Subject: MCR10101, Uniform Numeric Strings for Multics Programs
Author: Gary Dixon
Date: June 13, 2022

Abstract

This MCR proposes installing changes supporting uniform numeric strings introduced by MTB-1006.

Table of Contents

Introduction

MTB-1006: Uniform Numeric Strings describes the need for changes to numeric strings on Multics and details an initial set of changes. This MCR proposes installation of these changes which may be summarized as follows.

1. Create a new radix_indicator_string_.pl1 subroutine to centralize processing of a radix indicator sub-field appended to the end of any numeric string. The indicator begins with an optional underscore, followed by one or more indicator character(s). This subroutine accepts the superset of indicator characters currently supported in various commands: letters b q o d x; plus the more general rN radix string covering bases from 2 through 16. Indicator characters may be either uppercase or lowercase.
   - radix_indicator_string_ will be installed with cv_integer_string_ in bound_library_1_.
   - Documentation for radix_indicator_string_ is included in MTB-1006.

2. Change the existing cv_integer_string_.pl1 to call radix_indicator_string_ to handle any radix indicator sub-field appended to its input string. It currently accepts a subset of the radix indicators, and evaluates the radix indicator itself.

   Also revise this subroutine to follow PL/I rules for converting integer strings to a fixed bin(35) data type. These rules govern the order in which sub-fields of the numeric string are evaluated, and the methods used to report errors as signaled conditions.
   - Modified documentation for cv_integer_string_ is included in the Documentation section below.

3. Merge functionality of the cv_dec_.pl1 subroutine into cv_integer_string_.pl1. This enables the following entry points to support radix indicator strings:
   cv_(binary dec oct hex)_ and cv_(binary dec oct hex)_check_
   - Modified info segments documenting new input strings accepted by these subroutines is included in the Documentation section.

4. Create a new cv_fixed_point_string_.pl1 subroutine which converts an input string to a float decimal(59) data type. Such *fixed-point strings* may include a radix point in their mantissa. Two optional sub-fields may follow the mantissa: an exponent sub-field; and radix indicator sub-field. This subroutine will accept strings expressed in bases 2 through 16, where either the default_base argument or a trailing radix indicator in the input string specifies the numeric base in which mantissa characters are expressed.
   - This new subroutine will call radix_indicator_string_ to evaluate any radix indicator sub-field.
   - cv_fixed_point_string_ will be installed in bound_library_1_ where cv_integer_string_ resides.
   - The cv_fixed_point_string_.info segment is included in the Documentation section below. It includes a fixed_point_string.gi.info block describing fixed-point numeric strings.

5. Change command/active function programs that operate with float decimal(59) values to call cv_fixed_point_string_ which will enable them to accept inputs in bases 2 through 16.
   hex.pl1 defines: binary octal decimal hexadecimal
   plus.pl1 defines: plus minus times divide quotient mod max min trunk floor ceil
   - Changed info segments for these command/AFs were included in MTB-1006.

   Add a new round command/AF to plus.pl1. It will call the existing move_r_or_t_ subroutine to perform a rounding operation on the float dec(59) data returned by cv_fixed_point_string_.
   - A round.info segment is included in Documentation below.

6. Create a new cv_condition_.pl1 subroutine to simplify reporting of conditions that can occur when converting numeric strings to an arithmetic data type: conversion, fixedoverflow, overflow, size, and underflow. It will provide PL/I oncode information to characterize each condition, plus onsource, onchar, and onchar_index details for a conversion condition. Much of this information is difficult to obtain when establishing an on-unit in a program accepting numeric inputs.
   - Add cv_condition_ to bound_multics_bce_. Both plus and calc will use cv_condition_, and these commands reside in that bound segment.
   - Documentation for cv_condition_ is in MTB-1006.

7. The calc.pl1 command/AF (calc 1.1) currently performs arithmetic using the float bin(27) data type. Change calc version 2.0 to use the more precise float decimal(59) data type instead. calc 2.0 will:
   - Call cv_fixed_point_string_ to convert numeric strings to float dec(59).
   - Call cv_condition_ to report conditions signaled while entering input numbers.
   - Call numeric_to_ascii_ to convert displayable float decimal(59) numbers to output strings.
   - Add trig functions which accept an input in degrees. The current functions accept only radian input.
   - Change log functions to match PL/I built-in names:
     `log` calculates ln(x); `log10` calculates log base 10(x); `log2` calculates log base 2(x)
   - Add db_on and db_off requests to display calc's internal push-down-stack used for evaluating expressions. This can aid user debugging of complicated expressions. This aid was needed to verify the above changes in calc 2.0.

   calc 2.0 no longer needs the separate ffip.pl1 and ffop.pl1 support routines (currently called only by calc) for getting input and displaying output numbers. Eliminating these two subroutines leaves only calc.pl1 in bound_calc_.
   - Move calc.pl1 into bound_multics_bce_ where plus resides.
   - Revised calc.info appears in the Documentation section below.

8. The interpret_info_struc_.pl1 subroutine is called by condition_interpreter_, default_error_handler_, et al. to create an error message describing arithmetic conditions like conversion, overflow, size, and underflow. Change it to provide more detailed information in its message about a conversion condition: the exact character causing the condition from mantissa, exponent, or radix indicator.
   - This is a non-retained object of bound_error_handlers_; therefore no user documentation exists.

To implement and test the above changes, several unplanned changes were also required. These are described in the next two sections.

Problems with numeric_to_ascii_base_

The test program for cv_fixed_point_string_ uses the numeric_to_ascii_base_ function to display the returned float dec(59) output. While testing with this program, several problems were found in the user interface specification and implementation of numeric_to_ascii_base_. These problems had to be corrected to successfully complete testing of cv_fixed_point_string_.

The numeric_to_ascii_base_ user interface is patterned after the numeric_to_ascii_ interface. It just adds a final *base* argument to the numeric_to_ascii_ calling sequence. By following numeric_to_ascii_'s calling sequence, two design flaws were introduced into numeric_to_ascii_base_. In describing these flaws, the current numeric_to_ascii_ interface is shown below for reference.

```
Syntax:
declare numeric_to_ascii_ entry (float dec(59), fixed bin) returns
     (char(72) varying);
result = numeric_to_ascii_ (value, precision);

Arguments:
value
   number to be formatted. (Input)
precision
   maximum number of significant digits placed in the result string.
   (Input)  If 0 is given, from 1 to 59 significant digits are placed
   in the result string depending on the value being formatted.  If
   precision is less than 0, then at most -precision significant
   digits are returned with excess digits truncated from the result.
   If precision is greater than 0, the excess digits are rounded
   upward to form the result having at most +precision significant digits.
result
   character-string representation of value expressed with decimal
   digits. (Output)  It contains no spaces.
```

The first two flaws in numeric_to_ascii_base_ are described below. These flaws will be corrected by revising the user interface to this function.

A. precision argument: This argument of numeric_to_ascii_ supports either rounding or truncation of the decimal digits in the *value* mantissa before it is converted to a string. The move_r_or_t_ assembler subroutine operates directly on these digits using an MVN instruction to move the float dec(59) value to a float dec(*precision*) temporary, adding a round option to do the rounding, or omitting that option to do truncation.

Such methodology does not work for numeric_to_ascii_base_ interface. For its *precision* argument, the significant digits of the binary, octal, hex or other non-decimal *result* string are not known until after the decimal digits of the float dec(59) mantissa are converted to the target base. Since each decimal digit in the mantissa can represent several digits in a small-base output string (like binary), or only part of a digit in a large-base output string (like hex), any rounding or truncating of input digits affects the wrong number of digits, and may incorrectly round upwards rather than downwards.

Round or truncate operations would instead have to be performed on digits expressed in the target base; that is, on the *result* string's mantissa characters. But trying to round character positions in the *result* string would be difficult to implement. Rounding up could require increasing the value of several characters left of the *precision* rounding place in the *result* string. There are no existing source programs supporting non-decimal arithmetic operations on numeric characters in a string.

Therefore the corrected interface for numeric_to_ascii_base_ will only truncate to a particular number of *result* radix places for a positive *precision* argument. A zero or negative *precision* argument will allow all significant radix places to be returned in the *result*.

B. **result**: The function return value for numbers in bases less than 10 can be much longer than the 72 characters returned by the numeric_to_ascii_ interface pattern. A return value expressing the largest supported input number as a binary string might need 200+ characters to represent the 59 significant decimal digits in a float dec(59) input *value*.

PL/I runtime supports conversion of both binary and decimal strings to numeric data types. It declares its onsource value (saved by PL/I runtime routines if a conversion condition occurs) as a char(256) varying string. Therefore, that data length will be used for the corrected numeric_to_ascii_base_ *result*.

The following issues will also be corrected in the numeric_to_ascii_base_ implementation to improve accuracy of the *result* value. The inaccuracies stem from incorrect implementation of the standardized algorithm for expressing decimal digits in another base.

C. **integer algorithm**: The standard algorithm separates the input float dec(59) *value* argument into two parts: an integer float dec(59) and a fraction float dec(59). The rightmost character in the integer part of the *result* mantissa is calculated by dividing the float dec(59) integer part of *value* by the *base*. The remainder from this division gives a digit value to be converted to a *result* character expressed in the target *base*. The quotient from the division becomes the new dividend when calculating the next mantissa character. This division process repeats until the quotient is zero, with each remainder giving the value of a result character, and integer result characters being generated from right-to-left.

The existing numeric_to_ascii_base_ function calculates the quotient and remainder using the statements:

```
new_quotient = trunc (divide (quotient, base, 59));
remainder = quotient - (base * new_quotient);
char1 = dig_ch (remainder);
```

Code generated by PL/I for the divide built-in function includes a DV3D instruction with rounding. However, rounding up the quotient conflicts with the requirements of the base conversion algorithm. That algorithm depends on having an unrounded integer quotient to obtain a correct remainder. This rounding causes the existing numeric_to_ascii_base_ to sometimes return an integer part of *result* whose final character is greater than the largest digit allowed for the target base (i.e., an octal *result* ending in an "8" character).

No method was found to convince PL/I to generate a DV3D instruction without rounding. The revised numeric_to_ascii_base_ will therefore call a new decimal_no_round_$divide subroutine to avoid this rounding problem. This new subroutine performs a single DV3D operation without round, then returns both the unrounded quotient and the indicator bits set by the DV3D instruction. Any of the following indicators might denote a divide failure: truncation of quotient digits, overflow of quotient mantissa, exponent overflow or underflow in the quotient. numeric_to_ascii_base_ will check those indicator bits after each divide operation.

D. **fraction algorithm**: The standard base conversion algorithm calculates fractional mantissa characters of the *result* string from the fractional part of the *value* argument. This part is used as the multiplicand when computing the first fractional character of the *result* mantissa.

product = *base* * multiplicand.

If product > 1.0, the integer part of that product is the value of the next character in the fractional component of the *result* mantissa; otherwise, that character is "0". Subtracting that digit value from the product gives the multiplicand for calculating the next character of that *result* mantissa. The loop continues until the multiplicand is 0.0, or until the required number of fractional digits are obtained (as specified by the *precision* argument).

Conversion to some bases can generate in an infinite sequence of fractional *result* characters, so algorithm iterations must be limited by honoring the *precision* argument. But the existing numeric_to_ascii_base_ imposes the wrong limits on this loop. If the *precision* argument is 0, the existing code uses a default precision of 59 irrespective of the requested target base. For a small-base, this default max precision is far too small: a binary *base* should have a default precision of 199 mantissa characters; an octal *base* should have default precision of 66. For a hex *base*, the default precision should only be 49. Always defaulting the precision to 59 causes the algorithm for calculating fractional digits to end too soon or too late, producing drastically shortened *results* (even producing 0.0 for some small non-zero fractional input values) or *results* containing too many digits.

Furthermore, for an input *value* containing only a fractional part, the precision limit must not be imposed until the first non-zero fractional digit has been found. Those leading zeros in the fraction are not significant. The existing code makes no such allowance when imposing its precision limit. This again causes 0.0 to be incorrectly returned for a very small fraction-only input *value*.

The revised numeric_to_ascii_base_ will use the correct default precision calculated for each base, and will impose a precision limit on the fraction calculation loop only after the first non-zero integer or fractional digit has been found in the output string.

E. **float dec(59) inaccuracy**: The integer and fraction algorithms described above could be totally accurate only if the calculations were performed with data elements holding an infinite number of digits (using a hypothetical float dec(∞) data element). Calculations using float dec(59) data items lose some accuracy when computing mantissa characters for a *result*. A base 3 input *value* demonstrates this inaccuracy. The following shows an input value being converted to a float dec(59) data element by cv_fixed_point_string_; and this data element then being converted back to a fixed-point string expressed in base 3 by the current numeric_to_ascii_base_.

numeric string: 0.11_r3
float dec(59) *value*: 0.44444444444444444444444444444444444444444444444444444444444444
inaccurate *result*: 0.10222222222222222222222222222222222222222222222222222222222222_r3

Some precision is lost in converting the base 3 numeric string to a float dec(59) data element. A float dec(∞) data item would store an infinite number of "4" digits in this fixed-point string conversion. Our float dec(59) data item can hold only 59 such "4" digits.

Further inaccuracy occurs in the calculations to convert the float dec(59) fraction back to a base 3 numeric string. This accuracy loss often produces a *result* string that is slightly less than the original input string such as seen in the example above.

The revised numeric_to_ascii_base_ code tries to detect inaccuracies associated with slight data losses, and compensate for these loses in several ways. These compensations correct the data loss in many cases, and reduce its effects in other cases. In all testing so far, these compensations provide more accurate results than the existing, flawed numeric_to_ascii_base_ code.

The revised numeric_to_ascii_base_ subroutine will have the following user interface.

```
Syntax:
declare numeric_to_ascii_base_ entry (float dec(59), fixed bin,
   fixed bin) returns (char(256) varying);

result = numeric_to_ascii_base_ (value, precision, base);

Arguments:
value
   number to be formatted. (Input)
precision
   maximum number of significant digits placed in the result string.
   (Input)  If a 0 or negative precision is given, all significant
   digits are represented in the result.  A positive precision
   limits number of significant digits to the given precision.
base
   radix of the numbering system in which the result mantissa is
   expressed. (Input)  For example, base=2 produces a binary
   representation and base=8 produces an octal string.  The domain
   supported for base is:   2 <= base <= 16
```

```
result
   character-string representation of value using digits of the
   base numbering system in the mantissa. (Output)
```

Problem with help_

Change 3 described above (merge functionality of the cv_dec_.pl1 subroutine into cv_integer_string_.pl1) also involves adding text in the info segments describing each cv_XXX_ and cv_XXX_check_ subroutine. Each subroutine has only a single entry point, so each info block is currently stored in a separate info segment. This separation makes it difficult to update all the info blocks when a code change affects every subroutine in the same ways. Grouping these info blocks into a single info segment would simplify such updates by allowing all changes to be done via editing of a single file.

help_ allows single-entrypoint info blocks to be grouped in the same info_seg if each block begins with an :Info: divider replacing the :Entry: divider normally used for each entry point of a subroutine. But putting more than one of these single-entrypoint info blocks into one info_seg exposed a minor bug in help_ code. When help_ displays the final paragraphs describing a given entry point, it mistakenly thinks the next info block describes another entry point of the same subroutine; such info block really describes a single entrypoint of a DIFFERENT subroutine. So help asks if the user wants to see information about that subsequent info block. That is a bug.

This bug was not exposed earlier because there are very few subroutines having only one entrypoint, and none of the info segments describing those subroutines contain more than one :Info: block.

This MCR will repair this just-exposed help_ bug as follows. When a help_ argument selects a single-entrypoint info block for display, help_ will temporarily unthread any other blocks from that info segment's iFile structure. When display of the selected info block completes, help_ will reattach threads for those blocks back to the iFile structure. This tiny change (<50 lines of code) resolves the problem without interfering with the complex mechanisms enabling help_ and its supporting routines to display multi-entrypoint subroutine info segments.

Proposed Changes

The following build script describes the segments being modified to make all of the changes described above. The description lists program changes ordered by the bound segment in which each source program does/will reside. Info segments changing to reflect these source modifications are mentioned with their associated source (even though the info segments are not part of the given bound segment).

```
Description:

   bound_library_1_:
 A) Create a new radix_indicator_string_.pl1 subroutine to uniformly handle
    radix strings at the end of all numeric strings.
 B) Rewrite cv_integer_string_ to use radix_indicator_string_ for all of
    its conversions (instead of using an internal procedure for this service),
    and to follow PL/I rules for converting integer strings to fixed bin(35).
    cv_integer_string_ signals conversion, size and error conditions using PL/I
    runtime mechanisms.
 C) Merge all functionality of cv_dec_.pl1 into cv_integer_string_.pl1.  This allows
    cv_(binary oct dec hex)_ and cv_(binary oct dec hex)_check_ routines to
    accept a trailing radix indicator and centralizes code for converting integer
    strings to fixed bin(35) data.
 D) Add new cv_fixed_point_string_.pl1 to bound_library_1_ to support conversion
    of fixed-point strings in bases 2-16 to float dec(59) data type.  It also
    uses radix_indicator_string_ to evaluate any radix suffix ending the string.
    It follows PL/I rules for converting fixed-point strings to float dec(59),
    and signals conversion, underflow and overflow using PL/I runtime mechanisms.
    Add cv_fixed_point_string_.incl.pl1 to declare constants used for arguments
    passed to cv_fixed_point_string_ subroutine.

 E) Repair incorrect info in cv_integer_string_.info and
    cv_integer_string_check_.info.  Merge documentation from the cv_XXX_.info
    segments into the cv_integer_string_.info segment (as separate info blocks).
 F) Add radix_indicator_string_.info and cv_fixed_point_string_.info.  Include a
    fixed_point.gi.info block in cv_fixed_point_string_.info to give general details
    of fixed-point numeric strings.

   bound_multics_bce_:
 G) Add decimal_no_round_ to bound_multics_bce_ for use by numeric_to_ascii_base_.
 H) Add cv_condition_ to bound_multics_bce_ to make this subroutine available
    to plus.pl1 and calc.pl1.  Its entry points will be retained for use by other
    programs.
 I) Merge code for numeric_to_ascii_ into numeric_to_ascii_base_.pl1 so both can
    share routines for formatting the output result.  Correct/enhance algorithms
    used by numeric_to_ascii_base_.
 J) Replace plus in bound_multics_bce_ with a version that calls
    cv_fixed_point_string_ to convert numbers.  Add the round command/AF to
    plus.pl1; the round command/AF calls existing move_r_or_t_ to do rounding.
 K) Move calc from bound_calc_ to bound_multics_bce_.  calc 2.0 uses:
     - float dec(59) for its arithmetic operations (replacing the
       less accurate float bin(27) values);
     - cv_fixed_point_string_ to convert numeric strings;
     - cv_condition_ for display of errors found by cv_fixed_point_string_;
     - numeric_to_ascii_ to convert numbers to strings for display.

    calc 2.0 adds new db_on/db_off requests to display its push-down stack
    to aid in debugging EXPRESSIONs as they are evaluated.  It adds trig
    functions that accept input in degrees (the existing routines
    accept/return radians).  And it changes log functions to follow PL/I
    naming conventions: log calculates ln(x); log10 calculates log base 10(x);
    log2 calculates log base 2(x).
```

```
 L) Replace >sss>pl1.dcl to remove incorrect declarations for numeric_to_ascii_ and
    numeric_to_ascii_base_ subroutines.  The entry descriptors in these two
    subroutines are used by display_entry_point_dcl and get_entry_point_dcl_ to get
    these subroutine declarations.
 M) Replace numeric_to_ascii_.info with an updated segment that also includes
    numeric_to_ascii_base_.info describing that subroutine.
 N) Update the plus.info and related info segment blocks to describe above
    changes; add a round.info block describing new round command/AF.
 O) Update arithmetic_af.gi.info to reference the new round.info.
 P) Add cv_condition_.info to document the two entry points of this new subroutine.
 Q) Add decimal_no_round_.info to document the two entry points of this subroutine.
 R) Update the calc.info segment for calc 2.0 changes.

   bound_full_cp_:
 S) Change hex.pl1 to call cv_fixed_point_string_ instead of doing its own
    conversions.

 T) Merge info for all commands implemented by hex.pl1 into hex.info, and
    improve documentation for fixed-point strings (accepted by cv_fixed_point_string_)
    which are now available in each of these command/AFs.

   bound_error_handlers_:
 U) Change interpret_info_struc_.pl1 to enhance onsource information when
    reporting a conversion condition by including a pointer to the character
    identified by onchar in that onsource.

   bound_help_
 V) Change help_ to properly support info blocks each describing a single-entrypoint
    subroutine when those blocks are grouped together in a single info segment.

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

 Build_script:            cvis.mb;

 Bound_obj:               bound_calc_                    IN: sss  DELETE;

 Bound_obj:               bound_library_1_               IN: hard  UPDATE;
     bindfile:            bound_library_1_.bind           REPLACE;
     source:              cv_dec_.pl1                     DELETE;
     source:              cv_fixed_point_string_.pl1      ADD;
     source:              cv_integer_string_.pl1          REPLACE;
     source:              radix_indicator_string_.pl1     ADD;

 Bound_obj:               bound_multics_bce_             IN: hard  UPDATE;
     bindfile:            bound_multics_bce_.bind         REPLACE;
     source:              numeric_to_ascii_.pl1           DELETE;
     source:              plus.pl1                        REPLACE;
     source:              cv_condition_.pl1               ADD;
     source:              decimal_no_round_.alm           ADD;
     source:              numeric_to_ascii_base_.pl1      REPLACE;
     source:              calc.pl1                        ADD;

 Bound_obj:               bound_full_cp_                 IN: sss  UPDATE;
     bindfile:            bound_full_cp_.bind             REPLACE;
     source:              hex.pl1                         REPLACE;

 Bound_obj:               bound_error_handlers_          IN: hard  UPDATE;
     source:              interpret_info_struc_.pl1       REPLACE;
```

```
Bound_obj:              bound_help_                      IN: sss   UPDATE;
     source:             help_.pl1                         REPLACE;

Seg(dcl_file):          pl1.dcl                          IN: sss  REPLACE;

Include:                cv_fixed_point_string_.incl.pl1  IN: sss.include ADD;

Info:                   ceil.info                        IN: sss.info  DELETE;
Info:                   cv_dec_.info                     IN: sss.info  DELETE;
Info:                   cv_dec_check_.info               IN: sss.info  DELETE;
Info:                   cv_hex_.info                     IN: sss.info  DELETE;
Info:                   cv_hex_check_.info               IN: sss.info  DELETE;
Info:                   cv_integer_string_.info          IN: sss.info  DELETE;
Info:                   cv_integer_string_check_.info    IN: sss.info  DELETE;
Info:                   cv_oct_.info                     IN: sss.info  DELETE;
Info:                   cv_oct_check_.info               IN: sss.info  DELETE;
Info:                   decimal.info                     IN: sss.info  DELETE;
Info:                   divide.info                      IN: sss.info  DELETE;
Info:                   floor.info                       IN: sss.info  DELETE;
Info:                   hexadecimal.info                 IN: sss.info  DELETE;
Info:                   max.info                         IN: sss.info  DELETE;
Info:                   min.info                         IN: sss.info  DELETE;
Info:                   minus.info                       IN: sss.info  DELETE;
Info:                   mod.info                         IN: sss.info  DELETE;
Info:                   octal.info                       IN: sss.info  DELETE;
Info:                   quotient.info                    IN: sss.info  DELETE;
Info:                   times.info                       IN: sss.info  DELETE;
Info:                   trunc.info                       IN: sss.info  DELETE;

Info:                   arithmetic_af.info               IN: sss.info  REPLACE;
                           add_name:
                             arithmetic_af.gi.info;
Info:                   binary.info                      IN: sss.info  REPLACE;
                           add_name:
                             bin.info
                             octal.info
                             oct.info
                             decimal.info
                             dec.info
                             hexadecimal.info
                             hex.info;
                             radix.info;
Info:                   calc.info                        IN: sss.info  REPLACE;
Info:                   cv_binary_.info                  IN: sss.info  ADD;
                           add_name:
                             cv_binary_check_.info
                             cv_dec_.info
                             cv_dec_check_.info
                             cv_hex_.info
                             cv_hex_check_.info
                             cv_integer_string_.info
                             cv_integer_string_check_.info
                             cv_oct_.info
                             cv_oct_check_.info;
Info:                   cv_condition_.info               IN: sss.info  ADD;
```

```
Info:                    cv_fixed_point_string_.info       IN: sss.info  ADD;
                            add_name:
                              fixed_point_string.gi.info
                              fixed_point_string.info
                              fixed_point.gi.info
                              fixed_point.info
Info:                    decimal_no_round_.info            IN: priv.info ADD;


Info:                    numeric_to_ascii_.info            IN: sss.info  REPLACE;
                            add_name:
                              numeric_to_ascii_base_.info;
Info:                    plus.info                         IN: sss.info  REPLACE;
                            add_name:
                              minus.info
                              times.info
                              divide.info
                              quotient.info
                              mod.info
                              max.info
                              min.info
                              ceil.info
                              trunc.info
                              floor.info
                              round.info;
Info:                    radix_indicator_string_.info      IN: sss.info  ADD;
                            add_name:
                              radix_indicator.gi.info
                              radix_indicator.info;
```

The following sections describe the main purpose of creating a new source program, or making changes to an existing source. More fundamental subroutine changes are listed first, followed by changes to commands/active functions that will use these subroutines; this happens to be the order in which the changes were developed and tested.

New Subroutine: radix_indicator_string_

radix_indicator_string_.pl1 implements a new subroutine for finding/evaluating any radix indicator sub-field that appears in a numeric string. If uses an algorithm excerpted from the cv_integer_string_ code. The algorithm was expanded to include all known radix indicator suffixes, and to facilitate use by a wide variety of potential calling programs.

Main callers will be conversion routines like cv_integer_string_ and cv_fixed_point_string_ subroutines which need details about whether a radix indicator sub-field is present at the end of a numeric string. If one is found, the subroutine returns the length of the numeric string without that radix indicator sub-field and the effective base selected by the sub-field. If an incorrect sub-field contents is found, the index of the failing character is provided allowing the caller to diagnose the onchar location causing the conversion error.

Other callers might just need to remove any radix indicator sub-field from a numeric string, perhaps so they could pass the shortened string to the PL/I convert built-in function, etc.

Changed Subroutine: cv_integer_string_("" check_), and cv_XXX_("" check_)

cv_integer_string_.pl1 currently implements two subroutines: cv_integer_string_ and cv_integer_string_check_. The existing source was heavily modified:

- to replace internal code for evaluating any radix indicator sub-field with a call to the new radix_indicator_string_ subroutine.
- to reorder steps taken to convert the integer string so they follow the order used by PL/I runtime convert routines; then the same diagnostics will be reported by PL/I and cv_integer_string_ when diagnosing a malformed string; and
- to incorporate functionality of the cv_(binary octal dec hex)_ and cv_(binary octal dec hex)_check_ subroutines into cv_integer_string.

cv_integer_string_.pl1 should therefore be audited as a new subroutine. Use compare_ascii to see main areas of change from original to heavily-revised code.

While only a few programs call cv_integer_string_check_, many programs call the cv_(binary octal dec hex)_ and cv_(binary octal dec hex)_check_ subroutines. Great pains were taken to ensure the new implementation of these cv_(…)_ entry points exactly matches results returned by the original version of those entry points (from cv_dec_.pl1 source). Algorithms used in cv_dec_.pl1 and cv_integer_string_.pl1 were almost identical, but cv_integer_string_.pl1 also provided code which signaled errors as a conversion condition; cv_dec_.pl1 lacked such code. So the cv_(dec oct binary hex) _ and cv_(…)_check_ routines were replaced by adding the identical entry points into cv_integer_string_.pl1 program.

The method used by the existing cv_integer_string_.pl1 for signaling conversion was not compatible with PL/I runtime. The oncode, onsource, and onchar pseudo-variables were not being set. And there was no way for a calling program to setup a conversion on-unit to intercept such errors, modify either onchar or onsource pseudo-variables, then restart the conversion. Such restarts are also a documented part of PL/I runtime signaling of the conversion condition. So the existing cv_integer_string_ code for signaling conversation was changed to:

- perform conversions in the exact order used by PL/I runtime convert built-in function;
- select an appropriate oncode integer value to describe nature of each detected conversion failure, matching the value used by PL/I runtime convert whenever possible;
- call help_plio2_signal_ to actually signal a conversion condition, passing oncode, onsource, and onchar_index values. help_plio2_signal_ is a runtime support routine designed to setup the pseudo-variables and signal the conversion condition.
- call pl1_signal_from_ops_ to signal a size or error condition if either of these situations is detected. pl1_signal_from_ops_ is a runtime support routine designed to signal size, error and other PL/I conditions.

The size condition is signaled if the integer string represents a value too large to fit in a fixed bin(35) data item. The error condition is signaled: if the integer string is longer than 256 characters (maximum length set by PL/I runtime for the onsource pseudo-variable); or if the default_base argument is outside the range: 2 <= base <= 16. Support for detecting and signaling these conditions was added to existing cv_integer_string_.pl1 code.

New Subroutine: cv_fixed_point_string_

cv_fixed_point_string_.pl1 is a new source program requiring a full audit of the code. Multics currently has no program for converting a fixed-point numeric string to a float dec(59) data item, other than the PL/I runtime convert built-in function (which uses the PL/I any-to-any operators) to convert decimal or binary fixed-point character strings to that data type. Because it is desirable for calc to support float dec(59) arithmetic operations and for plus/times/subtract/divide/quotient/etc. to support non-decimal input strings, this new subroutine was created.

Ordering of steps in its conversion algorithm was modeled after those of the PL/I runtime convert routines. This enables returning the same diagnostics for malformed fixed-point strings by both PL/I runtime convert operators, and cv_fixed_point_string_. This includes setting of oncode, onsource, and onchar pseudo-variables when conversion is signaled. Extensive tests were run to determine the PL/I convert order of steps, and to identify the full set of oncode values and onchar locations returned by PL/I for each kind of conversion condition. That ordering is documented as a comment in the cv_fixed_point_string_ source code. The tests needed to determine that ordering are included in the tcfps.ec testing exec_com. The auditor should become familiar with this ordering to better understand the code in the new subroutine.

This change includes installation of a new cv_fixed_point_string_.incl.pl1 include file which declares constants needed for the *switches* argument of cv_fixed_point_string_ subroutine. Documentation for cv_fixed_point_string_ references that include file and describes the structure and named constants it declares. So the include won't be shown here.

Changed: bound_library_1_.bind

The following cpa output shows changes to the bound_library_1_.bind file.

```
cpa [lpn bound_library_1_.bind -lb o] == -he
A >ldd>hard>object>bound_library_1_.archive::bound_library_1_.bind (original)
B >user_dir_dir>Multics>GDixon>w>cvis>bound_library_1_.bind (new)

Inserted in B:
B3
Preceding:
A3        /* Added all sorts of user ring stuff, KPL '83 */


A18         1) change(86-08-16,JSLove), approve(86-08-16,MCR7518),
A19            audit(86-08-21,Parisek), install(86-10-02,MR12.0-1174):
A20            Added check_star_name_ entrypoint.
A21         2) change(86-12-15,GDixon), approve(86-12-17,MECR0004),
A22            audit(86-12-15,Farley), install(86-12-17,MR12.0-1250):
A23            Retain ascii_to_bcd_ entrypoint of bcd_to_ascii_.
A24         3) change(86-12-18,GDixon), approve(86-12-18,MCR7599),
A25            audit(87-01-05,Farley), install(87-01-06,MR12.0-1260):
A26            Approval for MECR0004, in history comment 2.
Changed by B to:
B19         1) change(1986-08-16,JSLove), approve(1986-08-16,MCR7518),
B20            audit(1986-08-21,Parisek), install(1986-10-02,MR12.0-1174):
B21            Added check_star_name_ entrypoint.
B22         2) change(1986-12-15,GDixon), approve(1986-12-17,MECR0004),
B23            audit(1986-12-15,Farley), install(1986-12-17,MR12.0-1250):
B24            Retain ascii_to_bcd_ entrypoint of bcd_to_ascii_.
```

```
B25          3) change(1986-12-18,GDixon), approve(1986-12-18,MCR7599),
B26             audit(1987-01-05,Farley), install(1987-01-06,MR12.0-1260):
B27             Approval for MECR0004, in history comment 2.
B28          4) change(2021-10-29,GDixon):
B29             A) Move cv_dec_, cv_dec_check_, cv_hex_, cv_hex_check_,
B30                 cv_binary_, cv_binary_check_, cv_oct_, cv_oct_check_
B31                 entry points into cv_integer_string_.pl1.
B32             B) Add cv_fixed_point_string_ subroutine.
B33             C) Add radix_indicator_string_ subroutine as an objectname.
B34                It will be called by cv_integer_string_ and subroutines
B35                in other bound segments.


A37                             cv_integer_string_,
A38                             cv_dec_,
Changed by B to:
B46                             radix_indicator_string_,
B47                             cv_fixed_point_string_,
B48                             cv_integer_string_,


A111       objectname:          cv_integer_string_;
A112        synonym:            cv_integer_string_check_;
A113        retain:             cv_integer_string_, cv_integer_string_check_;
Changed by B to:
B121       objectname:          radix_indicator_string_;
B122          retain:           radix_indicator_string_;
B123
B124       objectname:          cv_integer_string_;
B125        synonym:            cv_integer_string_check_,
B126                            cv_binary_, cv_binary_check_,
B127                            cv_dec_, cv_dec_check_,
B128                            cv_hex_, cv_hex_check_,
B129                            cv_oct_, cv_oct_check_;
B130        retain:             cv_integer_string_, cv_integer_string_check_,
B131                            cv_binary_, cv_binary_check_,
B132                            cv_dec_, cv_dec_check_,
B133                            cv_hex_, cv_hex_check_,
B134                            cv_oct_, cv_oct_check_;
B135
B136       objectname:          cv_fixed_point_string_;
B137          retain:           cv_fixed_point_string_;


A118       objectname:          cv_dec_;
A119        synonym:            cv_oct_, cv_hex_,
A120                            cv_dec_check_, cv_oct_check_, cv_hex_check_,
A121                            cv_binary_, cv_binary_check_;
A122        retain:             cv_dec_,
A123                            cv_oct_, cv_hex_,
A124                            cv_dec_check_, cv_oct_check_, cv_hex_check_,
A125                            cv_binary_, cv_binary_check_;
A126
Deleted by B, preceding:
B142       objectname:          cv_float_;

Comparison finished: 5 differences, 61 lines.
```

New Subroutine: cv_condition_$display, $message

cv_condition_.pl1 implements the cv_condition_$display and cv_condition_$message entry points, as described in MTB-1006. This subroutine is needed to make it easier for conversion on-units to provide clear simple descriptions of conversion, fixedoverflow, overflow, size and underflow conditions signaled by routines like cv_integer_string_ and cv_fixed_point_string_, along with exact oncode, onsource, and onchar pseudo-variable values.

Most of the test programs written for the cv_XXX_ subroutines depend on cv_condition_ to provide complete display of those conditions. Commands like calc and plus also use cv_condition_ to report numeric string input errors to their users.

Changed Subroutine: numeric_to_ascii_base_, numeric_to_ascii_

The original MTB-1006 plan did not anticipate changes to numeric_to_ascii_base_.pl1. Once problems in that routine were analyzed (see earlier section above), a complete rewrite was needed. The standard algorithm for converting numeric data items to a fixed-point string expressed in a non-decimal base numbering system was used incorrectly in the original subroutine. That algorithm is now correctly deployed in the replacement source program. Therefore, this source needs to be tested and audited like a new source program.

Changing the numeric_to_ascii_base_ function interface should cause very few problems; the revised interface returns char(256) varying string rather than char(72) varying. The peruse_crossref tool shows current callers of this subroutine are only: hex.pl1 (with its binary, octal, decimal and hexadecimal command/AFs). A hex.pl1 using the correct declaration for numeric_to_ascii_base_ is included in this MCR.

NOTE: Multics System Release Notes should describe this numeric_to_ascii_base_ function interface change and warn that other non-Multics programs may need to be changed.

numeric_to_ascii_ code was merged into the revised numeric_to_ascii_base_.pl1 because: if numeric_to_ascii_base_ is asked to convert float dec(59) to a base 10 numeric string, the code needs to follow the numeric_to_ascii_ method of copying decimal digits from mantissa directly to the result string; and both numeric_to_ascii_ and numeric_to_ascii_base_ should output numeric strings which are formatted consistently. These requirements are most easily met if both functions are implemented by the same source program.

The formatting done in the current numeric_to_ascii_ code differs somewhat from the documentation in numeric_to_ascii_.info. In the revised numeric_to_ascii_base_ source, both subroutines now provide this documented formatting by sharing the same formatting code. The goal of such code is to choose an output format that makes it easier for users to understand the magnitude of the number expressed by the output string, and that distinguishes fraction-only values from integer.fraction values.

NOTE: Multics System Release Notes should include the following requirement:

numeric_to_ascii_base_ uses the PL/I floor built-in function, which invokes the truncate/floor PL/I operator. This operator depends upon a dps8 repair for the MVN instruction, as described in dps8m issue #197: MVN instruction fails to set overflow indicator for float dec(59) values >= +1e65. numeric_to_ascii_base_ will operate correctly only if the Multics release containing numeric_to_ascii_base_ runs atop a dps8 simulator that includes this repair. To verify that your system's dps8 simulator contains this fix, run the eis_tester command as described in MCR10116. If TESTS 279 and 280 complete without error, your system's dps8 includes the necessary repair.

NOTE: Multics System Release Notes should describe the following change to the numeric_to_ascii_base_ result string.

The numeric_to_ascii_base_ code always appends a radix indicator sub-field to the result string if it is expressed in a non-decimal base. This minor change of interface makes it clear in which numbering system the mantissa characters of the result are expressed.

This change is expected to be compatible, because: only the binary, octal, decimal and hexadecimal command/AFs call numeric_to_ascii_ base_; most programs making use of those result strings will either display them to the user, or pass them as input to other command/AF/subroutines that make use of a subroutine that can evaluate a radix indicator sub-fields in a numeric input string.

New Subroutine: decimal_no_round_$divide, $multiply

decimal_no_round_.alm is a new subroutine written primarily to satisfy a need by numeric_to_ascii_base_ for a DV3D divide instruction performed without rounding. Use of rounding was a major cause of error in the earlier numeric_to_ascii_base_ code (which is being replaced).

Only the $divide entry point is used by numeric_to_ascii_base_. (At one point, it was thought that the $multiply without round functionality would be needed, but it was not.) Both entry points were briefly tested by using the call command to invoke these subroutines and examine their outputs.

Correctness of indicator bits returned by $divide is thoroughly tested by code in the new numeric_to_ascii_base_. With the new algorithm limiting the min/max input values accepted for each base numbering system, such underflow and overflow problems seldom arise.

Changed Command/AF: calc

MTB-1006 describes the main changes to calc.pl1: converting fixed-point string inputs to arithmetic values by calling cv_fixed_point_string_; displaying errors in entered numeric strings by calling cv_condition_ from calc's on-units; and converting output values from calc expressions to fixed-point strings by calling numeric_to_ascii_.

While making these changes, the expression processing logic of calc 1.1 was changed very little. But even with minimal changes, a few problems arose from differences in exactly how input strings were identified in the request lines typed by the user. These were not understood until debugging code was added to display state of calc's push-down stack of values each time an operation was ready to be performed. This debugging code proved quite useful, so it will be retained and documented for use by users and calc developers. Tests were developed to display its use to the auditor.

The change from float bin(27) to float dec(59) arithmetic also necessitated obtaining more precise values for pi and e values which are pre-loaded onto calc's variables list. Values were obtained from university sites on the web.

The enhanced pi value and PL/I runtime implementations of trigonometric functions (sin, cos, tan and atan) in float dec(59) arithmetic were tested. These trig functions accept/return input/output in radians. But PL/I also offers trig functions which process values in degrees (sind, cosd, tand and atand). Support for these functions was added to calc, and use of the new values was then tested.

Similarly, the enhanced value for e and the PL/I runtime implementation of log (the natural logarithm of argument X) were tested. Testing noted that PL/I now uses different built-in function names from the names assigned to the calc 1.1 functions. The PL/I names will be used in calc 2.0:

calc 1.1 name	Function	calc 2.0 name(s)
ln(x)	log base e (or natural logarithm) of x	log(x) or ln(x)
log(x)	log base 10 of x	log10(x)
	log base 2 of x	log2(x)

Changes to calc.info describing new/changed features of calc 2.0 are included in the Documentation section below.

Changed Command/AFs: plus, minus, times, divide, quotient, mod, min, max, trunc, floor, round

MTB-1006 describes the main changes to plus.pl1. These involve calling cv_fixed_point_string_ to convert fixed-point input strings to the float dec(59) values always accepted by plus. This enables these command/AFs to accept non-decimal input values that include a radix indicator sub-field.

The only enhancement is addition of a round command/AF to this source program. This is needed to facilitate shortening of possibly-long fractional results that are now provided by float dec(59) arithmetic operations.

For example, to shorten output from the calc command to 5 significant digits:

```
calc "sind(45)"
=   0.70710678118654752438189403651591646848828531801700592041016

r 15:39 0.245 0

round [calc "sind(45)"] 5
0.70711
r 15:40 0.139 0
```

A new round.info block is provided in the Documentation section to describe this new command/AF. It illustrates the extra content added to other command/AFs implemented by plus.pl1 describing support for the fixed-point strings supported by cv_fixed_point_string_.

Changed: bound_multics_bce_.bind

The following cpa output summarizes changes to bound_multics_bce_.bind:

```
cpa [lpn bound_multics_bce_.bind -lb o] == -he
A >ldd>hard>object>bound_multics_bce_.archive::bound_multics_bce_.bind (original)
B >user_dir_dir>Multics>GDixon>w>cvis>bound_multics_bce_.bind (new)

A8        /* HISTORY COMMENTS:
A9          1) change(84-01-01,Loepere), approve(), audit(), install():
A10            Keith Loepere 1/84
A11         2) change(84-08-01,Loepere), approve(), audit(), install():
A12            Modified 8/84 to add display_disk_label_ - ADB
A13         3) change(85-03-01,Loepere), approve(), audit(), install():
A14            Modified 3/85 to remove bce/Multics dual objects, Keith Loepere.
A15         4) change(85-05-01,Lippard), approve(85-01-23,MCR7151),
A16            audit(85-11-07,Spitzer), install(86-02-21,MR12.0-1024):
A17            Modified 5/85 to add reverse_substr, Jim Lippard
A18         5) change(87-07-13,GWMay), approve(87-07-13,MCR7730),
               audit(87-08-10,JRGray),
A19            install(87-09-10,MR12.1-1104):
A20            added definitions for the pipe_ subroutine.
Changed by B to:
B8
B9        /* HISTORY COMMENTS:
B10         1) change(1984-01-01,Loepere), approve(), audit(), install():
B11            Keith Loepere 1/84
B12         2) change(1984-08-01,Loepere), approve(), audit(), install():
B13            Modified 8/84 to add display_disk_label_ - ADB
B14         3) change(1985-03-01,Loepere), approve(), audit(), install():
B15            Modified 3/85 to remove bce/Multics dual objects, Keith Loepere.
B16         4) change(1985-05-01,Lippard), approve(1985-01-23,MCR7151),
B17            audit(1985-11-07,Spitzer), install(1986-02-21,MR12.0-1024):
B18            Modified 5/85 to add reverse_substr, Jim Lippard
B19         5) change(1987-07-13,GWMay), approve(1987-07-13,MCR7730),
B20            audit(1987-08-10,JRGray), install(1987-09-10,MR12.1-1104):
B21            added definitions for the pipe_ subroutine.
B22         6) change(2021-12-05,GDixon):
B23             A) Add and retain the round entry point in the plus object.
B24             B) Add the cv_condition_ object retaining its two entry points.
B25             C) Add decimal_no_round_ object retaining its two entry points.
B26             D) Make numeric_to_ascii_ an entrypoint in numeric_to_ascii_base_.
B27                Retain a new numeric_to_ascii_base_$debug entry point.
B28             E) Move calc into this bound segment.


Inserted in B:
B32
Preceding:
A24       /* Bindfile for bound_multics_bce_, the bce active functions (etc.) mostly


Inserted in B:
B49                           calc,
Preceding:
```

```
A40                         equal,


A43                         pl1_decat_char_,
A44                         move_r_or_t_,
A45                         numeric_to_ascii_,
Changed by B to:
B53                         cv_condition_,
B54                         pl1_decat_char_,
B55
B56                         move_r_or_t_,
B57                         decimal_no_round_,


Inserted in B:
B63       objectname:       calc;
B64        retain:          calc;
B65
Preceding:
A51       objectname:       command_processor_;


Inserted in B:
B83       objectname:       cv_condition_;
B84          retain:        display, message;
B85          delete:        symbol_table;
B86
B87       objectname:       decimal_no_round_;
B88          retain:        divide, multiply;
B89          delete:        symbol_table;
B90
Preceding:
A68       objectname:       display_disk_label_;


A82       objectname:       numeric_to_ascii_;
A83          retain:        numeric_to_ascii_;
A84
A85       objectname:       numeric_to_ascii_base_;
A86          retain:        numeric_to_ascii_base_;
Changed by B to:
B105      objectname:       numeric_to_ascii_base_;
B106         retain:        numeric_to_ascii_base_, numeric_to_ascii_, debug;


A100         synonym:       minus, times, divide, mod, max, min, quotient, ceil,
                            trunc, floor;
A101         retain:        plus, minus, times, divide, mod, max, min, quotient,
                            ceil, trunc, floor;
Changed by B to:
B120         synonym:       minus, times, divide, mod, max, min, quotient, ceil,
                            trunc, floor, round;
B121         retain:        plus, minus, times, divide, quotient, mod, max, min,
                            ceil, trunc, floor, round;


A109       synonym:         after, af, before, be, bool, collate, collate9,
Changed by B to:
B129         synonym:       after, af, before, be, bool, collate, collate9,

Comparison finished: 9 differences, 68 lines.
```

Changed Command/AFs: binary, octal, decimal, hexadecimal, radix

MTB-1006 describes the main changes to hex.pl1. These involve calling cv_fixed_point_string_ to convert fixed-point input strings to the float dec(59) values always accepted by hex.pl1, instead of doing its own input string conversion. The existing code accepted radix indicators: b q o x, plus a special 'u' (for unspec) indicator. By calling cv_fixed_point_string_, the new code will also support indicators: _b _q _o d _d _x rN _rN, where N is a 1- or 2-digit number between 2 and 16.

Because hex.pl1 now calls cv_fixed_point_string_, its result strings will end with a radix indicator sub-field if the result is non-decimal. This change has been added to the Function section of the binary, octal and hexadecimal command/AF info segments. It isn't clear to me whether this change needs to be listed in the Multics System Release Notes.

The existing hex.pl1 code supporting its special 'u' radix indicator value for octal and binary command/AFs is retained in the changed code. If cv_fixed_point_string_ returns error_table_$bad_conversion and the 'u' radix indicator is present and the input string is 8 chars, these characters are assigned to a char(8) string called char_8; if fewer than 8 chars are present in the input string (excluding the 'u' indicator), those chars are assigned to the rightmost character positions of char_8, with unfilled positions set to NUL (\000 or octal 000) characters. Then:

```
unspec(fixed_bin_71) = unspec(char_8);
float_dec_59 = fixed_bin_71;
result = numeric_to_ascii_base_ (float_dec_59, 0, base);
```

Note that numeric_to_ascii_base_ treats any leading zeros as non-significant digits which are ltrim'd from result. The result therefore contains only the binary or octal digits encoding the ASCII characters of the input string. For example:

```
octal abcu
141142143o
```

Support for the 'u' radix indicator has always been present in the binary and octal command/AFs, and is retained by the changed program.

Finally, code supporting a new radix entry point was added to hex.pl1 to support output in any base between 2 and 16. It has the following user interface:

```
2022-05-12  radix

Syntax as a command:  radix BASE VALUEs

Syntax as an active function:  [radix BASE VALUEs]

Function:  returns one or more numeric strings in expressed in digits
of a specified numbering system.  Each non-decimal string is
terminated by a radix indicator sub-field.  See "Notes on result"
below for further details.

Arguments:
BASE
   is the radix of the numbering system digits in which each numeric
   value is to be expressed as a string.  It must be a number between
   2 and 16.
```

```
VALUE
   is an input string to be processed.  It may be a fixed-point or
   floating point decimal string (e.g., 21 or 55.9), but is often a
   numeric string ending with a radix indicator which specifies a
   non-decimal base.  See "List of radix indicators" below.
   . . .
```

Full documentation for radix is shown in the Documentation section of this MCR.

Changed: bound_full_cp_.bind

The following cpa output summarizes changes to bound_full_cp_.bind:

```
cpa [lpn -lb o bound_full_cp_.bind] == -he
A >ldd>sss>object>bound_full_cp_.archive::bound_full_cp_.bind (original)
B >user_dir_dir>Multics>GDixon>w>cvis>bound_full_cp_.bind (new)

Inserted in B:
B87         6) change(2022-06-13,GDixon), approve(2022-06-13,MCR10101):
B88             A) Add the name radix to bound_full_cp_ and retain the new
B89                radix entry point in the hex object segment.
Preceding:
A87                                                     END HISTORY COMMENTS */


Inserted in B:
B139                          radix,
Preceding:
A136                          response,


A176       synonym:           bin, binary, oct, octal, dec, decimal, hexadecimal;
A177       retain:            bin, binary, oct, octal, dec, decimal, hex, hexadecimal;
Changed by B to:
B180       synonym:           bin, binary, oct, octal, dec, decimal, hexadecimal,
radix;
B181       retain:            bin, binary, oct, octal, dec, decimal, hex, hexadecimal,
radix;

Comparison finished: 3 differences, 8 lines.
```

Changed Subroutine: interpret_info_struct_

interpret_info_struc_.pl1 is an internal source in bound_error_handlers_. Its interpret_info_struc_ subroutine is called by the family of default_error_handler_ on-units to display information provided by PL/I runtime when conditions are signaled. For a conversion condition, it is already displaying onsource and onchar pseudo-variables, and the character interpretation of the oncode. This change adds code to put a circumflex character on the line following onsource = …, just below the onchar character causing the conversion condition.

For example:

```
tcfps 10 11 4 12.345e+*2

tcfps ------- 10   in: "12.345e+*2"

Error:  conversion condition by >udd>Multics>GDixon>w>cvis>t>tcfps|1703 (line 191)
onsource = "12.345e+*2", onchar = "*"
                   ^
Invalid character follows a numeric field.
system handler for error returns to command level
```

Changed Subroutine: help_

The small change in help_ to correct the bug described above is shown in the cpa details below. This change does not affect the help_ user interface so no documentation changes are needed.

```
cpa [lpn help_.pl1] == -he
A >ldd>sss>source>bound_help_.s.archive::help_.pl1 (original)
B >user_dir_dir>Multics>GDixon>work>cv_integer_string_>help_.pl1 (new)

Inserted in B:
B61         8) change(2022-06-06,GDixon), approve(2022-06-06,MCR10101):
B62            Change help_ to support several single-entrypoint subroutine info blocks
B63            in a single info segment.
Preceding:
A61                                                          END HISTORY COMMENTS */


Inserted in B:
B587        dcl 1 saved aligned,
B588            2 (blokP, fileP, firstP, lastP, prevP, nextP)  ptr;
B589
Preceding:
A584      PROCESS:


Inserted in B:
B598         saved = null; /*     Nothing stored in "saved" pointers.  Any saving in */
B599                       /*     this loop will be un-"saved" at NEXT_INFO label.   */
B600
Preceding:
A592         hi.info_ptrs = null();
                          /* Setup environment for processing new info block.       */
```

```
A654         call display_block();
                          /*    NOTE: This could be single-entrypoint subroutine    */
A655         go to NEXT_INFO;
                          /*          like match_star_name_$match_star_name_;       */
A656         end;         /*          display_block will decide.                    */
Changed by B to:
B663
B664         if  info_seg_util_$is_Subroutine_kind (iBlok.kind)  then do;
B665                       /*    If this is a single-entrypoint subroutine in a      */
B666                       /*    seg possibly documenting other single-entrypoint    */
B667                       /*    subroutines...                                      */
B668
B669             saved.blokP = iBlokP;
                          /*    temporarily unthread any other blocks.              */
B670             saved.fileP = iFileP;
B671
B672             saved.firstP = iFile.bloks.firstP;
B673             saved.lastP  = iFile.bloks.lastP;
B674             iFile.bloks.firstP, iFile.bloks.lastP = iBlokP;
B675
B676             saved.prevP = iBlok.sib.prevP;
B677             saved.nextP = iBlok.sib.nextP;
B678             iBlok.sib.prevP, iBlok.sib.nextP = null();
B679             end;
B680
B681             call display_block();
                          /*    Display the selected block.                         */
B682             go to NEXT_INFO;
B683             end;


Inserted in B:
B741         if  saved.blokP ^= null()  then do;
                          /* Before processing NEXT_INFO argument...                */
B742                      /*  Redo any "saved" block threads which were unthreaded. */
B743             saved.fileP->iFile.bloks.firstP = saved.firstP;
B744             saved.fileP->iFile.bloks.lastP  = saved.lastP;
B745             saved.blokP->iBlok.sib.prevP = saved.prevP;
B746             saved.blokP->iBlok.sib.nextP = saved.nextP;
B747             saved = null();
                          /*  Clear "saved" pointers for later re-use.              */
B748             end;
B749
Preceding:
A714         end WALK_POINTERS_TO_INFO_BLOCKS;

Comparison finished: 5 differences, 42 lines.
```

Changed Data File: pl1.dcl

Associated with the changed numeric_to_ascii_base_ interface is a small change to the >sss>pl1.dcl file. This file contains PL/I declarations for subroutine entry points that accept a variable number of arguments, or otherwise have entry point argument descriptors (provided by the PL/I compiler) which incorrectly describe their documented user interface.

Both numeric_to_ascii_ and numeric_to_ascii_base_ appear in the pl1.dcl file, and neither meets the criteria described above. Both the existing subroutines and their replacements have correct entry point argument descriptors; the replacement for numeric_to_ascii_base_ will have different argument descriptors because of the change in the function return length (from char(72) varying to char(256) varying). The correct repair is to remove declare lines for both these subroutines from pl1.dcl, as shown in the following cpa output.

```
cpa [lpn pl1.dcl -lb sss.o] == -he
A >ldd>sss>o>pl1.dcl (original)
B >user_dir_dir>Multics>GDixon>work>cv_integer_string_>pl1.dcl (new)

A392    numeric_to_ascii_      entry (float dec(59), fixed bin) returns(char(72)var)
A393    numeric_to_ascii_base_ entry (float dec(59), fixed bin, fixed bin) returns(char(72) var)
Deleted by B, preceding:
B392    random_$exponential           entry (float bin)

Comparison finished: 1 difference, 2 lines.
```

The PL/I-provided entry point descriptors in the revised numeric_to_ascii_base_ will provide the correct syntax to the display_entry_point_dcl (depd) command and get_entry_point_dcl_ subroutine. The emacs ESC-^D (pl1dcl) function calls get_entry_point_dcl_$emacs and therefore will also obtain the correct interface declaration.

Testing

Testing of the changes in this MCR diverged from the planned order of development with the decision that numeric_to_ascii_base_ had to be rewritten. All of the test programs call one or both of these two functions. So these functions had to be successfully corrected and tested before other testing could complete. Test methods and results are therefore described starting with numeric_to_ascii_.

Test: numeric_to_ascii_

The tna.pl1 program tests an individual call to the numeric_to_ascii_ function, and the tna.ec exec_com gathers the several test suites that were performed. These are summarized by a table-of-contents for the exec_com; this summary ends with a brief description of the tna program itself.

```
ec tna toc
Usage: ec tna {TEST_GROUP}
where TEST_GROUP is one of the following labels:
    simple               test small positive/negative integer/fraction combinations.
    zeros                test numbers with zeroes near the decimal point.
    precision, prec      test numbers displayed with selected precisions.
    truncate_round, t_r  test numbers truncated or rounded to selected precision.
    edge                 test smallest and largest numbers.


Usage for tna.pl1 testing command ...

Syntax:  tna NUMERIC_STRING  PRECISION

Function:  Test numeric_to_ascii_.
```

Running the tna exec_com without a TEST_GROUP runs all the test scripts in the order specified by its table-of-contents.

The tna.pl1 program output shows the input numeric string (usually in decimal) and requested precision, a dump of the float dec(59) data created (by cv_fixed_point_string_) from that input string, and two lines showing a numeric string built by numeric_to_ascii_ from that data item.

- The old: line shows the current (unimproved) numeric_to_ascii_ program's output string.
- The new: line shows the replacement (improved) numeric_to_ascii_ program's output string.

The following example shows a test which truncates the input value 123.456e+2 to 4 significant digits.

```
tna 123.456e+2 -4
             in:  123.456e+2  prec: -4
           dump:  +00000000000000000000000000000000000000000000000000123456. E-1
  old: 10   out:  1.234e4
  new: 10   out:  1.234e+4
```

The next example shows the same number being rounded to 4 significant digits of precision.

```
tna 123.456e+2 4
             in:  123.456e+2  prec: 4
           dump:  +00000000000000000000000000000000000000000000000000123456. E-1
  old: 10   out:  1.235e4
  new: 10   out:  1.235e+4
```

The following shows the output of the “edge” test script run by tna.ec.

```
ec tna edge

EDGE CASES:  Test smallest and largest numbers

        - Test the most negative number storable in a float bin(27) variable.

tna -1.701411834604692317e+38 0
             in:  -1.701411834604692317e+38      prec: 0
           dump:  -0000000000000000000000000000000000000001701411834604692317. E19
  old: 10   out:  -17014118346046923172000000000000000000
  new: 10   out:  -1.701411834604692317e+38



        - Test the smallest magnitude number storable in a float dec(59)

tna (+ -)1e-128 0
             in:  +1e-128     prec: 0
           dump:  +00000000000000000000000000000000000000000000000000000000001. E-128
  old: 10   out:  1.e-128
  new: 10   out:  0.1e-127

             in:  -1e-128     prec: 0
           dump:  -00000000000000000000000000000000000000000000000000000000001. E-128
  old: 10   out:  -1.e-128
  new: 10   out:  -0.1e-127



        - Test the largest magnitude number storable in a float dec(59)

tna [cps 9 59]e127 0
             in:  99999999999999999999999999999999999999999999999999999999999e127
                            prec: 0
           dump:  +99999999999999999999999999999999999999999999999999999999999. E127
  old: 10   out:  9.99999999999999999999999999999999999999999999999999999999e185
  new: 10   out:  9.99999999999999999999999999999999999999999999999999999999e+185
```

The last test above uses a copy_string (cps) active function which returns a string consisting of its first argument repeated by the count given in its second argument. For example:

[cps 9 59]

returns a string of 59 “9” digits. This copy_string (cps) active function is used in many parts of the tna.ec test script, and in other test scripts described below. It was developed for this testing and has not been installed. Is there any interest in having it installed?

RESULTS OF TESTING: All the test scripts in the tna.ec completed with the expected results. Some tests demonstrate expected changes in output string between the old and new numeric_to_ascii_ functions. Those differences shown by the tests demonstrate designed-in changes/improvement in the new code.

Test: numeric_to_ascii_base_

The tnab.pl1 program tests an individual call to the numeric_to_ascii_base_ function, and the tnab.ec exec_com gathers the test suites that were performed. These are summarized by a table-of-contents for the exec_com; this summary ends with a brief description of the tnab program. The ? argument is an alternative to toc (for table-of-contents).

```
ec tnab ?

Syntax: ec tnab {TEST_SUITE}

Arguments:
TEST_SUITE

 base_10                  - test all the following base 10 cases...

   simple_10                - simple positive/negative integer/fractions.
   zeros_10                 - numbers with zeroes near the decimal point.
   precision_10, prec_10    - numbers displayed with selected precisions.
   truncate_10, trunc_10    - large exponent numbers w/ precision truncates.
   edge_10                  - smallest/largest numbers.
   frac_10                  - fraction-only numbers near value: .1


 other                    - test all the following non-base-10 cases...

   simple                   - simple positive/negative integers.
   zeros                    - numbers with zeroes near the decimal point.
   precision, prec          - numbers displayed with selected precisions.
   inaccuracy, inacc        - check number accuracy corrections.
   frac                     - fraction-only numbers near value: .1
   errors                   - display error messages that might be returned.


Usage for tnab.pl1 testing command ...

Syntax:  tnab NUMERIC_STRING  PRECISION  BASE

Function:  Test numeric_to_ascii_base_.
```

Because numeric_to_ascii_base_ follows a different algorithm for decimal output versus non-decimal output, there are separate test suites in the exec_com for these two major cases. The decimal output test cases use the numeric_to_ascii_ algorithm to copy decimal digits from the input float dec(59) mantissa into a result.integer string, and copies its exponent into a result.exponent value. It then calls get_result to format the result in Integer, Integer.Fraction, or Integer.FractionEexponent format.

The test cases for decimal output are patterned after those used for the numeric_to_ascii_ function.

```
ec tnab edge_10

EDGE CASES:  Test smallest and largest numbers.

        - Test the most negative number storable in a float bin(27) variable.

tnab -1.701411834604692317e+38 0 10
             in:  -1.701411834604692317e+38      prec: 0
           dump:  -0000000000000000000000000000000000000001701411834604692317e+38. E19
  old: 10   out:  -170141183460469231720000000000000000000
  new: 10   out:  -1.701411834604692317e+38



        - Test the smallest magnitude number storable in a float dec(59)

tnab (+ -)1e-128 0 10
             in:  +1e-128     prec: 0
           dump:  +00000000000000000000000000000000000000000000000000000000001. E-128
  old: 10   out:  0.
  new: 10   out:  0.1e-127

             in:  -1e-128     prec: 0
           dump:  -00000000000000000000000000000000000000000000000000000000001. E-128
  old: 10   out:  -0.
  new: 10   out:  -0.1e-127



        - Test the largest magnitude number storable in a float dec(59)

tnab [cps 9 59]e127 0 10
             in:  99999999999999999999999999999999999999999999999999999999999e127
                            prec: 0
           dump:  +99999999999999999999999999999999999999999999999999999999999. E127
  old: 10   out:  9.99999999999999999999999999999999999999999999999999999999999e+185
  new: 10   out:  9.99999999999999999999999999999999999999999999999999999999999e+185


_____________________________________________________________________________
```

Note the inaccuracy of the old numeric_to_ascii_base_ in handling the +1e-128 and -1e-128 input values. The old program did not special-case decimal output by using the numeric_to_ascii_ algorithm. The inaccuracy of its conversion algorithm therefore shows up in the result string. For decimal output, the new program corrects this inaccuracy by transferring to the numeric_to_ascii_ algorithm to fill the result structure, then calling get_result to format the result string.

Testing the non-decimal output conversions shows differences between old and new implementations of the numeric_to_ascii_base_ function. The biggest difference comes from the new numeric_to_ascii_base_ appending a radix indicator string to all non-decimal output strings. The old program never appended a radix indicator.

The second difference in results comes from the inaccuracies of the old numeric_to_ascii_base_ implementation. When all the problems are corrected, the new implementation non-decimal output strings are much more accurate than those of the old program. The amount/type of inaccuracy varies depending upon the requested output base, as shown in the following output from the tnab.ec.

```
ec tnab inaccuracy

INACCURACY:  Test corrections for suspected inaccurate fractions.


tnab .[cps 0101 4]00001_b 0 2
             in:  .010101010101010100001_b        prec: 0
           dump:  +3333287239074707031250000000000000000000000000000000000000000000000. E-59
  old:  2   out:  0.010101010101010100001
  new:  2   out:  0.010101010101010100001b


--------

tnab .[cps 0121 4]00001_r3 0 3
             in:  .012101210121012100001_r3       prec: 0
           dump:  +19999999544948444127599311619535185976453877491501756128607. E-59
  old:  3   out:  0.012101210121012100000222222222222222222222222222222222222222
  new:  3   out:  0.012101210121012100001_r3


--------

tnab .[cps 0123 4]00001_r4 0 4
             in:  .012301230123012300001_r4       prec: 0
           dump:  +10588235291675118787679821252822875976562500000000000000000. E-59
  old:  4   out:  0.012301230123012300001
  new:  4   out:  0.012301230123012300001q


--------

tnab .[cps 0234 4]00001_r5 0 5
             in:  .023402340234023400001_r5       prec: 0
           dump:  +11057692307620049715200000000000000000000000000000000000000000000. E-59
  old:  5   out:  0.023402340234023400001
  new:  5   out:  0.023402340234023400001_r5


--------

tnab .[cps 02345 3]0001_r6 0 6
             in:  .0234502345023450001_r6         prec: 0
           dump:  +7318327974261126653333144744099134794769893201664578629620 9. E-60
  old:  6   out:  0.023450234502345000055555555555555555555555555555555555555555
  new:  6   out:  0.0234502345023450001_r6


--------

tnab .[cps 06345 3]0001_r7 0 7
             in:  .0634506345063450001_r7         prec: 0
           dump:  +1331667261691973149147828756283763629886733412212981466621 5. E-59
  old:  7   out:  0.063450634506345000066666666666666666666666666666666666666666
  new:  7   out:  0.0634506345063450001_r7


--------
```

```
tnab .[cps 567 3]0001_r8 0 8
            in:  .5675675670001_r8     prec: 0
          dump:  +7338551804441522108390927314758300781250000000000000000000. E-59
  old:  8   out:  0.5675675670001
  new:  8   out:  0.5675675670001o


--------

tnab .[cps 678 3]0001_r9 0 9
            in:  .6786786780001_r9     prec: 0
          dump:  +765109888135401160443726228894412782000599834462152679154 69. E-59
  old:  9   out:  0.67867867800008888888888888888888888888888888888888888888888888
  new:  9   out:  0.6786786780001_r9


--------

tnab .[cps 678 3]0001_r11 0 11
            in:  .6786786780001_r11    prec: 0
          dump:  +609774435831650644257458426828684582141767987629511603042 56. E-59
  old: 11   out:  0.6786786780000aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa98
  new: 11   out:  0.6786786780001_r11


--------

tnab .[cps 789 3]0001_r12 0 12
            in:  .7897897890001_r12    prec: 0
          dump:  +64447017937713334377777830944937130054574888526352564693603. E-59
  old: 12   out:  0.7897897890000bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb7836
  new: 12   out:  0.7897897890001_r12


--------

tnab .[cps 789 3]0001_r13 0 13
            in:  .7897897890001_r13    prec: 0
          dump:  +590163934370580589667942316881573485936930908522388956703 79. E-59
  old: 13   out:  0.78978978900010000000000000000000000000000000000000000044c03b
  new: 13   out:  0.7897897890001_r13


--------

tnab .[cps 789 3]0001_r14 0 14
            in:  .7897897890001_r14    prec: 0
          dump:  +5442945679649077179506226639543182015235008798204785239633 0. E-59
  old: 14   out:  0.78978978900010000000000000000000000000000000000000009c7c2dc
  new: 14   out:  0.7897897890001_r14


--------

tnab .[cps 890 3]0001_r15 0 15
            in:  .8908908900001_r15    prec: 0
          dump:  +5735032602103343044038127782136994824762610838581642490260 8. E-59
  old: 15   out:  0.89089089000010000000000000000000000000000000000004273d7bec
  new: 15   out:  0.8908908900001_r15
```

```
________

tnab .[cps 890 3]0001_r16 0 16
            in:  .8908908900001_r16    prec: 0
          dump:  +535286935279146058874744085187558084726333618164062500000000. E-59
  old: 16   out:  0.8908908900001
  new: 16   out:  0.8908908900001x


_____________________________________________________________________________________
```

RESULTS OF TESTING: All the test scripts in the tnab.ec completed with the expected results. Some tests demonstrate expected changes in output string between the old and new numeric_to_ascii_base_ functions. Those differences shown by the tests demonstrate designed-in changes/improvements in the new code.

Test: cv_integer_string_("" check_), and cv_XXX_("" check_)

The cv_cmp.pl1 test program feeds a given integer string argument and output base to the cv_XXX_ and cv_XXX_check_ functions (where XXX is one of: binary oct dec hex) appropriate to that base, and to the cv_integer_string_ and cv_integer_string_check_ functions which handle any base. For each selected function, an output is shown for both old: and new: implementation of that function. Because there were no inaccuracies in old vs. new implementations, testing focuses on ensuring that equivalent outputs are reported by all functions (for both old and new implementations) for each test; or equivalent error diagnostics are issued. For base 10 output tests, the PL/I runtime convert operator output is also shown for comparison purposes with those test outputs.

Syntax for cv_cmp is:

```
cv_cmp
Syntax as a command:  cv_cmp BASE INTEGER_STRING {ON-UNIT_OPTION}

Arguments:
ON-UNIT_OPTION
   0   Print message and stop conversion. (default)
   1   Print message and set onsource to "--123.5/6" and switch option to 2.
   2   Print message and set onchar to "0".
```

Let's analyze a sample test which demonstrates use of the ON-UNIT_OPTION values. The test integer string contains several errors. Since errors are diagnosed in a different order by the cv_dec_check_ subroutine than by the cv_integer_string_ and cv_integer_string_check_ subroutines, the first error found will differ for these routines. But more important, the old cv_dec_check_ and new version of that subroutine should find the same error, and report the same code value (which indicates index of the bad character within the integer string input value (the onsource value)).

```
cv_cmp 10 "--123.5/6" 0

cv_dec_:                      in: "--123.5/6"  out: 0
cv_dec_check_:                in: "--123.5/6"  code: 2  out: 0

new_cv_dec_:                  in: "--123.5/6"  out: 0
new_cv_dec_check_:            in: "--123.5/6"  code: 2  out: 0

cv_integer_string_:           in: "--123.5/6"  out: 0  conversion condition
  oncode: 720  Invalid argument to cv_integer_string_.
  onsource: "--123.5/6"

cv_integer_string_check_: Error in conversion.     in: "--123.5/6"
  code: error_table_$bad_conversion  out: 0

new_cv_integer_string_:       in: "--123.5/6"  out: 0  conversion condition
  oncode: 403  Invalid character follows a numeric field.
  onsource: "--123.5/6"  onchar: "/"  onchar_index:  8
                    ^
new_cv_integer_string_check_: Error in conversion. in: "--123.5/6"
  code: error_table_$bad_conversion  out: 0

PL/I convert:                 in: "--123.5/6"  out: 0  conversion condition
  oncode: 403  Invalid character follows a numeric field.
  onsource: "--123.5/6"  onchar: "/"  onchar_index:  8
                    ^
```

The test requests decimal conversion of the string "−−123.5/6" with no attempt to fix any errors and retry the conversion. Three errors are present in the input string: the invalid "/" character, presence of the unexpected decimal point "." in an integer string, and presence of an extra minus sign (-) character.

The first two blocks of output compare results from the original (old) cv_dec_ and cv_dec_check_ output with the new versions of those subroutines. Old and new cv_dec_ just return a 0 result, because one of these bad characters caused a conversion condition. Those interfaces suppress the conversion but always return 0 (as documented) if any error is found.

Old and new cv_dec_check_ both find the extra minus sign (−) character and diagnose that incorrect character location in their non-standard return code value (as documented by their interface).

The next two blocks show how the old cv_integer_string_ and cv_integer_string_check_ subroutines handle the errors. The old cv_integer_string_ signals a conversion condition with the entire incorrect input string as its onsource value, and oncode = 720: Invalid argument to cv_integer_string_. The old cv_integer_string_check_ just returns an error_table_$bad_conversion status code, as documented.

The next two blocks show how the new cv_integer_string_ and cv_integer_string_check_ subroutines handle the errors. The new cv_integer_string_ signals a conversion condition with the entire incorrect input string as its onsource value, but with oncode = 403: Invalid character follows a numeric field. And with the onchar pseudo-variable diagnosing the slash (/) as the incorrect character. The new cv_integer_string_check_ just returns an error_table_$bad_conversion status code, as documented.

The final block shows how the PL/I convert built-in function handles the bad integer string. It signals a conversion condition with the same onsource, oncode, and onchar as the new cv_integer_string_ routine.

All of these outputs agree with the design specification, and therefore represent a successful test in which old and new cv_dec_ and cv_dec_check_ results agree, new cv_integer_string_ signals conversion with identical diagnostic pseudo-variables to PL/I convert, and old and new cv_integer_string_check_ both return error_table_$bad_conversion.

All of the testing performed by the cv_cmp program follow this same pattern. Tests with bases other than 2, 8, 10 or 16 exclude the cv_XXX_("" check_) routines where XXX is one of: binary oct dec hex.

The ON-UNIT_OPTION values of 1 and 2 are setup to test retrying a conversion if the caller's condition on-unit changes the onsource or onchar pseudo-variable values. This change affects only the new cv_integer_string_ and PL/I convert built-in outputs, as shown below. new_cv_integer_string_ tries three successive conversion attempts before successful return of a value. The caller's on-unit changes the onsource or onchar strings between attempts as it corrects successive problems found in the integer string.

```
cv_cmp 10 "−−123.5/6" 1

. . .

new_cv_integer_string_:       in: "−−123.5/6"  out: 0  conversion condition
  oncode: 403  Invalid character follows a numeric field.
  onsource: "−−123.5/6"  onchar: "/"  onchar_index:  8
                    ^
```

```
new_cv_integer_string_:       in: "--123.56"  out: 0  conversion condition
  oncode: 404  Too many decimal points in a numeric field.
  onsource: "--123.56"  onchar: "."  onchar_index:  6
                  ^
new_cv_integer_string_:       in: "--123056"  out: 0  conversion condition
  oncode: 403  Invalid character follows a numeric field.
  onsource: "--123056"  onchar: "-"  onchar_index:  2
              ^
new_cv_integer_string_:       in: "-0123056"  out: -123056

. . .

PL/I convert:                 in: "--123.5/6"  out: 0  conversion condition
  oncode: 403  Invalid character follows a numeric field.
  onsource: "--123.5/6"  onchar: "/"  onchar_index:  8
                    ^
PL/I convert:                 in: "--123.56"  out: 0  conversion condition
  oncode: 419  Second half of complex number must be imaginary.
  onsource: "--123.56 "  onchar: " "  onchar_index:  9
                     ^
PL/I convert:                 in: "-123.56"  out: -123
```

The new cv_integer_string_ accepts three retries in which each of the three errors is corrected by the caller's on-unit, return the corrected final value of: "-123056". The on-unit successfully changed either onsource or onchar pseudo-variable to fix each error diagnosed by cv_integer_string_, and retried the conversion until no more errors were found. This is a totally correct set of conversions by the new cv_integer_string_ for this set of input string values.

The PL/I convert built-in diagnoses a different oncode after the first retry because its conversion algorithm supports input strings which are a complex number: <real-number><imaginary-number>I pair. Its oncode diagnoses that the imaginary indicator letter 'i' is missing from the input string. The test program's on-unit returns with the onsource set to "-123.56" which PL/I convert then accepts as a valid complex number (I don't know why the oncode=419 is not repeated), and PL/I returns a final "-123" result.

A tcis.ec exec_com provides test suites which call cv_cmp to test aspects of cv_integer_string_ operation.

```
ec tcis ?
Usage: ec tcis {TEST_GROUP}
where TEST_GROUP is one of the following labels:
    basic              test basic functionality of cv_integer_string_ plus edge cases
    size               test size condition handling
    conversion, conv   test conversion condition handling with several on-unit options
     restart            test several conversion restarts of one call...
                           with on-unit changing input after each attempt

Usage for cv_cmp.pl1 testing command ...

Syntax as a command:  cv_cmp BASE INTEGER_STRING {ON-UNIT_OPTION}

Arguments:
ON-UNIT_OPTION
   0   Print message and stop conversion. (default)
   1   Print message and set onsource to "--123.5/6" and switch option to 2.
   2   Print message and set onchar to "0".
```

RESULTS OF TESTING: All the test scripts in the tcis.ec completed with the expected results.

Test: cv_fixed_point_string_

The tcfps.pl1 test program feeds a default base, a control switch setting and a fixed-point string to the new cv_fixed_point_string_ function. The PL/I runtime convert built-in function provides the only equivalent conversion capabilities, so outputs from the cv_fixed_point_string_ are usually compared with outputs from the PL/I convert.

Syntax of the tcfps test program is shown below.

```
tcfps
Syntax:  tcfps DEFAULT_BASE SWITCHES ON-UNIT_CASE NUMBER_TO_CONVERT
 SWITCHES
  1      Report errors using signaled conditions.
  01     Allow number string with exponent sub-field.
  011    Use PL/I convert built-in for binary number string.
  0101   Use PL/I convert built-in for decimal number string.
  1111   With signals, exponents, convert.
 ON-UNIT_CASE
    0    Status code errors.
    1    Details from cv_condition_$display.
    2    Details from cv_condition_$display then cu_$cl.
    3    Details from cv_condition_$display then return to cv_fixed_point_string_.
    4    Details from default_error_handler_/condition_interpreter.
   1x    Like case 0-4 but don't call PL/I Convert after testing cv_fixed_point_string_.
```

These arguments enable testing of all of the control switch settings of the cv_fixed_point_string_ interface, plus the various caller-provided setups for handling conditions signaled by cv_fixed_point_string_.

The following example shows a typical use of the test program when no errors occur. The command sets base=10 for the default input string base, with switches calling for errors reported via PL/I conditions, and input numbers allowing use of an exponent sub-field. Any conditions that occur are handled by an on-unit setup by the test program: this on-unit calls cv_condition_$display to report data from the condition. The final argument is the input numeric string.

```
tcfps 10 1100 1 123.456e1

tcfps ------- 10   in: "123.456e1"
           dump:  +000000000000000000000000000000000000000000000000123456. E-2
 Sig Exp      10  out:  1234.56

PL/I convert- 10   in: "123.456e1"
           dump:  +000000000000000000000000000000000000000000000000123456. E-2
              10  out:  1234.56
```

First line of the test program output shows the default_base value (10) and the fixed-point string to be tested. The second line shows a dump of the mantissa digits and exponent in the float dec(59) value returned by the function. Third line gives a symbolic interpretation of the input control switch settings, followed by numeric_to_ascii_base_ interpretation of the return value as expressed in the default_base.

The fourth and fifth lines give equivalent output information from the PL/I runtime convert built-in function. For tests in which PL/I convert information is irrelevant, those lines may be suppressed by giving an ON-UNIT_CASE value >= 10.

The tcfps.ec exec_com includes test scripts for typical malformed input strings, edge tests for smallest and largest strings accepted in each number system base, tests for typical fractional fixed-point strings (in which inaccuracies can arise for non-decimal bases), plus tests for the order in which cv_fixed_point_string_ detects errors in a malformed input string, and radix indicator sub-field errors. The following table-of-contents listing shows the set of known test scripts.

```
ec tcfps ?
Usage: ec tcfps {TEST_GROUP}
where TEST_GROUP is one of the following labels:
    signs                       test error processing in mantissa/exponent sign
    magnitude_range, range, mr  test magnitude range boundaries
     mr2                         test base=2  (binary) magnitude boundaries
     mr8                         test base=8  (octal) magnitude boundaries
     mr10                        test base=10 (decimal) magnitude boundaries
     mr16                        test base=16 (hexadecimal) magnitude boundaries
     mr_other                    test other bases (3 to 15) magnitude boundaries
    fractions, frac             test handling of fractional mantissa digits
    oncodes, onc                test all oncodes
     onc401                      no exponent digits
     onc402                      no number digits
     onc403                      invalid chars in number
      onc403_collate              test all chars in collate …
     onc404                      many radix points in mantissa
     onc405                      long input string
     onc407                      many exponents in number
     onc408                      test exponent >  127
     onc409                      test exponent < -128
     onc413                      test more than 10 exponent digits
     onc414                      no mantissa digits
     onc418                      more than 59 significant mantissa digits
    fixed_point_order, order    test order in which cv_fixed_point_string_ finds
                                 errors
    radix_indicator, radix      test radix indicator sub-field for proper operation

Special Tests (used in developing cv_fixed_point_string_):
    PL/I_error_order, pl1_order  test order in which PL/I convert finds errors
    max_power_for_base, power    test largest  base ** POWER  value.
```

If no TEST_GROUP is specified, the exec_com runs all tests except the Special Tests at the bottom of the table-of-contents list. Running all tests generates more than 5000 lines of output and checks all the conversion functionality of cv_fixed_point_string_ across a broad range of input strings.

The Special Tests determine: PL/I convert's order of detecting errors in malformed input strings, and limits of the PL/I runtime decimal_exp2_ operator for the BASE ** POWER calculation used by the cv_fixed_point_string_ algorithm. Data from the pl1_order tests helped determine the order in which the conversion steps are executed by the cv_fixed_point_string_ code so that it diagnoses malformed input strings exactly like PL/I convert. Data from the power tests was stored in the MAX_POWER_FOR_BASE array used by the cv_fixed_point_string_ code to warn the caller when an input string expressed in a given base is too large to be properly converted.

Most tests are annotated with a brief description of what is being tested, and expected results from each test case. Typical annotations are shown in the test output below.

```
ec tcfps onc413
--- oncode = 413: >10 exponent digits --------------------


                            (10 exponent digits OK)
tcfps 10 (1100 1111) 11  1.E+0000000001

tcfps ------- 10   in: "1.E+0000000001"
           dump:  +00000000000000000000000000000000000000000000000000000001. E1
 Sig Exp      10  out:  10

tcfpsConvert- 10   in: "1.E+0000000001"
           dump:  +00000000000000000000000000000000000000000000000000000001. E1
 Sig Exp      10  out:  10

tcfps 10 (1100 1111) 11  1.E0000000001

tcfps ------- 10   in: "1.E0000000001"
           dump:  +00000000000000000000000000000000000000000000000000000001. E1
 Sig Exp      10  out:  10

tcfpsConvert- 10   in: "1.E0000000001"
           dump:  +00000000000000000000000000000000000000000000000000000001. E1
 Sig Exp      10  out:  10


                            (11 or more digits get error)
tcfps 10 (1100 1111) 11  1.E+00000000001

tcfps ------- 10   in: "1.E+00000000001"
 Sig Exp:  conversion condition:  oncode: 413  Too many digits found in exponent or
                                              scale factor of number.
  onsource: "1.E+00000000001"  onchar: "0"  onchar_index:  5
                ^

tcfpsConvert- 10   in: "1.E+00000000001"
 Sig Exp:  conversion condition:  oncode: 413  Too many digits found in exponent or
                                              scale factor of number.
  onsource: "1.E+00000000001"  onchar: "0"  onchar_index:  5
                ^

tcfps 10 (1100 1111) 11  1.E00000000001

tcfps ------- 10   in: "1.E00000000001"
 Sig Exp:  conversion condition:  oncode: 413  Too many digits found in exponent or
                                              scale factor of number.
  onsource: "1.E00000000001"  onchar: "0"  onchar_index:  4
                ^

tcfpsConvert- 10   in: "1.E00000000001"
 Sig Exp:  conversion condition:  oncode: 413  Too many digits found in exponent or
                                              scale factor of number.
  onsource: "1.E00000000001"  onchar: "0"  onchar_index:  4
                ^
```

The tcfps --------- blocks show results when cv_fixed_point_string_ does the conversion itself. The tcfpsConvert- blocks show results when the control switch argument tells cv_fixed_point_string_ to use the PL/I convert built-in function to let PL/I runtime do the actual conversion. Results from the two conversion mechanisms can then be compared.

PL/I convert can handle conversions for both decimal and binary input strings. Conversion of strings in other bases can be performed only using the cv_fixed_point_string_ conversion algorithm.

RESULTS OF TESTING: All the test scripts in the tcfps.ec completed with the expected results. Most of the seemingly inaccurate results seen in early test trials were resolved by fixing the problems in the numeric_to_ascii_base_ subroutine (as described earlier in this MCR).

Test: calc

Changes to the calc command/active function are tested by scripts in the tcalc.ec exec_com. Tests available are summarized below.

```
ec tcalc ?
Usage: ec tcalc {TEST_GROUP}
 where TEST_GROUP is one of the following labels:
    requests      Test calc version (.) and other requests.
    assign        Test calc subsystem use of pre-defined and user-defined variables.
    cmd           Test calc command invoked with a single EXPRESSION argument.
    af            Test calc active function invoked with a single EXPRESSION argument.
    functions     Test calc EXPRESSIONs that use calc functions.
    db            Test calc push-down stack display for debugging.
    errors        Test calc handling of errors and conditions.


Usage for calc command ...


>udd>m>gd>w>cvis>calc.info   (174 lines in info)

2022-04-28  calc

Syntax as a command:  calc {EXPRESSION}

Syntax as an active function:  [calc EXPRESSION]

Arguments:
 EXPRESSION

List of requests:
 <EXPRESSION>              list         q   ..<COMMAND_LINE>   db_off
 <VARIABLE>=<EXPRESSION>   <VARIABLE>   .   db_on

List of functions:
 sin    cos    tan    atan    abs       log10
 sind   cosd   tand   atand   log, ln   log2

List of radix indicators:
 b, _b   q, _q   o, _o   d, _d   x, _x   rN, _rN
```

Because the revised calc program uses cv_fixed_point_string_ to convert input numbers and numeric_to_ascii_ to display expression result numbers, there is no need to focus on testing particular input or output values. Such testing is already embodied in test exec_coms for those subroutines. calc testing therefore focuses on comparing operation of old calc 1.1 with the revised calc 2.0 in the following areas:

- mechanism to report input number errors to the user;
- whether switching from float bin(27) to float dec(59) arithmetic operations affects any of calc's expression parsing and error detection mechanisms;
- verification of new functions added to calc 2.0;
- operation of the new push-down-stack debug display mechanism;
- operation of calc in its three operating modes: one-line command; interactive command, and active function.

Comparisons are made by sending a series of calc requests to the old calc 1.1 program (using float bin(27) arithmetic), then to the calc 2.0 program (using float dec(59) values). The follow script tests interactive requests that assign values to named variables, and requests for listing those variables.

```
ec tcalc assign

   Assignment: Test use of pre-defined and user-defined variables.

oc$calc
.
CALC 1.1
list

e        =   2.718282
pi       =   3.141593

radius=2.0
area_cir = pi * radius**2
area_cir
=    12.56637
list

area_cir =   12.56637
radius   =   2
e        =   2.718282
pi       =   3.141593

q




calc
.
CALC 2.0
list

e        =  2.71828182845904523536028747135266249775724709369995957496 69
pi       =  3.14159265358979323846264338327950288419716939937510582097 49

radius=2.0
area_cir = pi * radius**2
area_cir
=   12.56637061435917295385057353311801153678867759750042328 39

list

area_cir =  12.56637061435917295385057353311801153678867759750042328 39
radius   =  2
e        =  2.71828182845904523536028747135266249775724709369995957496 69
pi       =  3.14159265358979323846264338327950288419716939937510582097 49

q

-----
```

The following script demonstrates how calc uses its push-down stack to process an expression by enabling display of stack operations via the new db_on and db_off requests. Notice how old calc 1.1 treats the db_on and db_off request names as unset variables.

```
ec tcalc db

   Debugging:  use of new db_on and db_off requests

oc$calc
db_on
=    0
10+(2+3*4**2)+5
=    65
db_off
=    0
q



calc
db_on
10+(2+3*4**2)+5

calc(1):  10+(2+3*4**2)+5

 ----- PUSH-DOWN STACK -----
 s( 1):      value: 10.

 ----- PUSH-DOWN STACK -----
 s( 1):      value: 10.
 s( 2):          +        precedence:  2

 ----- PUSH-DOWN STACK -----
 s( 1):      value: 10.
 s( 2):          +        precedence:  2
 s( 3): (    value: 2.

 ----- PUSH-DOWN STACK -----
 s( 1):      value: 10.
 s( 2):          +        precedence:  2
 s( 3): (    value: 2.
 s( 4):          +        precedence:  7

 ----- PUSH-DOWN STACK -----
 s( 1):      value: 10.
 s( 2):          +        precedence:  2
 s( 3): (    value: 2.
 s( 4):          +        precedence:  7
 s( 5):      value: 3.

 ----- PUSH-DOWN STACK -----
 s( 1):      value: 10.
 s( 2):          +        precedence:  2
 s( 3): (    value: 2.
 s( 4):          +        precedence:  7
 s( 5):      value: 3.
 s( 6):          *        precedence:  8
```

```
----- PUSH-DOWN STACK -----
 s( 1):      value: 10.
 s( 2):          +        precedence:  2
 s( 3): (    value: 2.
 s( 4):          +        precedence:  7
 s( 5):      value: 3.
 s( 6):          *        precedence:  8
 s( 7):      value: 4.

 ----- PUSH-DOWN STACK -----
 s( 1):      value: 10.
 s( 2):          +        precedence:  2
 s( 3): (    value: 2.
 s( 4):          +        precedence:  7
 s( 5):      value: 3.
 s( 6):          *        precedence:  8
 s( 7):      value: 4.
 s( 8):          **       precedence:  9

 ----- PUSH-DOWN STACK -----
 s( 1):      value: 10.
 s( 2):          +        precedence:  2
 s( 3): (    value: 2.
 s( 4):          +        precedence:  7
 s( 5):      value: 3.
 s( 6):          *        precedence:  8
 s( 7):      value: 4.
 s( 8):          **       precedence:  9
 s( 9):      value: 2.

 ----- PUSH-DOWN STACK -----
 s( 1):      value: 10.
 s( 2):          +        precedence:  2
 s( 3): (    value: 2.
 s( 4):          +        precedence:  7
 s( 5):      value: 3.
 s( 6):          *        precedence:  8
 s( 7):      value: 4.
 s( 8):          **       precedence:  9
 s( 9):      value: 2.  )
 s(10):          +        precedence:  2

 ----- PUSH-DOWN STACK -----
 s( 1):      value: 10.
 s( 2):          +        precedence:  2
 s( 3): (    value: 2.
 s( 4):          +        precedence:  7
 s( 5):      value: 3.
 s( 6):          *        precedence:  8
 s( 7):      value: 16.  )
 s( 8):          +        precedence:  2

 ----- PUSH-DOWN STACK -----
 s( 1):      value: 10.
 s( 2):          +        precedence:  2
 s( 3): (    value: 2.
 s( 4):          +        precedence:  7
 s( 5):      value: 48.  )
 s( 6):          +        precedence:  2
```

```
----- PUSH-DOWN STACK -----
 s( 1):      value: 10.
 s( 2):          +        precedence:  2
 s( 3):      value: 50.
 s( 4):          +        precedence:  2

 ----- PUSH-DOWN STACK -----
 s( 1):      value: 60.
 s( 2):          +        precedence:  2

 ----- PUSH-DOWN STACK -----
 s( 1):      value: 60.
 s( 2):          +        precedence:  2
 s( 3):      value: 5.
       END of input string

 ----- PUSH-DOWN STACK -----
 s( 1):      value: 65.
   END of input string
=   65

db_off

calc(1):  db_off
q

-----
```

RESULTS OF TESTING: All the test scripts in the tcalc.ec list of scripts completed with the expected results.

Documentation

Some of the changes proposed in MTB-1006 were documented adequately in that MTB. Such documentation will not be repeated here. This includes documentation for the following command/AFs and subroutines: radix_indicator_string_; binary, octal, decimal and hexadecimal; plus, minus, times, divide, quotient, mod, max, min, ceil, trunc, floor; and cv_condition_. Other documentation in MTB-1006 is superseded below with additions and/or corrections. Documentation for the numeric_to_ascii_ subroutine was changed in only minor ways, and will not be shown here.

arithmetic_af.gi.info

Changes to the arithmetic_af.gi.info segment are shown below.

```
cpa >doc>info>arithmetic_af.gi.info ==

A1        03/31/83  arithmetic_af
A2
A3        These commands/active functions perform arithmetic operations:
A4
A5             calc      minus
A6             ceil      mod
A7             divide    plus
A8             floor     quotient
A9             max       times
A10            min       trunc
A11
A12       To obtain information on any of these topics, type:
A13
A14            help TOPIC_NAME
Changed by B to:
B1        :Info: arithmetic_af: arithmetic_af.gi:  2021-12-31  arithmetic_af
B2
B3        The following command/active function programs perform arithmetic
B4        operations:
B5
B6             calc      minus      trunc
B7             ceil      mod
B8             divide    plus
B9             floor     quotient
B10            max       round
B11            min       times
B12
B13
B14       Each calls cv_fixed_point_string_ to convert numeric strings to a
B15       float dec(59) number.  To obtain information on any of these commands,
B16       type:
B17            help COMMAND_NAME
B18       where COMMAND_NAME is one of the names given above.
```

calc

Changes to documentation for the calc command/active function are shown below.

```
cpa >doc>info>calc.info ==

A1        12/07/88  calc
A2
A3        Syntax:  calc {expression}
A4
A5
A6        Syntax as an active function:  [calc expression]
A7
A8
A9        Function:  provides you with a calculator capable of evaluating
A10       arithmetic expressions with operator precedence, a set of often-used
A11       functions, and a memory that is symbolically addressable (i.e., by
A12       identifier).
A13
A14
A15       Arguments:
A16       expression
A17          is an arithmetic expression to be evaluated.  If this argument is
Changed by B to:
B1        :Info: calc:  2022-04-28  calc
B2
B3        Syntax as a command:  calc {EXPRESSION}
B4
B5
B6        Syntax as an active function:  [calc EXPRESSION]
B7
B8
B9        Function: provides a calculator capable of evaluating arithmetic
B10       expressions with operator precedence, a set of often-used functions,
B11       and a memory that is symbolically addressable (i.e., by identifier).
B12
B13       Invoke the command or active function with an EXPRESSION argument to
B14       immediately evaluate that expression.
B15
B16       Invoke the command without arguments to enter the calculator
B17       subsystem.  The user may enter EXPRESSION lines, assign EXPRESSIONs to
B18       VARIABLEs, list VARIABLEs, etc.  Enter a quit (q) request to exit the
B19       subsystem.
B20
B21
B22       Arguments:
B23       EXPRESSION
B24          an arithmetic expression to be evaluated.  If this argument is


A24       List of requests:
A25       <expression>
A26          prints out the value of expression.
A27       <variable>=<expression>
A28          assigns value of expression to variable.
A29       list
A30          prints out the names and values of all the variables that have
A31          been declared so far.
A32       <variable>
A33          prints out the name and value of variable.
```

```
A34
A35
A36       q
A37          returns to command level.
A38       .
A39          causes calc to identify itself by printing "calc".
A40       ..<command_line>
A41          causes the remainder of the line to be passed to Multics as a
A42          command line and executed.
A43
A44
A45       Notes:  Invocation of calc with a newline enters calculator mode.  You
A46       can then type in expressions, assignment statements, or list requests,
A47       separated from each other by one or more newline characters.  All of
A48       these operations are described below.
A49
A50       You must use the quit request with a newline character to return
A51       to command level.
A52
A53
A54       Notes on expressions: Arithmetic expressions involve real values and
A55       the operands +, -, *, /, and ** (addition, subtraction,
A56       multiplication, division, and exponentiation).  A prefix of plus (+)
A57       or minus (-) is allowed.  Parentheses can be used, and blanks between
A58       operators and values are ignored.  Calc evaluates each expression
A59       according to rules of precedence and prints out the result.  The quit
A60       request (followed by a newline character) returns you to command
A61       level.  The order of evaluation is as follows:
Changed by B to:
B31       Notes on expressions:
B32       Arithmetic expressions involve real values and the operands +, -, *,
B33       /, and ** (addition, subtraction, multiplication, division, and
B34       exponentiation).  Unary (or prefix) plus (+) or minus (-) operators
B35       are allowed.  Parentheses can be used, and blanks between operators
B36       and values are ignored.  Calc evaluates each expression according to
B37       rules of precedence and prints out the result.  The order of
B38       evaluation is as follows:


A79
A80       Operations of the same level are processed from left to right except
A81       for the prefix plus and minus, which are processed from right to left.
A82       This means 2**3**4 is evaluated as (2**3)**4.
A83
A84       Numbers can be integers (123), fixed point (1.23) and floating point
A85       (1.23e+2, 1.23e2, 1.23E2, or 1230E-1).  All are stored as float
A86       bin(27).  An accuracy of about seven figures is maintained.  Variables
A87       (see below) can be used in place of constants, e.g., pi * r ** 2.
A88
A89       Seven functions are provided--sin, cos, tan, atan, abs, ln, and log
A90       (ln is base e, log is base 10).  They can be nested to any level, e.g.,
A91       sin(ln(var).5*pi/180).
A92
A93
A94       Notes on assignment statements:  The value of an expression can be
A95       assigned to a variable.  The name of the variable must be from one to
A96       eight characters and must be made up of letters (uppercase and/or
A97       lowercase), digits (if not at the beginning of the name), and the
A98       underscore character (_).  The form is
A99
A100           <variable>=<expression>
```

```
Changed by B to:
B56       Operations sharing the same precedence level are processed from left
B57       to right except for the prefix plus and minus, which are processed
B58       from right to left.  This means 2**3**4 is evaluated as (2**3)**4.
B59
B60
B61       Notes on numbers:
B62       Numbers may be entered as integers (123), or fixed point strings
B63       (1.23).  Numbers may include an E-format exponent (123e5, 1.23e+2,
B64       1.23e2, 1.23E2, or 1230E-1).
B65
B66       Numbers may end with a radix indicator (477o) to enter non-decimal
B67       values in bases 2 through 16.  Base 15 and 16 numbers may not use an
B68       E-format exponent indicator since the "e" or "E" is interpreted as a
B69       digit in those numbering systems; use a "p" or "P" exponent indicator
B70       instead (159p+5x).  See "List of radix indicators" below.
B71
B72       Numbers are stored as float dec(59) values.  A precision of 59 decimal
B73       digits is maintained.  Exponent range: -128 <= EXPONENT <= +127
B74
B75
B76       List of requests:
B77          In the calculator subsystem, the following requests may be used.
B78       <EXPRESSION>
B79          prints out the value of expression.
B80       <VARIABLE>=<EXPRESSION>
B81          assigns value of expression to variable.
B82       list
B83          prints out the names and values of all the variables that have
B84          been declared so far.
B85       <VARIABLE>
B86          prints out the name and value of variable.
B87       q
B88          returns to command level.
B89
B90
B91       .
B92          causes calc to identify itself by printing "calc VERSION" where
B93          VERSION is the current version number of the calc subsystem.
B94       ..<COMMAND_LINE>
B95          causes the remainder of the line to be passed to Multics as a
B96          command line and executed.
B97       db_on
B98          sets an internal static flag enabling display of calc's push-down
B99          stack contents as expressions are evaluated.  Such display persists
B100         for future invocations of calc in the process until db_off is
B101         given.
B102      db_off
B103         disables display of calc's push-down stack contents as expressions
B104         are evaluated.
B105
B106
B107      Notes on assignment statements:
B108      In the calculator subsystem, the value of an expression can be
B109      assigned to a variable.  The form of an assisgnment is:
B110
B111           <VARIABLE> = <EXPRESSION>
B112
B113      The VARIABLE name must begin with a letter, have a total length
B114      of 8 or fewer characters, and must be made up of uppercase and/or
B115      lowercase letters, digits, and the underscore character (_).
```

```
B116


A108
A109      The calc command does not print any response to assignment statements.
A110      The variables "pi" and "e" have preassigned values of 3.14159265 and
A111      2.7182818, respectively.
Changed by B to:
B124      The calc command does not print any response for an assignment
B125      statement.
B126
B127
B128      Notes on variables:
B129      Variables can be used in place of literal numbers.  For example:
B130
B131          r = 32
B132          pi * r ** 2
B133
B134      Preset variables include:
B135          pi:   ratio of a circle's circumference to its diameter; and
B136           e:   Euler's number
B137      The first 59 digits of these irrational values have been stored.
B138
B139
B140      List of functions:
B141         Functions can be nested to any level.  For example:
B142            sin(log(var)+.5*pi/2)
B143      sin
B144         sine(x) where x is in radians.
B145      sind
B146         sine(x) where x is in degrees.
B147      cos
B148         cosine(x) where x is in radians.
B149      cosd
B150         cosine(x) where x is in degrees.
B151      tan
B152         tangent(x) where x is in radians.
B153      tand
B154         tangent(x) where x is in degrees.
B155
B156
B157      atan
B158         arctan(x) in radians where: -pi/2 < arctan(x) < pi/2
B159      atand
B160         arctan(x) in degrees where: -90 < arctan(x) < 90
B161      abs
B162         if x > 0, result is x; otherwise, result is -x.
B163      log, ln
B164         log base e of x, also called ln(x) or the natural logarithm of x.
B165      log10
B166         log base 10 of x.
B167      log2
B168         log base 2 of x.
B169
B170
B171      List of radix indicators:
B172         Indicator characters may also be given in uppercase.
B173      b, _b
B174         the number is interpreted as a base two number (binary).
B175         See "Notes" below.
```

```
B176      q, _q
B177         the number is interpreted as a base four number (quaternary).
B178      o, _o
B179         the number is interpreted as a base eight number (octal).
B180      d, _d
B181         the number is interpreted as a base ten number (decimal).
B182         See "Notes" below.
B183      x, _x
B184         the number is interpreted as a base sixteen number (hexadecimal).
B185
B186
B187      rN, _rN
B188         the number is interpreted in the base N which is a decimal number
B189         between 2 and 16 inclusive
```

cv_dec_

The following comparison shows changes to the cv_dec_ subroutine. Similar documentation changes for cv_dec_check_, cv_hex_, cv_hex_check_, cv_oct_, cv_oct_check_ will be installed, but are not shown in this MCR.

```
cpa >doc>info>cv_dec_.info ==

A8        :Entry: cv_dec_: 02/06/84  cv_dec_
A9
A10       Function: accepts an ASCII representation of a decimal integer and
A11       returns the fixed binary(35) representation of that number.  (See also
A12       cv_dec_check_).
Changed by B to:
B179      :Info: cv_dec_: 2021-10-28  cv_dec_
B180
B181      Function: accepts an ASCII representation of a decimal integer, or an
B182      integer with an optional trailing radix indicator.  It returns the
B183      fixed binary(35) representation of that number.  This entry point
B184      returns 0 if the convert operation fails.
B185
B186      See also cv_dec_check_ for a routine that returns a code if the
B187      convert operation fails.


A17       a = cv_dec_ (string);
A18
A19
A20       Arguments:
A21       string
A22          is the string to be converted.  (Input)
A23       a
A24          is the result of the conversion.  (Output)
A25
A26
A27       Notes:  If string is not a proper character representation of a decimal
A28       number, a will contain the converted value of the string up to, but not
A29       including, the incorrect character within the string.
A30
A31       This function can be performed more efficiently in PL/I by--
A32
A33             a = convert (a, string);
Changed by B to:
B192      result = cv_dec_ (string);
B193
B194
B195      Arguments:
B196      result
B197         is the converted fixed bin(35) value.  (Output)
B198      string
B199         is the string to be converted.  The number may be positive or
B200         negative, and may have a trailing radix indicator as its last
B201         character(s).  Space and/or horizontal tab characters around number
B202         or between sign and the number are ignored.  (Input)
B203
B204
B205      List of radix indicators:
B206         Indicator characters may also be given in uppercase.
B207      b, _b
B208         the number is interpreted as a base two number (binary).
```

```
B209          See "Notes" below.
B210       q, _q
B211          the number is interpreted as a base four number (quaternary).
B212       o, _o
B213          the number is interpreted as a base eight number (octal).
B214       d, _d
B215          the number is interpreted as a base ten number (decimal).
B216          See "Notes" below.
B217       x, _x
B218          the number is interpreted as a base sixteen number (hexadecimal).
B219
B220
B221       rN, _rN
B222          the number is interpreted in the base N which is a decimal number
B223          between 2 and 16 inclusive.
B224
B225
B226       Notes on input integers:
B227       A radix indicator may be appended to any input integer to specify its
B228       numeric base.  Decimal is assumed if no radix indicator is given.
B229
B230          INDICATOR   NUMERIC BASE              EXAMPLES
B231           b or _b      base  2 or binary        1101b
B232           q or _q      base  4 or quaternary    113q  (=23d)
B233           o or _o      base  8 or octal         21o
B234           d or _d      base 10 or decimal       17d or 17
B235           x or _x      base 16 or hexadecimal   11x
B236          rN or _rN     where N is base value
B237                          between 2 and 16        44r5  (=24d)
B238
B239       A negative value may be given using a leading minus (-) character:
B240          -23
B241
B242
B243       For octal input, a negative value may also be given using a 12-digit
B244       octal format in which the left-most digit is 5, 6, or 7:
B245          -1o is same as 777777777777o
B246
B247       Digits for bases 11 through 16 may be entered using letters for digit
B248       values beyond 9:
B249          0 1 2 3 4 5 6 7 8 9 A B C D E F
B250       with letters accepted in either uppercase or lowercase.
B251
B252       For hexadecimal input, a negative value may also be given using a
B253       9-digit hex format in which the left-most digit is: 8, 9, A, B, C, D,
B254       E, or F:
B255          -2x is same as FFFFFFFFEx
B256
B257
B258       Notes: If string is not a proper character representation of an
B259       integer, result will contain the converted value of the string, up to
B260       but not including the incorrect character within the string.  If the
B261       radix indicator or the default base is not valid, result will contain
B262       0.
B263
B264       Conversion of an integer without radix indicator can be performed more
B265       efficiently in PL/I by--
B266
B267             result = convert (result, string);
```

cv_integer_string_

The following comparison shows changes to the cv_integer_string_ subroutine. Similar documentation changes for cv_integer_string_check_ will be installed. For brevity, these are not shown in this MCR.

cpa >doc>info>cv_integer_string_.info ==

```
A12       :Entry: cv_integer_string_: 03/12/92  cv_integer_string_
A13
A14       Function: accepts an ASCII representation of an integer with a trailing
A15       radix indicator and returns the fixed binary(35) representation of that
A16       number.  (See also cv_integer_string_check_).
Changed by B to:
B557      :Info: cv_integer_string_: 2021-10-27  cv_integer_string_
B558
B559      Function: accepts an ASCII representation of an integer with an
B560      optional trailing radix indicator and returns the fixed binary(35)
B561      representation of that number.  This entry point signals a conversion
B562      or size condition if the convert operation fails.
B563
B564      See also cv_integer_string_check_ for a routine that returns a
B565      status code if the convert operation fails.

A22       a = cv_integer_string_ (string, default_base);
A23
A24
A25       Arguments:
A26       string
A27          is the string to be converted.  The number may be positive or
A28          negative, and may have a trailing radix indicator as its last
A29          character.  (Input)
Changed by B to:
B571      result = cv_integer_string_ (string, default_base);
B572
B573
B574      Arguments:
B575      result
B576         is the converted fixed bin(35) value.  (Output)
B577      string
B578         is the string to be converted.  The number may be positive or
B579         negative, and may have a trailing radix indicator as its last
B580         character(s).  Space and/or horizontal tab characters around number
B581         or between sign and the number are ignored.  (Input)
B582      default_base
B583         is the base to be used if the input string does not have a trailing
B584         radix indicator.  This base may be any integer between 2 to 16
B585         inclusive. (Input)
B586
B587
B588      List of radix indicators:
B589         Indicator characters may also be given in uppercase.
B590      b, _b
B591         the number is interpreted as a base two number (binary).
B592         See "Notes" below.
B593      q, _q
B594         the number is interpreted as a base four number (quaternary).
B595      o, _o
B596         the number is interpreted as a base eight number (octal).
```

```
B597      d, _d
B598         the number is interpreted as a base ten number (decimal).
B599         See "Notes" below.

B600      x, _x
B601         the number is interpreted as a base sixteen number (hexadecimal).
B602
B603
B604      rN, _rN
B605         the number is interpreted in the base N which is a decimal number
B606         between 2 and 16 inclusive.
B607
B608
B609      Notes on input integers:
B610      A radix indicator may be appended to any input integer to specify its
B611      numeric base.  Decimal is assumed if no radix indicator is given.
B612
B613         INDICATOR   NUMERIC BASE              EXAMPLES
B614          b or _b      base  2 or binary        1101b
B615          q or _q      base  4 or quaternary    113q  (=23d)
B616          o or _o      base  8 or octal         21o
B617          d or _d      base 10 or decimal       17d or 17
B618          x or _x      base 16 or hexadecimal   11x
B619         rN or _rN     where N is base value
B620                         between 2 and 16        44r5  (=24d)
B621
B622      A negative value may be given using a leading minus (-) character:
B623         -23
B624
B625
B626      For octal input, a negative value may also be given using a 12-digit
B627      octal format in which the left-most digit is 5, 6, or 7:
B628         -1o is same as 777777777777o
B629
B630      Digits for bases 11 through 16 may be entered using letters for digit
B631      values beyond 9:
B632         0 1 2 3 4 5 6 7 8 9 A B C D E F
B633      with letters accepted in either uppercase or lowercase.
B634
B635      For hexadecimal input, a negative value may also be given using a
B636      9-digit hex format in which the left-most digit is: 8, 9, A, B, C, D,
B637      E, or F:
B638         -2x is same as FFFFFFFFEx
B639
B640
B641      Notes: If string is not a proper character representation of an
B642      integer, result will contain the converted value of the string, up to
B643      but not including the incorrect character within the string.  If the
B644      radix indicator or the default base is not valid result will contain
B645      0.
B646
B647      If the default base is larger than 11, a final "b" is treated as a
B648      digit in the number rather than as a radix indicator.  Similarly, if
B649      the default base is larger than 13, a final "d" is treated as a digit
B650      in the number rather than as a radix indicator.  Use "_b" or "_d" as
B651      the radix indicator for a default base larger than 11.
```

cv_fixed_point_string_

Documentation for the new cv_fixed_point_string_ subroutine is shown below.

```
Entry:  2022-05-06  cv_fixed_point_string_

Function: accepts an ASCII representation of a fixed-point number with
optional exponent and radix indicator sub-fields.  It returns the
float decimal(59) representation of that number.

Syntax:
declare cv_fixed_point_string_ entry (char(*), fixed bin, bit(*),
   fixed bin(35))   returns(float dec(59));
result = cv_fixed_point_string_ (fixed_point_string, default_base,
   switches, code);

Arguments:
result
   a float decimal(59) result produced by successfully converting
   the fixed_point_string to a number. (Output)
fixed_point_string
   an ASCII representation of a fixed-point number ending with
   optional exponent and radix indicator sub-fields.  See the "Notes
   on input strings" and "List of radix indicators" sections.  If
   expressed as a decimal number, the string may include up to 59
   significant digits.  If expressed in another numeric base or radix,
   more digits may be given if base < 10; and fewer if base > 10.
   (Input)

default_base
   is the base in which the fixed_point_string is expressed if no
   radix indicator ends that string.  It must be a number between 2
   and 16 inclusive.  (Input)
switches
   control the action of this subroutine.  See the "Notes on switches"
   section.  (Input)

code
   standard status code indicating success of the convert operation.
   Failure values may be one of the following.
    error_table_$bad_arg
       default_base was not in the range:  2 <= default_base <= 16
    error_table_$bad_conversion
       an unexpected sign, digit, radix point or exponent was found in
       fixed_point_string; or more digits were given than will fit in
       the precision provided by a float dec(59) data type.
    error_table_$bigarg
       fixed_point_string exceeds implementation restriction of 256
       characters in length.

    error_table_$item_too_big
       an exponent overflow condition occurred when converting
       fixed_point_string to a float dec(59) value.
    error_table_$smallarg
       an exponent underflow condition occurred when converting
       fixed_point_string to a float dec(59) value.
```

```
Notes on switches:
The switches argument is assigned to a structure whose elements
ascribe meaning to the individual bits in switches.

   string(fixed_point_switches) = switches;

 dcl 1 fixed_point_switches based,
       2 signal_errors                 bit (1) unaligned,
       2 allow_exponent                bit (1) unaligned,
       2 convert_binary                bit (1) unaligned,
       2 convert_decimal               bit (1) unaligned;

This structure is declared in cv_fixed_point_string_.incl.pl1.

The structure elements are described in the "List of elements" section.
See the "List of named constants" section for named constants setting
bit values or combinations.

List of elements:
signal_errors
   "1"b: errors encountered during convert operation are signaled as
         PL/I conditions.  See the "Notes on condition handling"
         section.
   "0"b: errors are returned as status code values.

allow_exponent
   "1"b: fixed_point_string may include an exponent sub-field.
         See the "Notes on input strings" section.
   "0"b: an exponent sub-field causes a conversion condition or status
         code.

convert_binary
   "1"b: use PL/I convert built-in to convert a fixed_point_string
         expressed as binary (base 2) digits.  See the "Notes on
         convert built-in" section.
   "0"b: use cv_fixed_point_string_ algorithms for a binary string.

convert_decimal
   "1"b: use PL/I convert built-in to convert a fixed_point_string
         expressed as decimal (base 10) digits.  See the "Notes on
         convert built-in" section.
   "0"b: use cv_fixed_point_string_ algorithms for a decimal string.

List of named constants:
   The following constants may be OR'd together or used individually
   as a value for the switches argument.
FIXED_POINT_SIGNALS
   convert operation errors are signaled as PL/I conditions.
FIXED_POINT_EXPONENT
   fixed_point_string may include an exponent sub-field.
FIXED_POINT_CONVERT_BIN
   binary strings are processed using the PL/I convert built-in.
FIXED_POINT_CONVERT_DEC
   decimal strings are processed using the PL/I convert built-in.

FIXED_POINT_EXPONENT_CONVERT_DEC
   turns on allow_exponents and convert_decimal.
FIXED_POINT_EXPONENT_CONVERT
   turns on allow_exponents, convert_binary and convert_decimal.
```

```
FIXED_POINT_SIG_EXP
   turns on signal_errors and allow_exponents.
FIXED_POINT_SIG_EXP_CONVERT_DEC
   turns on signal_errors, allow_exponents and convert_decimal.
FIXED_POINT_SIG_EXP_CONVERT
   turns on all four of the bits above.

List of radix indicators:
   A radix indicator sub-field may appear at the end of a
   fixed_point_string to indicate the base in which its digits are
   expressed.  Indicator characters may also be given in uppercase.
b, _b
   digits are interpreted as a base two number (binary).
q, _q
   digits are interpreted as a base four number (quaternary).
o, _o
   digits are interpreted as a base eight number (octal).
d, _d
   digits are interpreted as a base ten number (decimal).
x, _x
   digits are interpreted as a base sixteen number (hexadecimal).

rN, _rN
   digits are interpreted in the base N which is a decimal number
   where:  2 <= N <= 16

Notes on input strings:
A fixed-point string is an ASCII representation of a float dec(59)
numeric data type.  The string consists of three sub-fields:

   +123456700.0034E+02o
   \_ mantissa__ /\_/ |
                 |  |
         exponent -'  '- radix indicator

The fixed_point_string may begin with an optional number of space (SP)
characters.  The fixed-point string continues with a mantissa.

By default, the fixed_point_string is assumed to be positive.  A
leading plus (+) is allowed.  A negative value is specified using a
leading minus (-) character:
   -23

The fixed_point_string continues with a sequence of digits, and may
include one radix point (.) character separating integer digits
preceding the point from fractional digits following the point.  This
optionally signed sequence of digits with optional radix point is
sometimes called the "mantissa" or "significand":
   -23.004

Digits for numbering system bases 11 through 16 may be entered using
letters for digit values beyond 9, with letters accepted in either
uppercase or lowercase.  Digits for the most common numeric bases are
shown in the table below, ordered in ascending value.  Digits for a
base N lower than 16 use the first N valid digits listed for base 16.
```

```
  BASE      VALID DIGITS IN THAT BASE
  ----   -------------------------------
    2    0 1
    8    0 1 2 3 4 5 6 7
   10    0 1 2 3 4 5 6 7 8 9
   12    0 1 2 3 5 5 6 7 8 9 A B
   16    0 1 2 3 4 5 6 7 8 9 A B C D E F
```

If fixed_point_switches.allow_exponent is "1"b, the fixed_point_string begins with a mantissa which may be followed by an exponent sub-field. The exponent begins with one of the exponent indicator letters: E F P (in either uppercase or lowercase). The exponent digits are an optionally-signed decimal number EXP. Each fixed point string is expressed in one of the following formats:

```
   MANTISSA
   MANTISSAeEXP   or   MANTISSAe+EXP   or   MANTISSAe-EXP
   MANTISSAfEXP   or   MANTISSAf+EXP   or   MANTISSAf-EXP
   MANTISSApEXP   or   MANTISSAp+EXP   or   MANTISSAp-EXP
```

While MANTISSA gives the integer or integer.fraction representation expressed in digits of the selected numbering system, the EXP exponent is always expressed using decimal digits.

An exponent for a base 15 string must begin with one of the exponent indicator letters: F P (in either uppercase or lowercase) because 'E' is treated as a mantissa digit in a base 15 number.

An exponent for a base 16 string must begin with exponent indicator letter: P (in either uppercase or lowercase) because 'E' and 'F" are treated as a mantissa digit in a base 16 number. However, note that the PL/I convert built-in interprets only binary and decimal numeric strings and therefore has not been modified to accept P as an exponent indicator letter.

No whitespace may appear between sign and digits of the exponent, or between mantissa and the exponent.

A positive exponent shifts the radix point right EXP places; a negative exponent shifts the radix point left EXP places. The following table shows decimal numbers with an exponent. Numbers expressed in another base would use that base to the power EXP as the multiplier.

```
   STRING      mantissa * BASE**EXP   RESULT
   ---------   --------------------   -----------
    123E-4      123     * 10**-4          0.0123
    1.23E+1       1.23  * 10**+1         12.3
    1.23E5_d      1.23  * 10**+5      123000
    .123E+2d       .123 * 10**+2         12.3
```

A radix indicator sub-field may end the fixed_point_string to specify its numeric base. The default_base specifies the base to assume if no radix indicator is given.

The indicator sub-field can begin with underscore (_) which separates the radix indicator letter that follows from digits of a base 12 (or higher) mantissa; the underscore prevents a "b" or "d" radix indicator from being treated as a digit of the mantissa.

The following table shows strings expressed in a variety of numbering bases, along with the decimal value of the float dec(59) number represented by that string.

INDICATOR	NUMERIC BASE	EXAMPLES	VALUE (base 10)
b or _b	base 2 or binary	10001.11b	17.75
q or _q	base 4 or quaternary	113_q	23
o or _o	base 8 or octal	21.6o	17.75
d or _d	base 10 or decimal	17.75d	17.75
x or _x	base 16 or hexadecimal	11.Cx	17.75
rN or _rN	N is a decimal base number: 2 <= N <= 16	43r5	23
		15.9_r12	17.75

The fixed_point_string may end with one or more space (SP) characters. These characters are ignored by the conversion.

The greatest magnitude for a data value stored as float decimal(59) real is 10**186. The smallest magnitude of such data value is 10**-128.

The range of values supported in the fixed_point_string argument varies according to the numeric base in which the mantissa digits are expressed. The following table summarizes the range of fixed-point string magnitudes accepted for each numbering base.

BASE	MIN MAGNITUDE	MAXIMUM MAGNITUDE	
2	0.1E-228b	1.1E+617b	
3	0.1E-143_r3	2.1E+389_r3	
4	0.1E-113q	3.2E+308q	
5	0.1E-97_r5	1.0E+266_r5	
6	0.1E-87_r6	1.0E+239_r6	
7	0.1E-80_r7	1.1E+220_r7	
8	0.1E-75o	7.2E+205o	
9	0.1E-71_r9	7.4E+194_r9	
10	0.1E-127	9.9...9E+185	(59 significant digits)
11	0.aE-66_r11	4.3E+178_r11	
12	0.aE-63_r12	2.4E+172_r12	
13	0.aE-61_r13	c.2E+166_r13	
14	0.aE-60_r14	2.1E+162_r14	
15	0.aP-58_r15	1.7P+158_r15	
16	0.aP-57x	3.aP+154x	

Accuracy of the conversion algorithm depends upon the selected base. A float dec(59) arithmetic may have insufficient precision to represent digits of a non-decimal input string as digits in the float dec(59) target value. The maximum value expressible in a decimal fixed_point_string argument is therefore larger than magnitudes expressed as an octal, binary, or other radix representation of a number.

Notes on condition handling:
If fixed_point_switches.signal_errors = "1"b, the following PL/1 conditions are signaled if an error is encountered during the conversion.

CONDITION	EVENT TRIGGERING THE CONDITION
conversion	- whitespace characters other than space (" "). - space characters anywhere in the string other than the very beginning or end of string. - more than one sign character (+ or -) in the mantissa or exponent. - an invalid character in the mantissa. - more significant digits in the mantissa than will fit in a float dec(59) data type. - using an exponent sub-field when not allowed. - a non-decimal number in an exponent sub-field. - an invalid radix indicator string.

CONDITION	EVENT TRIGGERING THE CONDITION
overflow	- a string has significant digits in the mantissa plus an exponent sub-field causing the result exponent to be > +127.
underflow	- a string has significant digits in the mantissa plus an exponent sub-field causing the result exponent to be < -128.

Significant digits in the mantissa are those that follow any leading zero (0) digits. Leading zeroes are ignored. The table below shows how many significant digits can be converted to a float dec(59) data type for each numeric base. A mantissa with more significant digits than this number will trigger a conversion condition.

BASE	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
MAX_DIGITS_ALLOWED	199	125	99	85	77	70	66	62	59	57	55	53	52	51	49

Just before the underflow or overflow condition is signaled, the current value of the oncode built-in function is pushed down and a fixed bin(35,0) code giving the reason for the signaling the condition is assigned to "oncode". The underflow and overflow conditions may not be restarted.

Just before the conversion condition is signaled, the current values of the oncode, onsource and onchar built-in functions are pushed down. A fixed bin(35,0) code giving the reason for the signaling conversion is assigned to "oncode". The string being converted is assigned to "onsource". The leftmost character for which the conversion failed is assigned to "onchar".

The caller's on-unit for any of these conditions may also call the cv_condition_$message subroutine to get additional information about that condition; or call cv_condition_$display to display such information to the user. For details, type: help cv_condition_

The conversion on-unit may perform a non-local goto to a label in the calling program to abort the failed convert operation. Or the convert operation may be restarted by having the on-unit use the onchar pseudo-variable to change the value of the leftmost failing character; or use the onsource pseudo-variable to change the entire contents of the string being converted. If the on-unit then ends without calling the continue_to_signal_ subroutine, the convert operation is retried using any replaced string values.

Notes on convert built-in:
If fixed_point_switches.convert_binary and/or .convert_decimal
switches are "1"b, cv_fixed_point_string_ uses the PL/I convert
built-in function to convert a binary and/or decimal string to a
float dec(59) data type.

The PL/I convert built-in function provides program access to PL/I
runtime operators that convert character strings to arithmetic data
types. These convert operators accept numeric strings expressed in
binary or decimal digits. Binary strings must end with a "b" radix
indicator. "b" is the only radix indicator accepted by the PL/I
convert operators. cv_fixed_point_string_ will add a "b" radix
indicator to an input string with base selected by default_base = 2,
or selected by another radix indicator form: _b or _r2 or r2

The PL/I convert built-in is more efficient than the
cv_fixed_point_string_ subroutine, and accepts binary and decimal
strings in a wider variety of styles than cv_fixed_point_string_.
This includes support for complex number strings including both real
and imaginary parts (e.g., 123.4+32.0i). The convert built-in may
employ conversion techniques that generate fewer exponent overflow or
underflow conditions than the cv_fixed_point_string_ algorithms.

Support for a wider variety of input styles comes with programming
costs. The program must be prepared to: explain accepted input styles
in its documentation; and handle additional conversion errors that can
occur when converting these added input styles. Also note that the
convert built-in does not accept P as an exponent indicator letter,
but cv_fixed_point_string_ changes any use of P indicator to E before
invoking the convert built-in. These issues are avoided if the PL/I
convert built-in bits in switches are off.

decimal_no_round_

Documentation for the new decimal_no_round_$divide and decimal_no_round_$multiply subroutines is shown below.

```
2022-04-08  decimal_no_round_

Function:  performs decimal multiply and divide operations without
rounding.

Entry points in decimal_no_round_:
 2022-04-08  decimal_no_round_$divide
 2022-04-08  decimal_no_round_$multiply


Entry:  decimal_no_round_$divide   (30 lines in entry point; 1 other entry point in
info)

2022-04-08  decimal_no_round_$divide

Function: Performs a decimal division stopping the operation when the
required precision of the target has been reached.  The normal PL/I
divide operation extends the quotient by two decimal places, then
rounds the target quotient using values in these final two digits.
This subroutine assigns the unextended quotient value without
rounding.

Syntax:
dcl decimal_no_round_$divide entry (float dec(59), float dec(59),
   bit(18) aligned) returns (float dec(59));
quotient =  decimal_no_round_$divide (dividend, divisor,
   indicator_bits);

Arguments:
dividend
   number to be divided. (Input)
divisor
   number dividing the dividend. (Input)
indicator_bits
   indicator bits set by the division (DV3D) instruction.  See
   mc.incl.pl1 declaration of scu.ir substructure for order of bits.
   See Multics Processor Manual (AL39) description of "Indicator
   Register (IR)" (page 3-5) for meanings of each bit.  The
   indicator_register,dl (right half-word) of this descriptor are the
   bits returned in the indicators value. (Output)
quotient
   division result without any extension of precision to permit
   rounding. (Output)


Entry:  decimal_no_round_$multiply   (26 lines in entry point; 1 other entry point in
info)

2022-04-08  decimal_no_round_$multiply

Function: Performs a decimal multiply without rounding of the product.
The normal PL/I multiply operation rounds the product's final decimal
place if it exceeds precision of the mantissa.
```

```
Syntax:
dcl decimal_no_round_$multiply entry (float dec(59), float dec(59),
   bit(18) aligned) returns (float dec(59));
quotient =  decimal_no_round_$multiply (multiplicand, multiplier,
   indicator_bits);

Arguments:
multiplicand
   number to be multiplied. (Input)
multiplier
   number multiplying the multiplicand. (Input)
indicator_bits
   indicator bits set by the multiplication (MP3D) instruction.  See
   mc.incl.pl1 declaration of scu.ir substructure for order of bits.
   See Multics Processor Manual (AL39) description of "Indicator
   Register (IR)" (page 3-5) for meanings of each bit.  The
   indicator_register,dl (right half-word) of this descriptor are the
   bits returned in the indicators value. (Output)
product
   multiply operation result without any rounding. (Output)
```

fixed_point_string.gi.info

A new fixed_point_string.gi.info block will be added to describe the general properties of a fixed-point string.

```
2022-05-19  fixed-point string

A fixed-point string is an ASCII representation of a float dec(59)
numeric data type.  The string consists of three sub-fields:

   +123456700.0034E+02o
   \_ mantissa__ /\_/ |
                  |  |
         exponent -'  '- radix indicator

The fixed-point string may begin with an optional number of space (SP)
characters.  The fixed-point string continues with a mantissa.

By default, the fixed point string is assumed to be positive.  A
leading plus (+) is allowed.  A negative value is specified using a
leading minus (-) character:
   -23

The fixed-point string continues with a sequence of digits, and may
include one radix point (.) character separating integer digits
preceding the point from fractional digits following the point.  This
optionally signed sequence of digits with optional radix point is
the "mantissa" or "significand":
   -23.004

Digits for numbering system bases 11 through 16 may be entered using
letters for digit values beyond 9, with letters accepted in either
uppercase or lowercase.  Digits for the most common numeric bases are
shown in the table below, ordered in ascending digit value.  Digits
for a base N lower than 16 use the first N valid digits listed for
base 16.

  BASE      VALID DIGITS IN THAT BASE
  ----   -------------------------------
    2    0 1
    8    0 1 2 3 4 5 6 7
   10    0 1 2 3 4 5 6 7 8 9
   12    0 1 2 3 5 5 6 7 8 9 A B
   16    0 1 2 3 4 5 6 7 8 9 A B C D E F

The mantissa may be followed by an exponent sub-field.  The exponent
begins with one of the exponent indicator letters: E F P (in either
uppercase or lowercase).  The exponent digits are an optionally-signed
decimal number EXP.  Each fixed point string is expressed in one of
the following formats:

   MANTISSA
   MANTISSAeEXP   or   MANTISSAe+EXP   or   MANTISSAe-EXP
   MANTISSAfEXP   or   MANTISSAf+EXP   or   MANTISSAf-EXP
   MANTISSApEXP   or   MANTISSAp+EXP   or   MANTISSAp-EXP

While MANTISSA gives the integer or integer.fraction representation
expressed in digits of the selected numbering system, the EXP exponent
is always expressed using decimal digits.
```

An exponent for a base 15 string must begin with one of the exponent indicator letters: F P (in either uppercase or lowercase) because 'E' is treated as a mantissa digit in a base 15 number.

An exponent for a base 16 string must begin with exponent indicator letter: P (in either uppercase or lowercase) because 'E' and 'F" are treated as a mantissa digit in a base 16 number. However, note that the PL/I convert built-in interprets only binary and decimal numeric strings and therefore has not been modified to accept P as an exponent indicator letter. The cv_fixed_point_string_ subroutine accepts exponent letters: E F P

No whitespace may appear between sign and digits of the exponent, or between mantissa and the exponent sub-fields.

A positive EXP value shifts the radix point right EXP places; a negative EXP shifts the radix point left EXP places. The following table shows decimal numbers with an exponent. Numbers expressed in another base would use that base to the power EXP as the multiplier.

STRING	mantissa * BASE**EXP	RESULT
123E-4	123 * 10**-4	0.0123
1.23E+1	1.23 * 10**+1	12.3
1.23E5_d	1.23 * 10**+5	123000
.123E+2d	.123 * 10**+2	12.3

A radix indicator sub-field may end the fixed-point string to specify its numeric base. If no radix indicator appears, the string is assumed to be expressed as decimal digits unless the program accepting the numeric string documents a different default numbering base.

The indicator sub-field can begin with underscore (_) which separates the radix indicator letter that follows from digits of a base 12 (or higher) mantissa; the underscore prevents a "b" or "d" radix indicator from being treated as a digit of the mantissa.

The following table shows strings expressed in a variety of numbering bases, along with the decimal value of the float dec(59) number represented by that string.

INDICATOR	NUMERIC BASE	EXAMPLES	VALUE (base 10)
b or _b	base 2 or binary	10001.11b	17.75
q or _q	base 4 or quaternary	113_q	23
o or _o	base 8 or octal	21.6o	17.75
d or _d	base 10 or decimal	17.75d	17.75
x or _x	base 16 or hexadecimal	11.Cx	17.75
rN or _rN	N is a decimal base number: 2 <= N <= 16	43r5	23
		15.9_r12	17.75

The fixed-point string may end with one or more space (SP) characters. These characters are ignored by the conversion.

List of radix indicators:
   A radix indicator sub-field may appear at the end of a
   fixed-point string to indicate the base in which its digits are
   expressed. Indicator characters may also be given in uppercase.
b, _b
   digits are interpreted as a base two number (binary).
q, _q
   digits are interpreted as a base four number (quaternary).
o, _o
   digits are interpreted as a base eight number (octal).
d, _d
   digits are interpreted as a base ten number (decimal).
x, _x
   digits are interpreted as a base sixteen number (hexadecimal).
rN, _rN
   digits are interpreted in the base N which is a decimal number
   where: 2 <= N <= 16

Notes on the range of values:
Numeric strings can represent a wide range of values. The specific
range depends on the numbering base in which mantissa digits are
expressed.

Conversion from a numeric string to a float dec(59) data value
supports the widest range of input strings. The
cv_fixed_point_string_ subroutine performs such conversions.

Conversion from a data value to a numeric string supports a narrower
range of values. The numeric_to_ascii_ subroutine converts a float
dec(59) data value to a decimal numeric string. The
numeric_to_ascii_base_ subroutine converts a float dec(59) data value
to a numeric string in which mantissa digits are expressed in one of
the numbering bases: 2 <= BASE <= 16; the program which calls this
subroutine specifies which numbering system to use.

The greatest magnitude for a data value stored as float decimal(59)
real is 10**186. The smallest magnitude of such data value is
10**-128.

The following table summarizes the range of fixed-point string
magnitudes accepted for each numbering base.

		MAXIMUM MAGNITUDE	
BASE	MIN MAGNITUDE	float dec-to-character	char-to-float dec
2	0.1E-228b	1.1E+198b	1.1E+617b
3	0.1E-143_r3	2.0E+124_r3	2.1E+389_r3
4	0.1E-113q	2.0E+100q	3.2E+308q
5	0.1E-97_r5	3.4E+86_r5	1.0E+266_r5
6	0.1E-87_r6	4.2E+76_r6	1.0E+239_r6
7	0.1E-80_r7	1.5E+70_r7	1.1E+220_r7
8	0.1E-75o	2.0E+65o	7.2E+205o
9	0.1E-71_r9	1.7E+62_r9	7.4E+194_r9
10	0.1E-127	9.9...9E+185 (up to 59 significant digits)	9.9...9E+185
11	0.aE-66_r11	4.aE+57_r11	4.3E+178_r11
12	0.aE-63_r12	7.0E+55_r12	2.4E+172_r12
13	0.aE-61_r13	b.bE+53_r13	c.2E+166_r13
14	0.aE-60_r14	3.8E+52_r14	2.1E+162_r14
15	0.aP-58_r15	1.bP+51_r15	1.7P+158_r15
16	0.aP-57x	f.fP+49x	

numeric_to_ascii_base_

Full documentation for the revised numeric_to_ascii_base_ subroutine is shown below.

```
Entry:  2022-03-04  numeric_to_ascii_base_

Function: formats a real decimal floating-point number as a string
with digits expressed in a numbering system with radix selected from
2 through 16 inclusive.  The result is expressed in integer, fraction,
or exponential format depending on the characteristics of the input
number.

If base=10 is selected, numeric_to_ascii_base_ calls the
numeric_to_ascii_ subroutine to construct the numeric string using a
more precise algorithm for expressing the decimal mantissa and
exponent of the input value as a string of decimal digits.

For a non-decimal base, the algorithm repeatedly divides and/or
multiplies the input value by the numeric base to obtain successive
digits expressed in that base.  A radix indicator string is appended
to the result.

Syntax:
declare numeric_to_ascii_base_ entry (float dec(59), fixed bin,
   fixed bin) returns (char(256) varying);

result = numeric_to_ascii_base_ (value, precision, base);

Arguments:
value
   number to be formatted. (Input)  See "Notes on value argument"
   below for more information.
precision
   maximum number of significant digits placed in the result string.
   (Input)  If a 0 or negative precision is given, all significant
   digits are represented in the result.  A positive precision
   limits number of significant digits to the given value.  See
   "Notes on precision argument" below for more information.
base
   radix of the numbering system in which the result is expressed.
   (Input)  For example, base=2 produces a binary representation and
   base=8 produces an octal string.  The domain supported for base:
      2 <= base <= 16

result
   character-string representation of value using digits of the
   base numbering system. (Output)  See "Notes on result" below for
   more information.

Notes on result:
The result returned by the numeric_to_ascii_base_ function is the
ascii representation of input value in the selected numbering system.
The result uses one of the following formats:

   MANTISSA   OR   MANTISSAe+EXP   OR   MANTISSAe-EXP

While MANTISSA gives the integer or integer.fraction representation
expressed in digits of the selected numbering system, the EXP exponent
is always expressed using decimal digits.
```

For results expressed in base 15 or 16, any exponent is preceded by the 'p' exponent indicator letter because 'e' is a valid digit that can appear in the MANTISSA of these numbering bases.

```
   MANTISSA   OR   MANTISSAp+EXP   OR   MANTISSAp-EXP
```

Mantissa digits for bases 11 through 16 represent values beyond 9 using lowercase letters. Digits for the most common numeric bases are shown in the table below, ordered in ascending value. Digits for a base N lower than 16 use the first N valid digits listed for base 16.

BASE	VALID DIGITS IN THAT BASE	COMMON NAME
2	0 1	binary
4	0 1 2 3	quaternary
8	0 1 2 3 4 5 6 7	octal
10	0 1 2 3 4 5 6 7 8 9	decimal
12	0 1 2 3 5 5 6 7 8 9 a b	
16	0 1 2 3 4 5 6 7 8 9 a b c d e f	hexadecimal

The MANTISSA includes all significant radix places selected by the precision argument. If the MANTISSA includes too few radix places to represent the magnitude of the value, an exponent is added. The EXP sub-field is a signed decimal integer representing a multiplier for the MANTISSA:

```
   MANTISSA * (base ** EXP)
```

That is, a positive exponent (E+n) shifts the point n radix-places to the right, while a negative exponent (E-n) shifts the point n radix-places to the left of any point displayed in the mantissa. For an integer mantissa, the point is not displayed; it is assumed to follow the last radix-place of the displayed integer.

A radix indicator sub-field appears at the end of each non-decimal result to indicate the base in which its digits are expressed. The following sub-field value is appended.

BASE	radix indicator sub-field	
2	b	(stands for binary)
3	_r3	
4	q	(stands for quaternary)
5	_r5	
6	_r6	
7	_r7	
8	o	(stands for octal)
9	_r9	
10		(no indicator used for decimal base)
11	_r11	
12	_r12	
13	_r13	
14	_r14	
15	_r15	
16	x	(stands for hexadecimal)

All result strings are in one of the formats accepted by the cv_fixed_point_string_ subroutine. Also, binary and decimal result strings are in a format acceptable to the PL/I convert built-in function and PL/I number character-string-to-number conversion rules.

A string expressing a large input value using a small base such as binary may require 200 or more characters to hold all significant digits and exponent. The PL/I runtime reserves up to 256 characters to hold numeric strings representing the largest values in binary digits. The numeric_to_ascii_base_ return value follows that maximum string length precedent.

The result is a numeric string that can represent a wide range of values. The specific range depends on the numbering base in which mantissa digits are expressed.

The numeric_to_ascii_base_ subroutine converts a float dec(59) data value to a numeric string in which mantissa digits are expressed in one of the numbering bases: 2 <= BASE <= 16; the program which calls this subroutine specifies which numbering system to use.

The following table summarizes the range of fixed-point string magnitudes representable for each numbering base.

BASE	MIN MAGNITUDE	MAXIMUM MAGNITUDE	
2	0.1E-228b	1.1E+198b	
3	0.1E-143_r3	2.0E+124_r3	
4	0.1E-113q	2.0E+100q	
5	0.1E-98_r5	3.4E+86_r5	
6	0.1E-87_r6	4.2E+76_r6	
7	0.1E-80_r7	1.5E+70_r7	
8	0.1E-75o	2.0E+65o	
9	0.1E-71_r9	1.7E+62_r9	
10	0.1E-127	9.9...9E+185	(59 significant digits)
11	0.1E-65_r11	4.aE+57_r11	
12	0.aE-63_r12	7.0E+55_r12	
13	0.aE-61_r13	b.bE+53_r13	
14	0.aE-60_r14	3.8E+52_r14	
15	0.aP-58_r15	1.bP+51_r15	
16	0.aP-57x	f.fP+49x	

Notes on value argument:
The maximum input value supported by the numeric_to_ascii_base_ conversion algorithm depends upon the selected base. A float dec(59) value may have insufficient precision to calculate digits of an input value in a non-decimal numbering system. The maximum value which may be converted to a decimal string is therefore larger than input limits for an octal, binary, or other radix representation of a number.

The following table shows the approximate maximum value supported for each base. An attempt to use a larger input value may signal the sub_error_ condition with an appropriate message. The caller can establish an on-unit for that condition if extremely large input numbers are expected.

BASE	MAXIMUM value
2	1.0e+84
3	3.0e+59
4	2.0e+84
5	9.99999999999999999999999999999999999999999999999999e+185
6	6.0e+59
7	3.2e+59
8	2.0e+85
9	5.0e+59
10	9.99999999999999999999999999999999999999999999999999e+185
11	1.6e+60
12	1.6e+60
13	1.4e+60
14	1.5e+60
15	1.5e+60
16	2.1e+84

A reasonably-correct numeric string can be generated in any supported numeric base for the smallest possible input value: 0.1e-127.

However, PL/I decimal support operations for float dec(59) arithmetic lack the precision needed to convert very small non-decimal fractional strings back to a number.

- A non-decimal string can be converted back to a float dec(59) number only if it represents a numeric value: >= 1.0e-68.

- Any attempt to convert a smaller fraction will result in an exponent underflow condition.

Notes on precision argument:
A float dec(59) input value precisely stores up to 59 radix places of precision in a base=10 representation of that number. More radix places are needed to represent a number when base is less than 10; and fewer radix places are needed when base is greater than 10.

The following table shows the maximum precision (significant digits) supported for each base representation of an input number. This precision is used when the input precision = 0.

BASE	MAXIMUM precision (radix places)
2	199
3	125
4	99
5	85
6	77
7	70
8	66
9	62
10	59
11	57
12	55
13	53
14	52
15	51
16	49

```
Notes on base argument:
A sub_error_ condition is signalled if the given base falls outside of
the range:  2 <= base <= 16
```

octal

The following comparison shows typical changes made to documentation for: bin, dec and hex.

```
cpa >doc>info>oct.info oct.info

A1        03/31/83  octal, oct
A2
A3        Syntax:  oct values
A4
A5
A6        Syntax as active function:  [oct values]
A7
A8
A9        Function:  returns one or more values in octal.
A10
A11
A12       Arguments:
A13       value
A14          is a value to be processed.  The last character of the value
A15          indicates its type.  Acceptable types are binary (b), quartenary
A16          (q), octal (o), hexadecimal (x), or unspec (u).  Any valid PL/I
A17          real value is allowed.  The absence of any specifier means decimal.
A18          The unspec value is limited to eight characters.
Changed by B to:
B81       :Info: octal: oct:  2022-05-19  octal, oct
B82
B83       Syntax as a command:  oct VALUEs
B84
B85
B86       Syntax as an active function:  [oct VALUEs]
B87
B88
B89       Function:  returns one or more numeric or character strings in octal
B90       (numeric base 8).  Each string is terminated by a radix indicator
B91       sub-field "o" (e.g., 177o).
B92
B93
B94       Arguments:
B95       VALUE
B96          is an input string to be processed.  It is usually a fixed-point
B97          decimal string (e.g., 21 or 55.9) and may begin with either + or -
B98          sign.  It may end with a radix indicator denoting use of digits in
B99          a non-decimal base.  See "List of radix indicators" below.  It may
B100         be an arbitrary character string (1 to 8 characters long) followed
B101         by a "u" (for "unspec") indicator.  See "Notes on unspec values"
B102         below.
B103
B104
B105      List of radix indicators:
B106         Each VALUE may end with one of the following strings which indicate
B107         which numbering system's digits are expressing the value.
B108         Indicator characters may also be given in uppercase.
B109      b, _b
B110         the number is interpreted as a base two number (binary).
B111      q, _q
B112         the number is interpreted as a base four number (quaternary).
B113      o, _o
B114         the number is interpreted as a base eight number (octal).
```

```
B115      d, _d
B116         the number is interpreted as a base ten number (decimal).
B117      x, _x
B118         the number is interpreted as a base sixteen number (hexadecimal).
B119
B120
B121      rN, _rN
B122         the number is interpreted in the base N which is a decimal number
B123         between 2 and 16 inclusive.
B124
B125
B126      Notes on unspec values:
B127      A VALUE argument ending with a "u" indicator is treated as character
B128      data.  The PL/1 unspec built-in converts each ASCII character to its
B129      octal encoding for the character.  Those octal digits are then
B130      displayed.  VALUE can be from 1 to 8 characters long.  This permits
B131      the user to see the octal encoding for the given characters.
B132
B133
B134      Examples:
B135      octal (21 -21)
B136      25o
B137      -25o
B138
B139      oct 55.4
B140      67.4o
B141
B142      oct 30d
B143      36o
B144
B145      oct AB_x
B146      253o
B147
B148
B149      oct 1.E2
B150      144o
B151
B152      octal 55.5e20
B153      1.131566707255337176e+24o
B154
B155      oct ABCu
B156      101102103o
```

radix

The following info segment describes the new radix command/active function.

```
2022-05-12  radix

Syntax as a command:  radix BASE VALUEs

Syntax as an active function:  [radix BASE VALUEs]

Function:  returns one or more numeric strings in expressed in digits
of a specified numbering system.  Each non-decimal string is
terminated by a radix indicator sub-field.  See "Notes on result"
below for further details.

Arguments:
BASE
   is the radix of the numbering system digits in which each numeric
   value is to be expressed as a string.  It must be a number between
   2 and 16.
VALUE
   is an input string to be processed.  It may be a fixed-point or
   floating point decimal string (e.g., 21 or 55.9), but is often a
   numeric string ending with a radix indicator which specifies a
   non-decimal base.  See "List of radix indicators" below.

List of radix indicators:
   Each VALUE may end with one of the following strings which indicate
   which numbering system's digits are expressing the value.
   Indicator characters may also be given in uppercase.
b, _b
   the number is interpreted as a base two number (binary).
q, _q
   the number is interpreted as a base four number (quaternary).
o, _o
   the number is interpreted as a base eight number (octal).
d, _d
   the number is interpreted as a base ten number (decimal).
x, _x
   the number is interpreted as a base sixteen number (hexadecimal).

rN, _rN
   the number is interpreted in the base N which is a decimal number
   between 2 and 16 inclusive.

Notes on result:
The result returned by the radix command or active function is the
ascii representation of input value in the selected numbering system.
The result uses one of the following formats:

   MANTISSA   OR   MANTISSAe+EXP   OR   MANTISSAe-EXP

While MANTISSA gives the integer or integer.fraction representation
expressed in digits of the selected numbering system, the EXP exponent
is always expressed using decimal digits.
```

For results expressed in base 15 or 16, any exponent is preceded by the 'p' exponent indicator letter because 'e' is a valid digit that can appear in the MANTISSA of these numbering bases.

```
   MANTISSA   OR   MANTISSAp+EXP   OR   MANTISSAp-EXP
```

Mantissa digits for bases 11 through 16 represent values beyond 9 using lowercase letters. Digits for the most common numeric bases are shown in the table below, ordered in ascending value. Digits for a base N lower than 16 use the first N valid digits listed for base 16.

BASE	VALID DIGITS IN THAT BASE	COMMON NAME
2	0 1	binary
4	0 1 2 3	quaternary
8	0 1 2 3 4 5 6 7	octal
10	0 1 2 3 4 5 6 7 8 9	decimal
12	0 1 2 3 5 5 6 7 8 9 a b	
16	0 1 2 3 4 5 6 7 8 9 a b c d e f	hexadecimal

The MANTISSA includes all significant radix places selected by the precision argument. If the MANTISSA includes too few radix places to represent the magnitude of the value, an exponent is added. The EXP sub-field is a signed decimal integer representing a multiplier for the MANTISSA:

```
   MANTISSA * (base ** EXP)
```

That is, a positive exponent (E+n) shifts the point n radix-places to the right, while a negative exponent (E-n) shifts the point n radix-places to the left of any point displayed in the mantissa. For an integer mantissa, the point is not displayed; it is assumed to follow the last radix-place of the displayed integer.

A radix indicator sub-field appears at the end of each non-decimal result to indicate the base in which its digits are expressed. The following shows which sub-field value is appended for each base.

BASE	radix indicator sub-field	
2	b	(stands for binary)
3	_r3	
4	q	(stands for quaternary)
5	_r5	
6	_r6	
7	_r7	
8	o	(stands for octal)
9	_r9	
10		(no indicator appended for decimal base)
11	_r11	
12	_r12	
13	_r13	
14	_r14	
15	_r15	
16	x	(stands for hexadecimal)

All result strings are in one of the formats accepted by the cv_fixed_point_string_ subroutine. Integer results are in a format accepted by the cv_integer_string_ subroutine. Also, binary and decimal result strings are in a format acceptable to the PL/I convert built-in function and PL/I character-string-to-number conversion rules.

```
A string expressing a large input value using a small base such as
binary may require 200 or more characters to hold all significant
digits and exponent.  The PL/I runtime reserves up to 256 characters
to hold numeric strings representing the largest values in binary
digits.  The radix return value follows that maximum string length
policy.

Examples:
radix 5 21
41_r5

radix 16 55.7o
2d.ex

radix 10 33q
15

radix 16 55.5e40
6.5eff97e9da95b78dbad0b2878p+34x
```

round

The following documentation describes the new round command/active function.

```
2021-12-05  round

Syntax as a command:  round NUMBER PRECISION

Syntax as an active function:  [round NUMBER PRECISION]

Function:  rounds NUMBER to PRECISION decimal places.

Arguments:
NUMBER
   a fixed-point number with precision up to 59 decimal places.
   NUMBER may end with an optional radix indicator to specify
   a value expressed in a non-decimal base.  See "List of radix
   indicators" below.  An E-Format exponent may be given with a
   decimal number in the range:     -128 <= EXPONENT <= +127
   A narrower range of exponents is permitted for non-decimal
   numbers.
PRECISION
   a decimal integer between 1 and 59 indicating the number of
   significant digits to retain when rounding NUMBER at the
   PRECISIONth significant digit.

List of radix indicators:
   Indicator characters may also be given in uppercase.
b, _b
   the number is interpreted as a base two number (binary).
   See "Notes" below.
q, _q
   the number is interpreted as a base four number (quaternary).
o, _o
   the number is interpreted as a base eight number (octal).
d, _d
   the number is interpreted as a base ten number (decimal).
   See "Notes" below.
x, _x
   the number is interpreted as a base sixteen number (hexadecimal).

rN, _rN
   the number is interpreted in the base N which is a decimal number
   between 2 and 16 inclusive

Examples:
The following example shortens a lengthy quotient to the 4 most
significant digits.

quotient 23.5 3
7.833333333333333333333333333333333333333333333333333333333333333

round [quotient 23.5 3] 4
7.833

The rounding operates on the mantissa digits of a floating-point
number.

round [quotient 23.5e+80 3] 4
7.833e80
```

Version History

Date	Revision	Author	Comment
2021-12-07	1.0	Gary Dixon	Initial draft of this MCR.
2022-06-10	2.0	Gary Dixon	Publication-version of this MCR.
2022-06-13	2.1	Gary Dixon	Fix typos and minor documentation errors.
2022-06-13	2.2	Gary Dixon	Add missing changes to bound_full_cp_.bind.