.default_compile_options(i) = "-no_table";
B423                info.intermediate_suffix(i) = "";
B424                info.object_suffix(i) = "";
B425                info.default_library(i) = "source";
B426                info.type_ID(i) = i;
B427                i = i + 1;
```

With this entry added, the mbuild_type command then shows pascal segments being handled using the **source** paradigm.

```
mbuild_type seg typical.pascal

----------  typical.pascal
source_starname:              **.pascal
description:                  Pascal Source File
mbuild_type:                  source
default_library:              source
build_paradigm:               source
 I. For replace/add of source:
    1. Compile source.
    2. If source is part of bound_xxx_.**.s.archive:
       a. Update source in its .s.archive.
       b. Update compiled object in corresponding .archive.
       c. update_seg add/replace changed source archive in >ldd>LIBRARY>s.
       d. Schedule bind of object archives comprising bound_xxx_.
    3. Else for non-archived source:
       a. update_seg add/replace source in its >ldd>LIBRARY>s dir.
       b. update_seg add/replace derived object in >ldd>LIBRARY>o and >LIBRARY.
 II. For delete of source:
    1. If source is part of bound_xxx_.**.s.archive:
       a. Delete source from bound_xxx_.**.s.archive.
       b. Delete derived object from corresponding object archive.
       c. update_seg replace changed source archive in >ldd>LIBRARY>s.
       d. update_seg replace changed object archive in >ldd>LIBRARY>o.
       d. Schedule bind of changed object archive.
    2. Else for non-archived source:
       a. update_seg delete the source in >ldd>LIBRARY>s.
       b. update_seg delete derived object in >ldd>LIBRARY>o and >LIBRARY.
compiler:                     pascal
default_compile_options:      -no_table
object_suffix:
```

Note that an existing seg_info entry in mbuild_data_.cds properly identifies pascal include files as following the **Include** paradigm.

```
mbuild_type seg typical.incl.pl1

----------  typical.incl.pl1
source_starname:              **.incl.*
description:                  Include File
mbuild_type:                  Include
default_library:              include
build_paradigm:               target_only
 1. update_seg add/replace/delete of target in library dir chosen by target's suffix.
```

The mbuild_request_tables_.alm segment must add a new “pascal” request which invokes the Multics pascal command when the compile request processes a .pascal source segment.

```
B307         multics_request  pascal,
B308                          (pas),
B309                          (Compile Pascal source segments.),
B310                          ,
B311                          flags.allow_command+flags.dont_list+flags.dont_summarize
B312
```

LISP Assembly Program Segments

The LISP Assembly Program compiler (lap) has the following compiler characteristics.

- Does support -list.
- Does not support -optimize or -table.
- Does not accept an archive component pathname.
- Does not support use of include files, nor translator search paths (so far as I can tell).

The -list, -optimize and -table control arguments are the default compiler characteristics required by mbuild_compile_.pl1.

To assign a **source** build paradigm to LISP Assembly Programs, the following seg_info entry must be added to the mbuild_info_.cds segment.

```
Inserted in B:
B393                info.source_starname(i) = "**.lap";               /* LISP */
B394                info.description(i) = "LISP Assembler Source File";
B395                info.mbuild_type(i) = "source";
B396                info.build_paradigm(i) = PDM_source;
B397                info.compiler(i) = "lap";
B398                info.default_compile_options(i) = "";
B399                info.intermediate_suffix(i) = "";
B400                info.object_suffix(i) = "";
B401                info.default_library(i) = "source";
B402                info.type_ID(i) = i;
B403                i = i + 1;
```

With this entry added, mbuild recognizes .lap segments being handled using the **source** paradigm.

The mbuild_request_tables_.alm segment must add a new “lap” request which invokes the Multics lap command when the compile request processes a .lap source segment.

```
289     multics_request  lap,
290                      (),
291                      (Compile LISP assembler source segments.),
292                      ,
293                      flags.allow_command+flags.dont_list+flags.dont_summarize
```

Compose Device Support Modules

The WordPro System employs device-specific support modules to enable the full formatting capabilities of each class of output devices when displaying textual data. Each class of device has its own *device support module* or *dsm* to map generic compose formatting operations onto device-specific display commands.

Each device support module has two parts: a **compose Device Table** which describes device capabilities to compose; and a **compose Device Writer** program which maps display operations from compose onto device-specific commands to display data. Both parts are bound together as a device support module bound segment. For example, ascii.compdv and ascii_writer_.pl1.xdw are stored in bound_ascii_dsm_.s.archive; and their objects are bound into the bound_ascii_dsm_ bound segment.

The compdv compiler translates statements in a compose Device Table segment into an ALM source declaring binary data structures described by the Device Table statements. For example, an ascii.compdv Device Table source is translated to an ascii.comp_dsm.alm intermediate source segment. The compdv compiler then invokes the Multics alm command to translate that intermediate source into a data-containing object segment ascii.comp_dsm.

The ascii.compdv Device Table source is stored in the bound_ascii_dsm_.s.archive; the ascii.comp_dsm output by the ALM translation is stored in the bound_ascii_dsm_.archive. Like other compilers that generate an intermediate source, the ascii.comp_dsm.alm intermediate is discarded after the ALM translation completes.

The compose Device Writer programs use a Device Writer Source Expander tool which is a special adaptation of a general text string manipulation facility. This tool expands a fairly compact device writer input segment into a full writer source output segment. The pre-expansion input is a mixture of literal text and expansion constructs. The output contains only literal text: all expansion constructs have been replaced by their corresponding strings in the target language: PL/I for compose Device Writers.

The expand_device_writer (xdw) compiler expands statements in a compose Device Writer program source into a complete PL/I source whose statements form the Device Writer program. The ascii_writer_.pl1.xdw segment gets expanded into an intermediate PL/I source: ascii_writer_.pl1; this intermediate source is then compiled by the pl1 compiler into an ascii_writer_ object segment. The ascii_writer_.pl1.xdw segment is stored in the bound_ascii_dsm_.s.archive. The ascii_writer_ object is stored in the bound_ascii_dsm_.archive.

As described in Multics WordPro Reference Manual (order no. AZ98-02) Appendix C “Device Support Tools”, the Expander provides these features:

- Variables and arrays with three data storage classes
- Value assignment
- Expression evaluation
- Iteration
- Conditional execution
- Internal and external expansion calling (with arguments)
- Active function calling

The expand_device_writer (xdw) program provides a general-purpose toolset for embedding macro-like program constructs within any source language segment. A compose Device Writer program does not use its full capabilities. For example, the xdw command supports many control arguments, none of which are needed to expand compose Device Writer programs.

```
1984-07-12  expand_device_writer, xdw

Syntax as a command:  xdw {path} {-control_args}

Arguments:
 path

Control arguments:
 -call command_line    -input_string string,         -no_print, -npr
 -brief, -bf             -instr string               -print, -pr
 -long, -lg            -output_file path, -of path   -arguments ..., -ag ...
```

The above syntax does not meet the requirements imposed by the mbuild compile request. Attempts to change expand_device_writer.pl1 were stymied by code which processes control arguments as they are found in the command line, but then fails to setup proper cleanup on-units to undo this processing if an error is found with later processing. Also, code complexity would be increased by trying to warp handling of all of the above arguments to meet requirements suitable to the mbuild compile request.

This MCR proposes to create a new mbuild xdw request that accepts the name of the Device Writer program plus pl1 control arguments supported by the mbuild compile request: -list -table -optimize.

The next two subsections discuss changes needed in mbuild to support each of these compose DSM-related compilers.

Compose Device Table Segments

The compdv compiler has the following compiler characteristics.

- Does support -list.
- Does not support -optimize or -table.
- Does not accept an archive component pathname.
- Compiles XXX.compdv to create XXX.comp_dsm.alm, then calls alm to translate this intermediate source into an object data segment: XXX.comp_dsm

To assign a **source** build paradigm to compose Device Table segments, the following seg_info entry must be added to the mbuild_info_.cds segment.

```
B453                info.source_starname(i) = "**.compdv";
                                    /* COMPDV (compose Device Table compiler) */
B454                info.description(i) = "compose Device Table Source File";
B455                info.mbuild_type(i) = "source";
B456                info.build_paradigm(i) = PDM_source;
B457                info.compiler(i) = "compdv";
B458                info.default_compile_options(i) = "";
B459                info.intermediate_suffix(i) = ".comp_dsm.alm";
B460                info.object_suffix(i) = ".comp_dsm";
B461                info.default_library(i) = "source";
B462                info.type_ID(i) = i;
B463                i = i + 1;
```

To assign the **Unbound_obj** build paradigm to the final output generated by the compdv compiler (the XXX.comp_dsm data object), the following seg_info entry must be added to mbuild_info_.cds.

```
B297                info.source_starname(i) = "*.comp_dsm";
                                   /* Compose Device Table Object Segment */
B298                info.description(i) = "compose Device Data Object";
B299                info.mbuild_type(i) = "Unbound_obj";
B300                info.build_paradigm(i) = PDM_Unbound_obj;
B301                info.compiler(i) = "";
B302                info.default_compile_options(i) = "";
B303                info.intermediate_suffix(i) = "";
B304                info.object_suffix(i) = "";
B305                info.default_library(i) = "execution";
B306                info.type_ID(i) = i;
B307                i = i + 1;
```

With this entry added, the mbuild_type command then shows .comp_dsm object segments being handled using the **Unbound_obj** paradigm.

```
mbuild_type seg typical.comp_dsm

----------  typical.comp_dsm
source_starname:              *.comp_dsm
description:                  Standalone Segment
mbuild_type:                  Unbound_obj
default_library:              execution
build_paradigm:               Unbound_obj
 1. If object is part of an object archive, bound_xxx_.**.archive:
    a. Object would be built as part of compile of its source file.
 2. Else if non-archived object:
    a. update_seg add/replace/delete target in >ldd>LIBRARY>o and >LIBRARY.
```

The mbuild_request_tables_.alm segment must add a new “compdv” request which invokes the Multics compdv command when the compile request processes a .compdv source segment.

```
B277         multics_request  compdv,
B278                          (),
B279                          (Compile compose Device Table files.),
B280                          ,
B281                          flags.allow_command+flags.dont_list+flags.dont_summarize
```

Three additional changes are required to support compose Device Table compilation.

First, the mbuild_info_.incl.pl1 field which holds the intermediate suffix for a Device Table must be lengthened from 12 to 16 characters to hold the 13-character suffix: .comp_dsm.alm

```
cpa [lpn mbuild_info_.incl.pl1] ==

A40                  4 intermediate_suffix      char (12) var,
                      /* suffix of intermediate file generated by compiler      */
Changed by B to:
B45                  4 intermediate_suffix      char (16) var,
                      /* suffix of intermediate file generated by compiler      */
```

Second, all of the mbuild source segments that reference this include file must be recompiled so their object segment accommodates the larger size of this structure element. The following files must be recompiled using their already-installed source. Other files mentioned earlier may also include mbuild_info.incl.pl1; but they will be recompiled anyway to make other changes.

```
mbuild_lpn.pl1,           mbuild_type.pl1,
mbuild_archive_.pl1,      mbuild_compare_.pl1,      mbuild_data_.pl1,
mbuild_display_.pl1,      mbuild_history_.pl1,      mbuild_info_checks_.pl1,
mbuild_info_find_.pl1,    mbuild_install_.pl1,      mbuild_print_.pl1,
mbuild_scan_.pl1,         mbuild_script_info_.rd,   mbuild_script_parse_.rd,
mbuild_xref_.pl1
```

Third, the compdv.rd compiler violates the rules for naming its output object segment. Compiling x9700_doc.compdv should produce an object segment whose primary name is derived from the input segment's name. In this case, x9700_doc.compdv should produce an object named x9700_doc.comp_dsm. However, the current compiler bases its output object names on the order in which devices were referenced in the input file. The current command operates as shown below. Notice that the name x9700_doc.comp_dsm appears third on the object segment.

```
compdv x9700_doc
COMPDV 2.0a-5
ALM 8.14
r 13:01 30.945 367

ls x9700*.*

Segments = 2, Lengths = 30.

r w    9  x9700_doc.compdv
re    21  x9700_pr.comp_dsm
          roman.comp_dsm
          x9700_doc.comp_dsm
```

This MCR proposes changing compdv to generate an output object whose primary name is based upon the argument name given to the compdv command. That is a requirement of the mbuild compile request. After this change, the x9700_doc.comp_dsm name appears first on the object segment.

```
compdv x9700_doc
COMPDV 2.0a-5
ALM 8.14
r 13:09 30.769 366

ls x9700*.**

Segments = 2, Lengths = 30.

re    21  x9700_doc.comp_dsm
          x9700_pr.comp_dsm
          roman.comp_dsm
r w    9  x9700_doc.compdv
```

The proposed change to compdv.rd is shown below.

```
cpa [lpn compdv.rd] ==

A1806             ename = rtrim (dvid.refname) || ".comp_dsm";
Changed by B to:
B1816             ename = rtrim (ename) || ".comp_dsm";
```

Compose Device Writer Programs

The proposed new mbuild xdw request has the following syntax:

```
2023-10-01  xdw

Syntax:  xdw INPUT_EXPANSION_SEG {-pl1_control_args}

Arguments:
 INPUT_EXPANSION_SEG
```

It will meet all requirements of the mbuild compile request.

- Does support -list, -optimize and -table .
- Does accept an archive component pathname.
- Expands XXX.pl1.xdw source to create an XXX.pl1 intermediate source, then calls pl1 to translate this intermediate source into XXX object segment.

To perform the actual expansion of the .pl1.xdw source, the new xdw request will call the same subroutines used by the expand_device_writer command: xdw_$expand and xdw_$free. If the expansion succeeds, the new xdw request will store the expanded result in an intermediate .pl1 source segment, then call pl1 to translate that source. Any compile request -control_args or default compile options it receives will be passed to pl1. The object segment produced by pl1 will be treated as the output from the xdw request to be incorporated into the compose DSM bound object.

Currently, the ascii_writer_.pl1 intermediate source is also being stored in the bound_ascii_dsm_.s.archive. But such storage does not adhere to the install paradigm used for intermediate source files. This MCR proposes to adhere to the existing paradigm by discarding the ascii_writer_.pl1 intermediate source once the PL/I compiler has translated it.

To assign a **source** build paradigm to compose Device Writer programs, the following seg_info entry must be added to the mbuild_info_.cds segment.

```
B501                info.source_starname(i) = "*.pl1.xdw";
                        /* compose Device Writer input expansion                */
B502                info.description(i) = "Device Writer input expansion";
B503                info.mbuild_type(i) = "source";
B504                info.build_paradigm(i) = PDM_source;
B505                info.compiler(i) = "xdw";
B506                info.default_compile_options(i) = "-ot";
B507                info.intermediate_suffix(i) = ".pl1";
B508                info.object_suffix(i) = "";
B509                info.default_library(i) = "source";
B510                info.type_ID(i) = i;
B511                i = i + 1;
```

The new xdw request requires four additional changes.

First, the bound_compose_tools_.bind file does retain the xdw_$expand and xdw_$free entry points, but does not add the name xdw_ to the bound object segment. Updating the .bind file to add this name will permit the new mbuild_xdw_.pl1 to reference these two xdw_ subroutine entry points.

```
cpa >ldd>unb>object>bound_compose_tools_.archive::bound_compose_tools_.bind ==


A10                             list_comp_dsm, lcdsm;
Changed by B to:
B17                             list_comp_dsm, lcdsm,
B18                             xdw_;
B19
```

Second, the existing compose Device Writer programs (e.g., ascii_writer_.pl1.xdw) make reference to the comp_dev_writer.xdw Library Expansion segment. This segment is currently installed at:
`>ldd>unbundled>source>comp_dev_writer.xdw`

The xdw_$expand subroutine searches for that segment using the “xdw” search paths. Currently, these search paths default as shown below.

```
psp xdw
xdw
          -referencing_dir
          -working_dir
```

Compilation of an existing Input Expansion segment will fail because these default search paths do not include the Multics Libraries directory in which the Library Expansion segments are typically stored.

This MCR proposes to change the xdw_defaults_.cds segment to add the missing library pathname.

```
cpa [lpn xdw_defaults_.cds] ==

A28               3 paths     (2) like search_path;
Changed by B to:
B36               3 paths     (3) like search_path;

Inserted in B:
B64             lists.xdw.paths (3).type = ABSOLUTE_PATH;
B65             lists.xdw.paths (3).pathname = “>ldd>unbundled>source”;
Preceding:
A56
A57             unspec (cdsa) = “”b;
```

After this change, the default search paths for “xdw” include the new directory.

```
psp xdw
xdw
          -referencing_dir
          -working_dir
          >ldd>unbundled>source
```

Third, to permit this (or other) Library Expansion segment to be updated or added, assign a **target_only** build paradigm to segments with one name followed by the .xdw suffix. The following seg_info entry must be added to the mbuild_info_.cds segment.

```
B513                info.source_starname(i) = "*.xdw";
                        /* compose Device Writer library expansion               */
B514                info.description(i) = "Device Writer library expansion";
B515                info.mbuild_type(i) = "expansion";
B516                info.build_paradigm(i) = PDM_target_only;
B517                info.compiler(i) = "";
B518                info.default_compile_options(i) = "";
B519                info.intermediate_suffix(i) = "";
B520                info.object_suffix(i) = "";
B521                info.default_library(i) = "source";
B522                info.type_ID(i) = i;
B523                i = i + 1;
```

When the entry above is added, the mbuild_type command shows .xdw Library Expansion segments being handled using the **target_only** paradigm.

```
mbuild_type seg comp_dev_writer.xdw

----------  comp_dev_writer.xdw
source_starname:              *.xdw
description:                  Device Writer library expansion
mbuild_type:                  expansion
default_library:              source
build_paradigm:               target_only
 1. update_seg add/replace/delete of target in library dir chosen by target's suffix.
```

Fourth, update the bound_mbuild_.bind segment to include mbuild_xdw_ in the Order statement, and in a new objectname statement.

```
compare_ascii >ldd>tools>object>bound_mbuild_.archive::bound_mbuild_.bind ==

Inserted in B:
B64                         mbuild_xdw_,
Preceding:
A61                         mbuild_xref_,


A150      objectname:       mbuild_xref_;
Changed by B to:
B154      objectname:       mbuild_xdw_;
B155
B156      objectname:       mbuild_xref_;
```

MRDS Create Database Segments

To install scripts for creating a MRDS data base (suffix: .cmdb), assign a **target_only** paradigm to put these segments in the source directory of a library.

```
B441                info.source_starname(i) = "**.cmdb";
                          /* MRDS Database Creation Segment                    */
B442                info.description(i) = "Create MRDS DB Segment";
                          /*  build script statement:                          */
B443                info.mbuild_type(i) = "cmdb";
                          /*   Seg(cmdb): xxx.cmdb  IN: sss.source  REPLACE;   */
B444                info.build_paradigm(i) = PDM_target_only;
B445                info.compiler(i) = "";
B446                info.default_compile_options(i) = "";
B447                info.intermediate_suffix(i) = "";
B448                info.object_suffix(i) = "";
B449                info.default_library(i) = "source";
B450                info.type_ID(i) = i;
B451                i = i + 1;
```

TSS Basic Source Program Archive

To make changes to the TSS basic program sample archive (entry: tss_basic_.archive), assign a **target_only** paradigm to put this segment in the unbundled.source directory.

```
B658                info.source_starname(i) = "tss_basic_.archive";
                    /* TSS BASIC sample programs ARCHIVE                       */
B659                info.description(i) =
                                "Archive for TSS basic_system Sample Programs";
B660                /*  build script statement:                               */
B661                info.mbuild_type(i) = "tss_basic_arch";
                    /* Seg(tss_basic_arch): tss_basic_.archive IN: unb.s REPLACE; */
B662                info.build_paradigm(i) = PDM_target_only;
B663                info.compiler(i) = "";
B664                info.default_compile_options(i) = "";
B665                info.intermediate_suffix(i) = "";
B666                info.object_suffix(i) = "";
B667                info.default_library(i) = "unbundled.source";
B668                info.type_ID(i) = i;
B669                i = i + 1;
```

Changes to mbuild_compile_.pl1 (compile request)

Minor changes are needed in the mbuild compile request as follows.

A. Inform the compile request that pascal and xdw support use of the -table control argument.

B. Change the compile request to support additional control arguments:
-no_list -no_table -brief_table

C. Inform the compile request that the new xdw request supports archive component pathname arguments.

The following shows code changes needed to accomplish these changes in mbuild_compile_.pl1.

```
A56         dcl (addr, length, maxlength, null, reverse, string) builtin;
Changed by B to:
B60         dcl (addr, decat, length, maxlength, null, reverse, string) builtin;


Inserted in B:
B168            2 tableN fixed bin(2),
                          /* toggle between -table, -brief_table, -no_table         */
Preceding:
A164            2 S,


A166                4 (listS,
A167                   tableS
A168                   ) bit(1) unaligned;
Changed by B to:
B171                4 (listS
                          /* -list or -no_list                                      */
B172                   ) bit(1) unaligned;
B173        dcl (NO_TABLE init(1),
B174             BRIEF_TABLE init(2),
B175             TABLE init(3)
B176             ) fixed bin(2) int static options(constant);
B177

Inserted in B:
B181           C.tableN = NO_TABLE;
                          /* Set default control args: -no_list -no_table           */
Preceding:
A172           C.S = F;

Inserted in B:
B297
B298          NOTES:  The "xdw" request referenced below as a compiler is the
B299                  mbuild_xdw_$xdw_request entry point of mbuild (which is a
B300                  subset of functionality in expand_device_writer (xdw)
B301                  Multics command).
Preceding:
A287         ------------------------------------------------------------------- */
```

```
A323                if segt.compiler = "pl1" | segt.compiler = "alm" then
A324                     source_compiles_in_archiveS = T;
Changed by B to:
B338                if segt.compiler = "pl1" | segt.compiler = "alm"
                     | segt.compiler = "xdw" then
B339                     source_compiles_in_archiveS = T;

A382        dcl  source_tableS bit(1) aligned;
A383
A384           source_tableS = F;
A385           if      source.compiler = "pl1"  then
A386                source_tableS = c.tableS;
A387           else if source.compiler = "rdc"  | source.compiler = "reductions" then
A388                source_tableS = c.tableS;
A389
A390           call mbuild_$request (c.sciP,
                    "^2/^a ^[^a::^;^s^]^a ^a^[ -table^;^]^[ -list^;^]", T,
A391                source.compiler, source_compiles_in_archiveS, source.archive_name,
A392                source.name, source.compile_options, source_tableS, c.listS);
Changed by B to:
B397        dcl  source_tableN fixed bin(2);
B398        dcl  source_compile_opts char(32) var;
B399
B400           source_compile_opts = source.compile_options;
                          /* pl1, pascal, reductions, xdw support use of -table     */
B401           source_tableN = NO_TABLE;
                          /*  and -brief_table.                                     */
B402           if      source.compiler = "pl1"  then
B403                source_tableN = c.tableN;
B404           else if source.compiler = "rdc"  | source.compiler = "reductions" then
B405                source_tableN = c.tableN;
B406           else if source.compiler = "xdw"  then
B407                source_tableN = c.tableN;
B408           else if source.compiler = "pas"  | source.compiler = "pascal" then
B409                source_tableN = c.tableN;
                          /* pascal defaults to -table rather than -no_table, but   */
B410                      /*  its mbuild_info_.cds entry for *.pascal specifies     */
B411                      /*  -no_table as default source_compile_opts              */
B412
B413           if source_tableN ^= NO_TABLE then do;
                          /* If compile request includes -table or -brief_table,    */
B414                source_compile_opts = decat(source_compile_opts, "-no_table",
                                            "101"b);
B415                source_compile_opts = decat(source_compile_opts, "-ntb",
                                            "101"b);
B416                end;  /*   remove any -no_table from source_compile_opts.       */
B417
B418           call mbuild_$request (c.sciP,
                    "^2/^a ^[^a::^;^s^]^a ^a^[^; -brief_table^; -table^]^[ -list^;^]",
                    T,
B419               source.compiler, source_compiles_in_archiveS, source.archive_name,
B420               source.name, source_compile_opts, source_tableN, c.listS);
```

```
A681                    if       arg = "-ls"
                         |       arg = "-list"          then c.listS = T;
A682                    else if  arg = "-tb"
                               |  arg = "-table"         then c.tableS = T;
Changed by B to:
B709                    if       arg = "-ls"
                         |       arg = "-list"          then c.listS = T;
B710                    else if  arg = "-nls"
                               |  arg = "-no_list"       then c.listS = F;
B711
B712                    else if  arg = "-tb"
                               |  arg = "-table"         then c.tableN = TABLE;
B713                    else if  arg = "-bftb"
                               |  arg = "-brief_table"   then c.tableN = BRIEF_TABLE;
B714                    else if  arg = "-ntb"
                               |  arg = "-no_table"      then c.tableN = NO_TABLE;

Comparison finished: 9 differences, 70 lines.
```

Minor Build Defect

To correct the minor build defect in ticket #319, mbuild_analyze_ needs to pass a hint to mbuild_data_$get_UNBOUNDOBJ that the installation paradigm calls for installing the object segment described by this Unbound_obj data structure in the “object” directory of its library. The following change to mbuild_analyze_.pl1 passes this hint:

```
A497     revised_library = before(Seg.library,".");
A498     UNBOUNDOBJp =
A499          mbuild_data_$get_UNBOUNDOBJ (addr(bld), revised_name, revised_library,
                                          "REPLACE", addr(Seg));
Changed by B to:
B501     revised_library = mbuild_library_$replace_library_component( Seg.library,
                                          "", "object");
B502     UNBOUNDOBJp =
B503          mbuild_data_$get_UNBOUNDOBJ (addr(bld), revised_name, revised_library,
                                          "REPLACE", addr(Seg));
```

Minor Problems Found/Fixed during Testing

While testing the changes proposed in this MCR, a problem was discovered in mbuild_install_.pl1. That program loops through the list of segments eligible for installation in the target libraries, comparing the just-built segment with its library copy. If the two segments are identical, then the update_seg replace task for that segment is omitted.

However, testing of the above changes was usually done by recompiling existing library sources rather than making changes to them. Testing focused on whether recompiled objects (and intermediate sources) matched their library instances. In such case, no source archive is changed, and therefore no source segments should be replaced in the library. But mbuild_install_ was checking only the first source segment with its library instance. It found they were identical and reported that the install would be skipped. But it stopped checking other sources eligible for installation. That stopping is the bug being fixed.

The bug occurred because when a segment eligible for install was found to be identical with its library instance, the Seg structure describing that segment was removed from the list. But the Tlist_remove operation thwarted further checks in the list. The Seg structure held a pointer to the next item in the list. But once that Seg structure was removed from the list, that next pointer gets set to null() by the Tlist_remove operation. So scanning of the list stops prematurely.

The repair is to obtain a pointer to the next list item before checking the Seg structure for identical content with its library instance. All instances of using the Tlist_remove subroutine within mbuild software were checked for presence of this bug. mbuild_scan_.pl1 had the same problem. The repair will be applied in both source programs.

```
compare_ascii >ldd>tools>source>bound_mbuild_.s.archive::mbuild_install_.pl1 ==

Inserted in B:
B477        dcl  Seg_nextP ptr;
                        /* Pointer to next Seg structure in the request list.     */
Preceding:
A473        dcl  starname char(32) var;


A476       do SegP = Tlist_first_item (addr(Areq.request_Tb))
A477          repeat Tlist_next_item  (addr(Seg.request_Td))  while (SegP ^= null() );
A478
Changed by B to:
B481       do SegP = Tlist_first_item (addr(Areq.request_Tb))
B482          repeat Seg_nextP   while (SegP ^= null() );
B483
B484            Seg_nextP = Tlist_next_item  (addr(Seg.request_Td));
B485                      /* Get pointer to next Seg structure in the request list  */
B486                      /*  in case current Seg structure gets removed from list. */
B487
```

```
compare_ascii >ldd>tools>source>bound_mbuild_.s.archive::mbuild_scan_.pl1 ==

Inserted in B:
B234        dcl  Seg_nextP ptr;
                        /* Pointer to next Seg structure in the request list.     */
Preceding:
A230
A231          do SegP = Tlist_first_item (addr(bld.scan_Tb))


A231          do SegP = Tlist_first_item (addr(bld.scan_Tb))
A232             repeat Tlist_next_item  (addr(Seg.scan_Td))  while (SegP ^= null() );
Changed by B to:
B236          do SegP = Tlist_first_item (addr(bld.scan_Tb))
B237             repeat Seg_nextP  while (SegP ^= null() );
B238               Seg_nextP = Tlist_next_item  (addr(Seg.scan_Td));
                        /* Get pointer to next Seg structure in the request list  */
B239                    /*  in case current Seg structure gets removed from list. */
B240
```

Additional MCR Needed

If this MCR proposal is approved, then a repair MCR will be needed to correct two current installation issues.

A. The bound_XXX_dsm_.s.archive segments currently have both the original source (e.g., ascii_writer_.pl1.xdw) and the intermediate source file (e.g., ascii_writer_.pl1) installed. This intermediate source must be removed from each of the source archives to match the requirements of the **source** installation paradigm. The three bound archives involved are:

```
lpn -lb ** bound_*_dsm_.s.archive
>ldd>unb>source>bound_xerox_dsm_.s.archive
>ldd>unb>source>bound_hyterm_dsm_.s.archive
>ldd>unb>source>bound_ascii_dsm_.s.archive
```

B. The compose Device Writer Library Expansion segment, comp_dev_writer.xdw, is currently installed in two places:

```
lpn -lb ** comp_dev_writer.xdw
>ldd>unb>source>comp_dev_writer.xdw
>unb>comp_dev_writer.xdw
```

The **target_only** paradigm proposed for this segment calls for installation only in the source directory. So the copy in >unbundled directory should be deleted.

Build Script Description of the Changes

```
Description --
 1) Change mbuild_analyze_.pl1 to give proper "object" directory
    hint when replacing a standalone Unbound_obj such as a library
    descriptor (.ld) segment.  This avoids spurious warning message.
 2) Change mbuild_compile_.pl1 to:
     A) Support: -no_list, -no_table, and -brief_table control args.
     B) Support: new xdw request
 3) Change mbuild_info_.cds to:
     A) Support compiling **.pascal source segments.
     B) Support compiling **.lap LISP Assembler source segments.
     C) Support compiling **.compdv compose Device Writer tables.
     D) Support compiling *.pl1.xdw compose Device Writer programs.
     E) Support installing  *.xdw Library Expansion segments.
     F) Support installing **.cmdb Create MRDS DB segments.
     G) Properly install tss_basic_.archive ONLY in >ldd>unb>s
        directory.  That is how it is installed now (only in the
        unb.source directory), and where tss_basic.gi.info documents
        its install location.
 4) Change mbuild_request_tables_.alm to add Multics commands or
    mbuild request to invoke new compilers:
     A) pascal  - Multics command
     B) lap     - Multics command
     C) compdv  - Multics command
     D) xdw     - mbuild_xdw_$xdw_request
 5) Change mbuild_info_.incl.pl1 as follows:
     A) Comment which describes the PDM_source_x_only paradigm is
        marked as "NOT SUPPORTED BY mbuild".  Nothing in current
        libraries uses this paradigm.
     B) Change length of mbuild_info.seg_type_info.intermediate_suffix from
        char(12) var to char(16) var to support suffix:  .comp_dsm.alm
 6) Recompile the following source segments which are otherwise unchanged
    because of their reference to the mbuild_info.seg_type_info structure.
     mbuild_lpn.pl1, mbuild_type.pl1,
     mbuild_archive_.pl1,     mbuild_compare_.pl1,
     mbuild_data_.pl1,        mbuild_display_.pl1,
     mbuild_history_.pl1,     mbuild_info_checks_.pl1,
     mbuild_info_find_.pl1,   mbuild_install_.pl1,          mbuild_print_.pl1,
     mbuild_scan_.pl1,        mbuild_script_info_.rd,       mbuild_script_parse_.rd,
     mbuild_xref_.pl1
 7) Change compdv.rd to use name of its input argument when generating
    primary name for its resulting object segment.  Thus, x9700_doc.compdv
    always generates an output file whose primary name is x9700_doc.comp_dsm.
    This permits mbuild's source paradigm to properly build/install this
    source and its object segment.
 8) Change mbuild_install_.pl1 and mbuild_scan_.pl1 SegP =/repeat/while loop
    to pre-compute Seg_nextP in case loop calls Tlist_remove on current SegP
    value.  Otherwise, loop ends when SegP gets removed; its threads are
    no longer visible to Tlist_remove subroutine.
 9) Add mbuild_xdw_.pl1 which will include a new xdw_request entry point
    that will implement the feature subset of the Multics expand_device_writer
    (xdw) command needed to compile existing compose Device Writer programs
    (XXX.pl1.xdw).  Features of the mbuild_xdw_$xdw_request are:
     A) Automatically invoke the pl1 command if expand operation succeeds.
     B) Accept control arguments supported by pl1 command, and pass them
        thru to the pl1 compiler when it is automatically invoked.
```

```
10) Create a separate MCR to:
     A) Change content of bound_XXX_dsm_.s.archives to include only the
        XXX_writer_.pl1.xdw segment, rather than that seg plus its derived
        XXX_writer_.pl1 segment (which is the intermediate source created by
        the xdw command before it then invokes the pl1 compiler to create the
        XXX_writer_ object segment).  This MCR will propose re-installing all three
        bound_XXX_dsm_ bound objects to implement this revised install paradigm.
     B) delete >unb>comp_dev_writer.xdw leaving only the copy in >ldd>unb>s
        directory.
11) Change xdw_defaults_.cds to add a 3rd directory to the xdw search paths:
      -referencing_dir
      -working_dir
      >ldd>unbundled>source
    The >ldd>unbundled>source directory stores installed device writer macros like
    comp_dev_writer.xdw.  This file is included by all three existing
    device writer (.pl1.xdw) source programs.
12) Change mbuild.info to include new :[Info]: block describing xdw request.
    The following areas need to be documented in the new info block.
     A) USE OF:  xdw search paths
     B) EXAMPLES OF:  -call COMMAND_LINE
     C) EXAMPLES OF:  expanding actual device writer file
     D) NO SUPPORT FOR:  archive component pathnames
     E) REFERENCE TO: documentation of expansion language
13) Change bound_compose_tools_.bind to add the name: xdw_ in order to
    make the already-retained xdw_$expand and xdw_$free entry points
    callable.


Changes --


 Bound_obj:    bound_compose_tools_              IN: unb  UPDATE;
   bindfile:   bound_compose_tools_.bind          REPLACE;
   source:     compdv.rd                          REPLACE  compiler: rdc
                                                           -ot -trace off;

 Bound_obj:    bound_mbuild_                     IN: tools  UPDATE;
   bindfile:   bound_mbuild_.bind                 REPLACE;
   source:     mbuild_analyze_.pl1                REPLACE  compiler: pl1 -ot;
   source:     mbuild_archive_.pl1                REPLACE  compiler: pl1 -ot;
   source:     mbuild_compare_.pl1                REPLACE  compiler: pl1 -ot;
   source:     mbuild_compile_.pl1                REPLACE  compiler: pl1 -ot;
   source:     mbuild_data_.pl1                   REPLACE  compiler: pl1 -ot;
   source:     mbuild_display_.pl1                REPLACE  compiler: pl1 -ot;
   source:     mbuild_history_.pl1                REPLACE  compiler: pl1 -ot;
   source:     mbuild_info_.cds                   REPLACE  compiler: cds;
   source:     mbuild_info_checks_.pl1            REPLACE  compiler: pl1 -ot;
   source:     mbuild_info_find_.pl1              REPLACE  compiler: pl1 -ot;
   source:     mbuild_install_.pl1                REPLACE  compiler: pl1 -ot;
   source:     mbuild_lpn.pl1                     REPLACE  compiler: pl1 -ot;
   source:     mbuild_print_.pl1                  REPLACE  compiler: pl1 -ot;
   source:     mbuild_request_tables_.alm         REPLACE  compiler: alm;
   source:     mbuild_scan_.pl1                   REPLACE  compiler: pl1 -ot;
   source:     mbuild_script_info_.rd             REPLACE  compiler: rdc
                                                           -ot -trace off;
   source:     mbuild_script_parse_.rd            REPLACE  compiler: rdc
                                                           -ot -trace off;
   source:     mbuild_type.pl1                    REPLACE  compiler: pl1 -ot;
   source:     mbuild_xdw_.pl1                    ADD      compiler: pl1 -ot;
   source:     mbuild_xref_.pl1                   REPLACE  compiler: pl1 -ot;
```

```
Unbound_obj:  xdw_defaults_                    IN: unb  REPLACE;
                add_name:  xdw.search;
  source:     xdw_defaults_.cds                 REPLACE  compiler: cds;

Include:      mbuild_info_.incl.pl1            IN: sss.include  REPLACE;

Info:         mbuild.info                      IN: priv.info  REPLACE;
               add_name:
                mb.info
                build_script.gi.info
                build_script.info;
```

Testing

As each bug repair was made, the mbuild subsystem was then rebuilt and run against the test case scenario in which the bug was encountered. Additional tests were made to ensure the repair did not adversely impact other aspects of mbuild operation.

As each enhancement to support a new segment type was made, an updated version of bound_mbuild_ was generated including the proposed changes supporting that segment type. This version of mbuild was then used to recompile, archive/bind, and create an install script to handle all (or sometimes many) of the existing segments of that type now in the Multics Libraries. Contents of any intermediate source segment was compared with equivalent source generated by hand-compile of the original source segment.

Length, retained entry points, names added, etc. of object segments were compared with equivalent segments now in the libraries. Then an installation script was created and examined to ensure that all/only the correct segments were being installed at the proper location in the library, with proper attributes.

The few problems found during testing were identified and corrected: in the mbuild code; or in the source compilers; or their search path mechanisms, etc. Those corrections: were identified in the proposed changes above; and are included as part of this MCR.

Documentation

*Documentation for the changed mbuild **compile** request:*

```
>udd>Multics>GDixon>work>mbt_test>mbuild.info  [Info]: compile   (34 lines in info)

2023-10-08  compile, comp

Syntax:  compile {SEG_NAME} {-control_args}
         comp    {SEG_NAME} {-control_args}

Function: compiles source segments to create their derived Unbound_obj
segments.

Arguments:
SEG_NAME
   names a single segment to compile.  The name includes the language
   suffix (e.g., source.pl1).  If not given, all segments in the
   COMPILE list are compiled.  Useful to recompile a single source
   after correcting a compilation error.  The star convention is
   supported.

Control arguments:
   If given, the following control arguments are added to any compile
   option given in the source statement for each segment.
-brief_table, -bftb
   generates a partial symbol table that consists of only the
   information necessary to set breakpoints by source line number.
-no_table, -ntb
   don't generate a runtime symbol table.  (Default)
-table, -tb
   produces an Unbound_obj output containing a full symbol table for
   use by probe and debug commands.

-list, -ls
   produces a source program listing followed by a list of all the
   names used in the compilation, followed by an assembly-like listing
   of the compiled object program.
-no_list, -nls
   don't produce a source program listing.  (Default)
```

*Documentation for the new mbuild **xdw** request:*

```
>udd>Multics>GDixon>work>mbt_test>mbuild.info  [Info]: xdw   (56 lines in info)

2023-10-01  xdw

Syntax:  xdw INPUT_EXPANSION_SEG {-pl1_control_args}

Function:  expands a compose Device Writer source code using a library
of additional expansions supplied in the comp_dev_writer.xds segment.

The expanded pl1 source segment is then translated by the pl1 compiler
into an object segment that can be bound into a compose Device Service
Module (e.g., bound_ascii_dsm_).

The xdw request is normally invoked by using the compile request.  For
example:  compile ascii_writer_.pl1.xdw

Arguments:
INPUT_EXPANSION_SEG
   is the entryname of the expansion input segment.  The entryname of
   this segment must have the suffix pl1.xdw, but the suffix need not
   be given in the command line.  The expanded output file is written
   to a segment in the build directory with .xdw removed from the input
   segment's suffix.   Multi-segment files and the star convention are
   not supported; archive component pathnames are supported.

Control arguments:
   Any control arguments supported by the pl1 command may be given
   to xdw.  These will be validated by the pl1 compiler when it
   translates the expanded input segment.

Notes on the xdw search paths:
When searching for library expansions (like comp_dev_writer.xdw), the
xdw search paths are used to find that segment.  The default values are
established when displaying these search paths: psp xdw;
or when using the search paths to locate a library expansion segment:
  wsp xdw EXPANSION_NAME.xdw

Notes on compiling expanded inputs:
If the INPUT_EXPANSION_SEG is successfully expanded, the expanded
output is written to a segment in the working directory with the .xdw
portion of the suffix removed.  For example, output from expanding the
input segment ascii_writer_.pl1.xdw would be written to the
ascii_writer_.pl1 segment.  The PL/I compiler is then invoked to
translate this pl1 source segment into an object segment ascii_writer_.
Any -pl1_control_args given with the xdw request are passed to the pl1
compiler for this translation.

Notes on the expansion language:
The Multics Wordpro Reference Manual (order no. AZ98-02) Appendix C
describes the Device Writer Source Expander language used to build an
INPUT_EXPANSION_SEG.

While input expansion segments are treated like normal source programs
to be installed in the source archive of a bound_XXX_dsm_ device
support module object segment, library expansion segments are installed
as standalone segments in the >ldd>unbundled>source directory.  These
library segments are not compiled directly, but are included by an
input expansion segment and compiled with that input segment.
```

Version

Date	Revision	Author	Comment
2023-10-08	1.0	Gary Dixon	Initial review/comment version of the MCR.
2023-10-09	1.1	Gary Dixon	Fix minor typo issues found proof-reading the MCR.
2023-10-10	1.2	Gary Dixon	Fix more typos.