APL+Win Version 14.2.10 Beta

Feb 13, 2015

This document contains version history information for this APL+Win 14.2.10 Beta. Please report any problems or comments pertaining to this beta to support@apl2000.com.

Note: This beta will expire on or about Feb 28, 2015.

APL+Win 14.2.10

Bug Fix

1. In the 14.2.09 beta, ⎕CV did not correctly apply a left argument that was any arbitrary vector of characters.
2. The 14.2.09 beta crashed when loading some older workspaces storing ⎕cself data.

APL+Win 14.2.09

Bug Fix

3. In the 14.2.06 Beta, APL+Win crashed when executing the following Win32 functions: EnumFontFamiliesEx, EnumWindows, GetOpenFileName and SHBrowseForFolder.
4. APL+Win could crash or corrupt the workspace if APL+Win performed a workspace compaction while parsing assignment statements.

Enhancements

1. Update: ⎕RL - Random Link
   The ⎕RL system function has been updated to support several new pseudo-random number generation algorithms to complement the existing pseudo-random number generation algorithm, Multiplicative Linear Congruential Multiplier. They are:
   - Historical Multiplicative Linear Congruential Multiplier
   - Extended Historical Multiplicative Linear Congruential Multiplier

- Mersenne Twister (32-bit)
- Subtract with Carry (24-bit)

IMPORTANT: This enhancement has resulted in the removal of the □RNG system function and □RLX system variable previously introduced in version 14.2.06.

□rl Random Link

Purpose:
This workspace-related system variable sets or gets information about the type of the pseudo-random number generation algorithm used by the system, the current state of that pseudo-random number generator, and the optional discard value used by the pseudo-random number generator in advancing its internal state.

Syntax: *result* ← □rl
□rl ← *an integer scalar or a vector of two or three elements*

Domain:

The result and the range of permissible values of □rl depend on the pseudo-random number generation algorithm in use by the system.

The default value of □rl in a clear workspace is scalar 7*5. This indicates that the historical pseudo-random number generation algorithm, Multiplicative Linear Congruential with multiplier 16807 (7*5) will be used. Setting □rl to any scalar positive integer will cause the system to use the historical algorithm. Setting □rl to a scalar positive integer will modify the internal state of the historical algorithm. This definition of □rl preserves the historical syntax and features of □rl.

Several additional pseudo-random number generation algorithms may be used by the system in which case the result and the range of permissible values of □□□ must be a two or three element (possibly nested) vector. Additional pseudo-random number algorithms may be implemented in a future version of APL+Win.

Pseudo-random Number Algorithm	Element #1	Element #2	Element #3
Historical Multiplicative Linear Congruential Multiplier 16807	positive scalar integer indicating the internal state of the historical algorithm	n/a in this case ⎕rl is a scalar	n/a in this case ⎕rl is a scalar
Extended Historical Multiplicative Linear Congruential Multiplier 16807	zero	internal state	optional positive integer discard value
Mersenne Twister 19937 (32-bit)	¯1	internal state	optional positive integer discard value
Subtract with Carry (24-bit)	¯2	internal state	optional positive integer discard value
Multiplicative Linear Congruential Multiplier 48271	¯3	internal state	optional positive integer discard value

For pseudo-random number generation algorithms other than the non-extended historical algorithm, the value of ⎕rl must be a two or three element vector. The first element is a non-positive integer which indicates the pseudo-random number generation algorithm to be used by the system.

The second element of ⎕rl is used to set or restore the internal state of the pseudo-random number generation algorithm. If the second element of ⎕rl is ⍬ the default internal state will be used by the algorithm. The second element of ⎕rl can be a scalar or homogeneous vector, containing integer, floating point, character, Boolean, or a UCS character vector containing the representation of the internal state along with other information pertaining to the pseudo-random number generator. For the extended historical algorithm the internal state is represented by a scalar.

The default internal state is described in the following table.

Algorithm	Default Internal State
Historical Multiplicative Linear Congruential Multiplier 16807	16807
Extended Historical Multiplicative Linear Congruential Multiplier 16807	16807
Mersenne Twister 19937 (32-bit)	obtained by capturing the second element of `⎕rl ¯1 0`
Subtract with Carry (24-bit)	obtained by capturing the second element of `⎕rl ¯2 0`
Multiplicative Linear Congruential Multiplier 48271	48271

In some cases the internal state of the algorithm is returned in the second element of `⎕rl` as a vector of Unicode characters because the information necessary to represent the native-C++ internal state of the algorithm cannot be represented in the form of an APL+Win data type. In this case, the first four elements of the vector comprise an MD5 checksum. The fifth element of the vector contains the algorithm type code. The sixth element of the vector is the discard value. The seventh element of the vector contains the size of the algorithm state in bytes. The eighth and subsequent elements comprise the native-C++ internal state of the algorithm. The checksum is based on the algorithm type, the discard value, the state size and the state data and is used to validate the data. This UCS-format internal state can be captured by the APL+Win programmer and subsequently used to set the internal state of the algorithm.

The third element of `⎕rl` is an optional non-negative integer scalar in the range `[0, ¯1+2⋆32]`, which is the specified number of pseudo-random numbers to be discarded when the selected algorithm is first used after it has been selected. The default value of the discard value is zero of it is not otherwise specified. For certain pseudo-random number generation algorithms the discard value is useful to eliminate the initial values generated which, by design, may not be reasonably distributed. The maximum allowable optional discard value can be specified by adding the MaxDiscard entry in the [RNG] section of the APLW.INI file, which by default, this maximum value is set to `¯1+2⋆32`.

Effect:
The system uses the value of `⎕rl` to select the type of pseudo-random number generator, set the internal state of that generator, and optionally discard initial generated values for computing the result of the APL+Win roll (monadic `?`) and deal (dyadic `?`) functions.

The value of `⎕rl` is preserved when an APL+Win workspace is saved.

As the system generates pseudo-random numbers, the internal state of the generation algorithm is updated in ⎕rl. The captured value of ⎕rl may be used to set the value of ⎕rl in order to reproduce the same sequence of pseudo-random numbers.

Examples:

```
      )clear
CLEAR WS
```

By default the system uses Algorithm 0.

```
      ⎕rl
16807
```

Generate three random numbers from 1 to 100:

```
      ?3⍴100
14 76 46
      ⎕rl
984943658
```

When the historical algorithm is selected, ⎕rl[⎕io] may only be set to a non-negative integer in the range [0, ¯2+2*32].

```
      ⎕rl
16807
```

Select Algorithm 0 and specify the use of the default state:

```
      ⎕rl← 0 ⍬
```

Generate three random numbers from 1 to 100:

```
      ?3⍴100
14 76 46
```

Select Algorithm 0, set the use of the default state and specify an initial discard value:

```
      ⎕rl← 0 ⍬ 3
      state← ⎕rl
      state
0 <<<UCS Characters>>> 3
```

Note: In the example above, if the discard value is 0 instead of 3, the internal state returned would have been an integer scalar. In order to preserve the discard value, the ⎕rl returns a vector.

Generate three random numbers from 1 to 100:

```
      ?3⍴100
54 22 5
```

Note: In the example above, the numbers were generated after the internal state of the generator has been advanced three times.

Select the Mersenne Twister generator, specify the use of the default state and no optional discard value and capture the value of ⎕rl:

```
      ⎕rl← ¯1 ⍬
      state ← ⎕rl
      state
¯1 <<<UCS Characters>>> 0
```

Note: In the example above the optional discard value has been set to zero and ⎕rl returns a three-element vector.

The UCS character vector above contains:

```
      elm2← ⊃state[2]
      data← ⎕ucs elm2
      data[⍳8]
¯405587845 ¯1425600422 ¯143747245 1727617676 ¯1 0 5000 624
```

Note: The first four elements of the data vector as a whole contains the checksum; the fifth element contains the algorithm type (`¯1`); the sixth element contains the discard value (`0`); the seventh element contains the state size (`5000`); the eighth element (`624`) marks the beginning of the state data.

Generate three random numbers from `1` to `100`:

```
      ?3⍴100
82 14 91
```

Reset the state of the Mersenne Twister generator:

```
      ⎕rl← state
```

Generate three random numbers from `1` to `100`:

```
      ?3⍴100
82 14 91
```

Select Algorithm `0` and specify the initial internal state:

```
      ⎕rl← 0 16807
      ⎕rl
16807
```

Select Algorithm `0` and specify the initial internal state:

```
      ⎕rl← 0 'a'
      ⎕rl
2027626332
```

Note: In the example above, character ‘a’ is used by a seed sequence object to set the internal state of the historical algorithm.

Select Algorithm `0` and specify the initial internal state:

```
      ⎕rl← 0 ⎕ts
```

Note: In the example above, the value of `⎕ts` (an integer vector) is used by a seed sequence object to set the internal state of algorithm.

Select Algorithm `¯1` and specify the initial internal state:

```
      ⎕rl← ¯1 ⎕guid
```

Note: In the example above, the value of `⎕guid` (a character vector containing a Global Unique Identifier) is used by a seed sequence object to set the initial internal state of the algorithm.

Select Algorithm `¯2` and specify the initial internal state:

```
      ⎕rl← ¯2 (12.45 ¯23.67 34.125)
      ⎕rl
¯2 <<<UCS Characters>>> 0
```

Note: In the example above, the floating point vector is used by a seed sequence object to set the initial internal state of the algorithm.

Select Algorithm `¯2` and specify the initial internal state:

```
      ⎕rl← ¯2 (10 ⍴ 1 0)
```


```
      ⎕rl
¯2 <<<UCS Characters>>> 0
```

Note: In the example above, the Boolean vector is used by a seed sequence object to set the internal state of the algorithm.

Select Algorithm `¯3` and specify the initial internal state:

```
      ⎕rl← ¯3 234.56
      ⎕rl
¯3 <<<UCS Characters>>> 0
```

Note: In the example above, the floating point value is used by a seed sequence object to set the internal state of the algorithm.

Pseudo-Random Number Algorithms

The APL+Win roll (monadic `?`, e.g. `?100`) and deal (dyadic `?`, e.g. `10?100`) functions employ pseudo-random number generation algorithms so that programmers can simulate random events or create cryptographic keys. These algorithms are fully determined by their seed value so that the sequence generated can be repeated by using the same seed value. The repeatable property of these algorithms is useful for testing applications involving the roll or deal functions, however this repeatable property is what makes them pseudo-random. The period of a pseudo-random number algorithm is the maximum length, among all possible seed values, of a sequence of values generated by the algorithm which is non-repeating.

Prior to APL+Win v15, the pseudo-random number algorithm inherently supported was the 'linear congruential' algorithm with linear constant term set to zero and initial multiplier set to 7*5. Starting with APL+Win v15, additional pseudo-random number algorithms are inherently supported and may be selected by the APL+Win programmer, with the potential for additional algorithms to be added in the future.
Pseudo-random number generator algorithms are analyzed on the following bases:

- Period 'length' for specific seed values
- Correlation among successively generated values
- Distribution of generated values among the possible values
- Mathematical 'complexity' resulting in processing time variation

The underlying pseudo-random number generator algorithm is used to obtain a pseudo-random number in a specified range. This value is mapped to the integer range space of the APL+Win roll function. For example the range space of the pseudo-random number generator may be values in [0, 1], but the range space of the APL+Win roll function is programmer determined, e.g. For `?10` the range space is [1, 10] in index origin 1.

A pseudo-random number generator seed may itself be generated by a seeding generator. This technique is deemed to 'inject entropy' into the resulting sequence of pseudo random numbers, e.g. Seed Sequence. Refer to http://www.cplusplus.com/reference/random/seed_seq/. For detailed information about the specification of the Seed Sequence utility, refer to Section 26.5.7.1 of the working draft of the Standard for Programming Language C++. This specification is available at http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4296.pdf.

Mersenne Twister

Mersenne Twister algorithm provides fast generation of high-quality pseudorandom integers. The specific variant of Mersenne Twister implemented in APL+Win is 'MT19937' as provided by the Microsoft operating system. It is based on the Mersenne prime 2^{19937}-1 and produces a sequence of 32-bit numbers with a state size of 19937 bits. Its predefined parameters are:

- 32: word size
- 624: state size
- 397: shift size
- 31: mask bits
- 0x9908b0df: xor mask
- 11:tempering shift parameter u
- 0xffffffff: tempering bitmask parameter d
- 7: tempering shift parameter s
- 0x9d2c5680: tempering bitmask parameter b
- 15: tempering shift parameter t
- 0xefc60000: tempering bitmask parameter c
- 18: tempering shift parameter l
- 1812433253: initialization multiplier

Range of values generated: [0, $2^{32} - 1$]
Default Seed: 5489
More information:

- http://en.wikipedia.org/wiki/Mersenne_twister
- http://www.cplusplus.com/reference/random/mt19937

Linear Congruential with Initial Multiplier 48271

Linear congruential algorithm yields a sequence of numbers computed with a discontinuous piecewise linear equation. The pre-existing version of this algorithm in APL+Win uses zero as the linear constant term. The specific variant of Linear Congruential with Initial Multiplier 48271 implemented in APL+Win is 'MINSTD_RAND' as provided by the Microsoft operating system. Its predefined parameters are:

- 48271: the multiplier
- 0: the increment
- 2147483647:the modulus

Range of values generated: [Min: 1, $(2^{31} - 1) - 1$]]
Default Seed: 1
Initial State: 48271
More information:

- http://en.wikipedia.org/wiki/Linear_congruential_generator
- http://www.cplusplus.com/reference/random/minstd_rand.

Subtract with Carry

Subtract with carry is a lagged Fibonacci algorithm with a state sequence of integer elements, plus one carry value. This algorithm is a generalization of the L'Ecuyer RNG (University of Montreal). The specific version of Subtract with Carry implemented in APL+Win, as provided by the Microsoft operating system, is 'RANLUX24_BASE' which produces 24-bit numbers. Its predefined parameters are

- 24: number of bit in each word in the state sequence
- 10: number of elements between advances
- 24: value that determines the degree of recurrence in the generated series

Range of values generated: [0, $2^{24} - 1$]
Default Seed: 19780503
More information:

- http://en.wikipedia.org/wiki/Subtract_with_carry
  http://www.cplusplus.com/reference/random/ranlux24_base

2. Update: ⎕FX system function

   The ⎕FX system function has been improved to allow the argument to be a nested vector of character vectors (or scalars), rather than requiring it to be a matrix. For example, the line below in version 15

```
⎕FX 'res←left MyFn right' 'res←left + right' 'res←res,left * right'
```

   is equivalent to the line below in prior versions of APL+Win

```
⎕FX ⊃[2] 'res←left Foo right' 'res←left + right' 'res←res,left * right'
```

   In addition to supporting a vector of character vectors, the argument can actually be a nested array of any rank containing sub-items that can themselves be character arrays of any rank, or even nested sub-items together in the same argument.

   ⎕FX has also been extended to allow embedded ⎕TCNLs in the argument (at any level of nesting) to represent new lines in the function. It also removes trailing blanks from the ends of lines.

3. New: ⎕CM - Character Matrix

   The new Character Matrix system function, ⎕CM, can be used to normalize a character argument into a character matrix result.

   Like the ⎕FX system function, ⎕CM accepts as its argument a nested or heterogeneous array of any rank, gluing all items together into a character matrix. The overall array and each item and

subitem in the array is processed in ravel order.

The overall array and each item and subitem in the array is processed in ravel order. But this is not to say it is the same as ⎕ENLIST. Each character matrix or higher rank item in the argument (including sub-items) gets an implicit newline for each row. Each nested sub-item gets at least one new row in the result also (and if it contains nested sub-items or character matrix or higher rank items, those also get newlines in the result).
In addition to allowing ⎕TCNL as a line separator in the argument, ⎕CM (as well as ⎕FX and ⎕CV) also recognize ⎕TCLF or the two-character sequences (⎕TCNL,⎕TCLF) and (⎕TCLF,⎕TCNL) to represent ONE newline in the result. This means that the string "abc",⎕TCNL,⎕TCNL,"def" gives three lines in the result (one of them blank between "abc" and "def"). But the similar string "abc",⎕TCNL,⎕TCLF,"def" will only produce two lines in the result because the ⎕TCNL,⎕TCLF sequence is considered a single newline in argument. These rules make it easier to handle text files that may have originated in Windows, Linux, Unix, OSX, etc, with equal ease.

For example:

```
      ⎕CM 'abc' (2 2⍴'DEFG') ('hello',⎕tcnl,'world!')
abc
DE
FG
hello
world!
      ⍴⎕CM 'abc' (2 2⍴'DEFG') ('hi',⎕tcnl,'there')
5 5
```

Or even more complicated, we have the following:

```
      ⎕CM 3 2 ⍴ 'abc' (2 2⍴'DEFG') ('hello',⎕tcnl,'world!')
abc
DE
FG
hello
world!
abc
DE
FG
hello
world!
      ⍴⎕CM 3 2 ⍴ 'abc' (2 2⍴'DEFG') ('hello',⎕tcnl,'world!')
```

```
10 6
```

Note 1: This is not the same as applying the THORN (⍤) primitive to the argument. For example, thorn of the same array would produce the following result:

```
      ⍤ 3 2 ⍴ 'abc' (2 2⍴'DEFG') ('hello',⎕tcnl,'world!')
 abc          DE
              FG

 hello
world! abc

 DE           hello
world!
 FG
```

Note 2: ⎕CM will remove trailing blanks from lines.

4. New: ⎕CV - Character Vector
   The ⎕CV system function can be used to normalize a character vector argument into a character vector result. By default, the character vector result is ⎕TCNL delimited. But available is an optional left argument that can specify a different line delimiter.

   Syntax: res ← *{separator}* ⎕CV *array*
   Arguments:
   {separator} is the optional left argument to specify the line separator sequence. The default ⎕TCNL line separator can be overridden by specifying a left argument that is a character scalar or non-empty character vector.
   For example, comma and semi-colon. The separator is inserted between each line of output, but not to the beginning or end of the result.
   array is any array of characters arrays or scalars.
   Result:
   The result is a character vector.
   Example:

```
      ⎕CV 'ONE' 'TWO' 'THREE'
ONE
TWO
THREE
      ⎕TCHT ⎕CV 'ONE' 'TWO' 'THREE'
```

```
ONE      TWO   THREE
      ':' ⎕CV 'ONE' 'TWO' 'THREE'
ONE:TWO:THREE
      ':' ⎕TCHT ⎕CV 'ONE' 'TWO' 'THREE'
ONE:     TWO:  THREE
```

Note: ⎕CV will remove trailing blanks from lines.

5. New: ⎕CRLF - Newline + Line feed

   The ⎕CRLF system variable returns a two element vector consisting of (⎕TCNL,⎕TCLF) and is a convenient left argment to ⎕CV for producing standard Windows delimited output. In C/C++/C# this is the "\r\n" sequence.

6. New: ⎕guid - Global Unique Identifier Generator

   The ⎕guid system function is used to obtain a character vector containing a new Global Unique Identifier (GUID). The result may be used as a persistent unique identifier with a high degree of certainty. For more information see http://en.wikipedia.org/wiki/Globally_unique_identifier.

   Syntax: *result* ← ⎕guid
   Result:
   result is a character vector, generated by the Windows operating system, which is enclosed in 'curly braces'. The representation is of a 128-bit integer in the format of 32 hexadecimal digits grouped according to convention.
   Example:

```
      )clear
CLEAR WS
      ⎕guid
{A3EE881B-7AB4-4B38-9237-A2846E43EF9A}
```

7. Update: aplw.exe.manifest and aplwr.exe.manifest
   The updated aplw.exe.manifest and aplwr.exe.manifest files enable the GetVersionEx Windows function to now correctly detect and report the version number as 6.3 for the Windows 8.1 and Windows Server 2012 R2 operating systems.

APL+Win 14.2.08

BUG FIX

A virtual memory allocation failure error in APL+Win 10.0 and higher could prevent APL+Win from starting with a workspace size greater than 2 GB.

APL+Win 14.2.07

BUG FIX

In the 14.2.06 Beta, APL+Win crashed when executing the following Win32 functions: EnumFontFamiliesEx, EnumWindows, GetOpenFileName and SHBrowseForFolder.

APL+Win 14.2.06

Enhancements

1. **Colon primitive (:)**

Description:

A colon prefix followed by a space (colon-space) at the beginning of a statement in a user defined function will suppress the output on any line of execution including a :THEN expression in Inline Control Structures.

2. **Sink primitive (←)**

Description:

Suppress the output from any line of execution in immediate execution (session) and user defined function when executed monadically.

Syntax:

← expression

where expression is any valid APL expression that returns a value.

Example:

```
      ← 2 3⍴⍳6
⍬ no output displayed
```

3. **⎕rng - Random Number Generator (obsoleted as of v14.2.09)**

Purpose:

This workspace-related system variable sets the type of the pseudo-random number generator to be used by the system.

Syntax: result ← ⎕rng

⎕rng ← number

Domain:

Integer in [0,3]. In a clear workspace, the default value is 0 which corresponds to the historical method used by APL+Win.

Effect:

The system uses the value of ⎕rng to select the desired random number generators to be used by the roll (monadic ?) and deal (dyadic ?) primitive functions. Changing the value of ⎕rng to a value different than its previous value will cause the system to reset ⎕rl and ⎕rlx to their default values.

4. **⎕rlx - Random Link Extended (obsoleted as of v14.2.09)**

Purpose:

This workspace-related system variable sets or gets the discard value (or advance internal state), seed value (or random link), and additional optional value(s) used by a seed sequence object in seeding the RNGs. It is applicable only when ⎕rng > 0.

Syntax: result ← ⎕rlx

⎕rlx ← vector of numbers with a minimum of two elements

5. **The APLW.WS ActiveX Server object now supports being instantiated by a 32-bit non-APL+Win client application.**

Note: that this enhancement facilitated renaming the XStart method to XInit.

Observe the following behaviors when starting the new APLW.WS server from a non-APL+Win client:

a. The default Workspace (if no Workspace specified in XInit or XInit invoked implicitly) will be "" (no default workspace).

b. The default Argument (if no Argument specified in XInit or XInit invoked implicitly) will be "".

c. The default Directory (if no Directory specified in XInit or XInit invoked implicitly) will be the current directory where the aplwCo.dll.

d. The default ExePath (if no ExePath specified in XInit or XInit invoked implicitly) will be based on the name of the aplwCo.dll.

BUG FIX

The display or formatting of a deeply nested array could cause APL+Win to crash unexpectedly instead of reporting a LIMIT ERROR. This was found to occur when the product of the elements in the number of rows and cols in the nested array exceeded the maximum number of elements supported in a 32-bit signed integer.