Subject: MCR10155, Changes for call, define_area_, check_equal_name_ and lex_string_

Author: Gary Dixon

Date: 2025-07-22

Multics Ticket: http://multics-trac.swenson.org/ticket/333
http://multics-trac.swenson.org/ticket/334
http://multics-trac.swenson.org/ticket/337
http://multics-trac.swenson.org/ticket/339

This MCR proposes a set of changes found necessary for an enhancement of the ssu_ subsystem. The changes fix problems or add enhancements to subroutines used by ssu_ and its many subsystems. However, these changes apply to other software besides ssu_ subsystems. They should be considered separately on their own merits.

A) lex_string_ should support define_area_ extensible areas

Ticket 333 reminds us that define_area_ now supports creation of both freeing and non-freeing extensible areas. It therefore suggests that lex_string_.pl1 support allocations in such areas, while retaining support for the older translator_temp_ version of extensible non-freeing areas.

This can easily be done by checking the area_header of the caller-supplied area for supported standard-area version numbers (0 or 1) and the area_header.flags.extended switch. If all these attributes are found, the area will be treated as supplied by define_area_ allowing lex_string_ token descriptors to be created using a PL/I alloc statement; otherwise, the descriptors will still be created using the translator_temp_-supplied allocate subroutine.

B) define_area_ ignores the ssu_$get_area comment argument

The ssu_$get_area subroutine includes a comment argument documented as follows.

```
help ssu_$get_area -ca comment

>doc>info>ssu_.info   (54 lines in entry point)

Entry:  1983-01-19  ssu_$get_area

Arguments:
comment
   is a comment identifying the use to which the area will be put.
   (Input) It is used in constructing the owner name for the call to
   define_area_, in the form:
           subsys_name.N (comment)
   where subsys_name is the name of the subsystem, N is the invocation
   level for this invocation, and the comment (if any) follows, in
   parentheses.  This is done to make it easier to identify the segment
   names listed by list_temp_segments.
```

ssu_temp_mgr_$get_area stores the "subsys_name.N (comment)" string in the area_info.owner element when it calls define_area_ to create an area with specified characteristics in a temporary segment.

Ticket 337 reports that define_area_ embeds the area_info.owner field in a caller argument it passes to the get_temp_segment_ subroutine. However, because area_info.owner is a non-varying character string, define_area_ truncates the .owner field before the first SP character, then appends a ".area" suffix to that owner. Since ssu_$get_area places its caller's comment in parenthesis preceded by a SP character, the comment argument to the ssu_$get_area subroutine is never used. This approach represents a defect.

For example, a call to ssu_$get_area by the test_parms subsystem with comment = "token_area" generates a temporary segment with the comment (displayed by list_temp_segments command) as follows...

```
list_temp_segments

          22 Segments,  8 Free
...
!BBBKXkgPkQlgbk.temp.0454  test_parms.1.area
```

A preferable display would include the caller-provided comment rather than the ".area" suffix added arbitrarily by define_area_. For example...

```
list_temp_segments

          22 Segments,  8 Free
...
!BBBKXkgPkQlgbk.temp.0454  test_parms.1 (token_area)
```

C) call command mis-allocates storage for decimal(P) unaligned arguments

The Multics data types involving 4-bit decimal digits are selected by the PL/I Attributes:

- real fixed decimal (P) unaligned:
  which selects data type 43: real_fix_dec_4bit_bytealigned_ls_dtype
- real float decimal (P) unaligned:
  which selects data type 44: real_flt_dec_4bit_bytealigned_dtype

The Multics Programmer's Reference Guide (AG91-04) Appendix D (page D-18) reports that stored scalar or array values of these data types must begin on a word boundary and occupy a full number of words. This requirement was verified by checking array argument descriptors created by the PL/I compiler. These argument descriptors report the offset in bits between adjacent array elements.

Ticket 334 reports that calculations made by the call command for these data types do not account for this word-boundary alignment, nor the requirement that each scalar element occupy a full number of words. These calculations are made in the storage_for_pl1_dtype subroutine whose code resides in the call_dtype_fcns.incl.pl1 include segment.

The proposed change to this calculation is shown below.

```
A209  unalStoreSize (real_fix_dec_4bit_bytealigned_ls_dtype):
                                  /* +2 for sign nibble + round up to full byte boundary.   */
A210     call st(BOUNDARY.Byte, divide(dsize+2, packed_digits_per_character,24,0));
A211  unalStoreSize (real_flt_dec_4bit_bytealigned_dtype):
                                  /* +2 for sign nibble + round up to full byte boundary.   */
A212                              /*  +1 after divide for the binary exponent field.        */
A213     call st(BOUNDARY.Byte, divide(dsize+2, packed_digits_per_character,24,0)+1);
Changed by B to:
B242  unalStoreSize (real_fix_dec_4bit_bytealigned_ls_dtype):
                                  /* +1 for sign nibble + round up to end on word boundary. */
B243     call st(BOUNDARY.Word,
B244        divide( dsize + 1 +
B245                (packed_digits_per_character * characters_per_word) - 1,
B246                (packed_digits_per_character * characters_per_word), 24, 0);
B247  unalStoreSize (real_flt_dec_4bit_bytealigned_dtype):
                                  /* +1 for sign nibble +2 for the binary exponent field    */
B248                              /*    rounded up to full word boundary                    */
B249     call st(BOUNDARY.Word,
B250        divide( dsize + 3 +
B251                (packed_digits_per_character * characters_per_word) - 1,
B252                (packed_digits_per_character * characters_per_word), 24, 0);
```

D) get_equal_name and check_equal_name_ improvements

Ticket_339 reports problems in subroutines which handle equal names. When the get_equal_name_$check_equal_name_ entry point was added, the change did not add the name check_equal_name_ onto the bound_library_1_ object segment. That name is needed to permit this subroutine to be referenced as check_equal_name_. Instead, it must be referenced as get_equal_name_$check_equal_name_.

Also, the change did not add an include file with named constants defining the EQUAL_TYPE_xxx values that can be returned in the subroutine's code argument (in addition to the possible error_table_ values that argument also returns).

The get_equal_name_.info segment also does not document these EQUAL_TYPE_xxx values. Any programmer that needs to use the check_equal_name_ entry point must: examine existing callers code to see how to use the results; or examine get_equal_name_$check_equal_name_ entry point code to see what code values it returns.

These omissions boost importance of this bug from an enhancement to a defect.

This MCR proposes the following changes.

1. Change bound_library_1_.bind: As part of the objectname: get_equal_name_; entry, add a synonym: keyword naming check_equal_name_ as a full synonym for the get_equal_name_ object segment. This was cause the loader to add check_equal_name_ as a name on >sl1>bound_library_1_ bound object containing get_equal_name_.

2. Create a >ldd>incl>check_equal_name.incl.pl1 segment defining three new constants describing the non-failing code values that may be returned by check_equal_name_$check_equal_name_.

```
declare   (
          EQUAL_TYPE_USE_ORIGINAL_NAME  initial (2),
                                    /* Equal name is == or ===; it selects full original name.*/
          EQUAL_TYPE_USE_GET_EQUAL_PROC initial (1),
                                    /* Equal name selects some parts of original name.       */
          EQUAL_TYPE_USE_EQUAL_NAME     initial (0)
                                    /* Equal name has no = or % chars; it replaces original.  */
          )                           fixed bin (2) static options (constant);
```

3. Create a new check_equal_name_.info segment to document the calling sequence of the check_equal_name_ entry point under its new, shorter name. It will also refer the user to the values in the new check_equal_name.incl.pl1 segment.

4. Remove use of underlining in the get_equal_name_.pl1 comments which give details about the finite state machine implementing that program. These comments are unreadable on a video terminal. Also replace \014 pagination chars used throughout with the more modern %page; macro.

Proposed Changes

```
Ticket_333
  L-1) Change lex_string_ to allocate token descriptors in either: a
       no-freeing extensible area created by translator_temp_$get_segment;
       or an extensible area (freeing or no-freeing) created by define_area_.
       Note: ssu_$get_area can create extensible areas using define_area_.

 Ticket_334
  C-1) In call_dtype_fcns.incl.pl1, change handling of dec(P) unaligned numeric
       values, which are 4-bit byte-aligned but always begin on word-aligned
       boundary and are left-justified in a full number of words.
  C-2) Recompile call.pl1 (the only includer of this include segment) to
       pickup the change.

 Ticket_337
  D-1) In define_area_.pl1, honor entire area_info.owner field when possible.

 Ticket_339
  E-1) Change bound_library_1_.bind:
       As part of the objectname: get_equal_name_; entry:
          add a synonym: keyword naming check_equal_name_ as a full synonym
          for the get_equal_name_ object segment.  This will cause the loader
          to add check_equal_name_ as a name on >sl1>bound_library_1_ bound
          object containing get_equal_name_.
  E-2) Create check_equal_name.incl.pl1 segment which declares three constants
       that can be returned by the check_equal_name_ subroutine.
  E-3) Add a check_equal_name_.info segment to fully document this existing
       subroutine under its new, shorter name: check_equal_name_.
  E-4) Edit get_equal_name_.info to remove reference to the check_equal_name_
       entry point in the get_equal_name_ subroutine.  Item (6) above substitutes
       for the removed entry point documentation.
  E-5) Remove use of underlining in comments atop get_equal_name_.pl1 source
       which detail the finite state machine implementing that program.  These
       comments are unreadable (because of use of backspace chars) on today's
       video terminals.
```

```
Changes --

 Bound_obj:    bound_library_1_                IN: hard  UPDATE;
   bindfile:   bound_library_1_.bind            REPLACE;
   source:     define_area_.pl1                 REPLACE  compiler: pl1 -ot;
   source:     get_equal_name_.pl1              REPLACE  compiler: pl1 -ot;

 Bound_obj:    bound_proj_admin_               IN: sss  UPDATE;
   source:     lex_string_.pl1                  REPLACE  compiler: pl1 -ot;

 Bound_obj:    bound_trace_stack_              IN: sss  UPDATE;
   source:     call.pl1                         REPLACE  compiler: pl1 -ot;

 Include:      call_dtype_fcns.incl.pl1        IN: sss.include  REPLACE;
 Include:      check_equal_name.incl.pl1       IN: sss.include  ADD;

 Info:         check_equal_name_.info          IN: sss.info  ADD;
 Info:         get_equal_name_.info            IN: sss.info  REPLACE;
 Info:         lex_string_.info                IN: sss.info  REPLACE;
```

The finite state machine diagram from get_equal_name_.pl1 is shown below.

```
Algorithm

1) Parse the equal name into components and subcomponents.
   a) each name component is classified into one of the following types:
               type      format     description
                13        ===       a triple equal sign component
                14        ==        a double equal sign component
                15        OO        a component of one or more other characters
                                    (characters besides "=", "%", or ".")
                16        O=%       a component containing %'s or ='s and other chars
                17        =         a single equal sign component
                18        BAD       a bad equal name component format.
   b) each component is, in turn, divided into one or more subcomponents of the
      following types:
               type      format     description
                1         O's       a string of one or more consecutive other chars.
                2         %'s       a string of one or more consecutive %'s.
                3         ='s       a string of one, two, or three consecutive ='s.
      Thus, component types === (13), == (14), OO (15), and = (17) are each composed
      of a single subcomponent, and component type O=% (16) is composed of two
      or more components.

2) Parse the entry name into components.

3) Construct the target name from the parsed equal and entry names.
```

The hardest part of this process is the parsing of the equal name into components and subcomponents, while at the same time checking for a bad equal name. The current restraints on the format of equal names are:

1) The equal name is an entry name composed of 32 or fewer ASCII characters or spaces, excluding ">" and "<". That is,
   0 < length(equal) <= 32
   Ei <= PAD
   Ei ^= ">"
   Ei ^= "<"
   where Ei are the characters of the equal name.

2) The equal name is composed of from one to sixteen components, none of which may be null. That is, the equal name may not begin or end with a period ("."), and may not contain two or more consecutive periods.

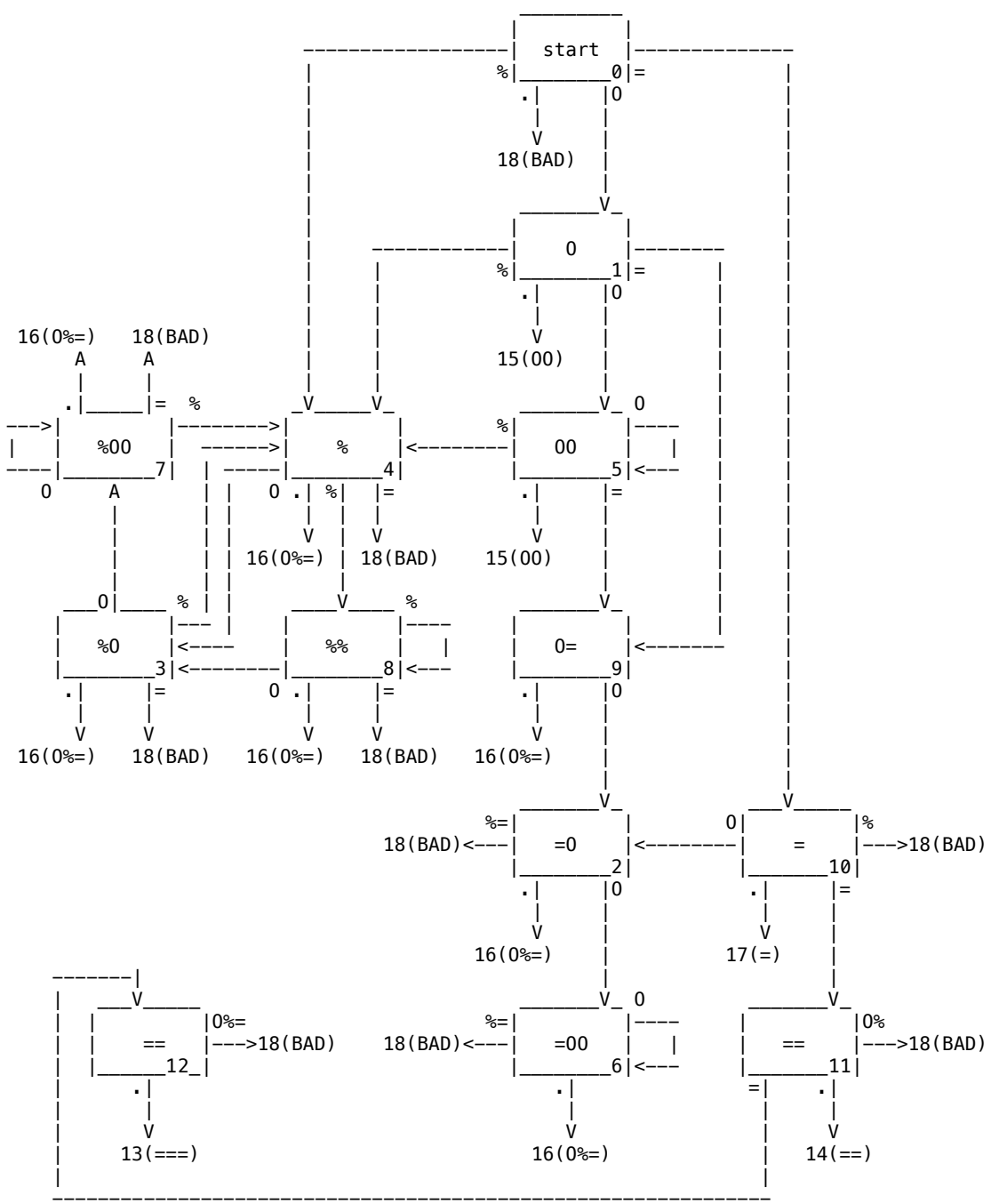
3) Each component of an equal name may contain one or more %'s, each of which represents the corresponding letter in the corresponding component of the entry name.

4) Each equal name component may contain one =, which represents the corresponding component of the entry name. An equal component which contains an = may not contain %'s.

5) An equal name may contain, at most, one == (double equal sign) component in any component position which represents all components of the entry name which are not represented by other components of the equal name.

6) An equal name may contain one or more === (triple equal sign) components in any component position which represents the original name. If === is used, no == component or component containing %'s or ='s may be used.

Each component of the equal name is parsed by a finite state machine, the diagram of which is shown below. The process of parsing each component begins in state start and it continues through the machine, according to the characters of the equal name component ("=", "%", ".", or another character) until a period ends the component, or until all the characters of the equal name have been exhausted. The parsing process identifies bad equal names (terminal state BAD), and it also classifies each valid component as being one of the component types mentioned above. The type is defined to be the state value of the finite state machine when the component has been parsed. During the parsing, information about the location and length of each component and subcomponent are gathered for later use during the construction of the target name.


start
%|0
.|0
18(BAD)
0
%|1
.|0
15(00)
16(0%=)
18(BAD)
%00
7|
%
4|
00
5|
0 .| %|=
16(0%=)
18(BAD)
15(00)
___0|
%0
3|
%%
8|
0=
9|
0 .|=
16(0%=)
18(BAD)
16(0%=)
18(BAD)
16(0%=)
%=|
18(BAD)<---|
=0
2|
0|
=
10|
.|=
16(0%=)
17(=)
0%=
==
12_|
18(BAD)
%=|
18(BAD)<---|
=00
6|
0%
==
11|
18(BAD)
13(===)
16(0%=)
14(==)

Testing

Code in get_equal_name_.pl1 was changed only to use the named constants defined in the new check_equal_name.incl.pl1 (rather than referencing them by numeric constants). No special testing beyond recompilation is required. The changes to bound_library_1_.bind were checked to see that the check_equal_name_ name was added to that bound object.

The change to define_area_ was tested by exercising the changed code dealing with the area_info.owner field processing to ensure that a suitably-short owner (with any parenthesized list) would be included in the comment displayed by the list_temp_segments command.

The changes to lex_string_ were tested by verifying that token descriptor structures could be generated in an area provided by define_area_; and that generation of structures in a translator_temp_ segment continued to work correctly.

The changes to call were verified by ensuring that a decimal(P) unaligned value is correctly handled by a test program.

The new/changed info segments were checked using the verify_info (vi) command.

Documentation

The info segment added for check_equal_name_ is shown below.

```
Entry:  2024-09-13  check_equal_name_

Function:  This subroutine checks the validity of an equalname
based on rules of the equal convention.  For information about these
rules, type:  help equal_name.gi

Syntax:
declare check_equal_name_ entry (char(*), fixed bin (35));

call check_equal_name_ (equalname, code);

Arguments:
equalname
   is the equal string to be checked.  (Input)
code
   is a standard system error code to diagnose an error in equalname
   formation; or it can be an EQUAL_TYPE value which summarizes how
   that equalname selects characters or name components from an
   original name. (Output) See "List of equal_type values" and
   "List of system error codes" below.

List of equal_type values:
   The following constants are declared in the
   check_equal_name.incl.pl1 include segment.
EQUAL_TYPE_USE_ORIGINAL_NAME (2)
   Equal name is "==" or "===".   Either string selects the full
   original name for the target name.  Caller may copy the original
   name to the target name.
EQUAL_TYPE_USE_GET_EQUAL_PROC (1)
   Equalname selects some parts of original name as the target name.
   Call the get_equal_name_ or get_equal_name_$component subroutines to
   determine the correct target name.
EQUAL_TYPE_USE_EQUAL_NAME (0)
   Equalname has no = or % chars; so it replaces original.  Caller may
   copy the equalname as the target name.

List of system error codes:
   The error code may be one of the following.
error_table_$bad_equal_name
   the equal name has a bad format.
```

Version History

INSTRUCTIONS: End with a version table including details shown below.

Date	Revision	Author	Comment
2025-07-22	1.0	Gary Dixon	Initial version of this MCR.