
APL+Win Version 14.2.06 Beta
Copyright (c) 2014 APLNow LLC.
All Rights Reserved
Nov 24, 2014

This document contains version history information for this APL+Win 14.2.06 Beta. Please report any problems or comments pertaining to this beta to support@apl2000.com.

**Note**: This beta will expire on or about Dec. 31, 2014.

## 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

**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.

| Value Range of ⎕rng | Type of RNG | Default Value of ⎕rl | Default Value of ⎕rlx |
|---|---|---|---|
| 0 | Multiplicative Linear Congruential with Multiplier set to 16807 | 16807 | N/A |
| 1 | Mersenne Twsister 19937(32-bit) | N/A | 2 ⍴ 0 3499211612 |
| 2 | Multiplicative Linear Congruential with Multiplier set to 48271 | N/A | 2 ⍴ 0 48271 |
| 3 | Subtract with Carry (24-bit) | N/A | 2 ⍴ 0 15039276 |

**Examples:**

```
      )clear
CLEAR WS
      ⎕rng
0
      ⎕rl
16807
```

Generate three random numbers from 1 to 100.

```
      ?3⍴100
14 76 46

      ⎕rl
984943658
```


Select a different random number generator.

```
      ⎕rng←2
      ⎕rlx
0 48271
```

Generate three random numbers from 1 to 100.

```
      ?3⍴100
9 61 90
      ⎕rlx
0 1914720637
```

## 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, among all possible seed values, length of a sequence of values generated by the algorithm which is non-repeating.

Prior to APL+Win v14.3, the pseudo-random number algorithm inherently supported was the 'linear congruental' algorithm with linear constant term set to zero. Starting with APL+Win v14.3, 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. The seed data may be a single unsigned integer, e.g. Linear Congruential, or a sequence of integer values

used by a seed generator to produce a seed vector for the pseudo random number generator, e.g. Seed Sequence. Refer to http://www.cplusplus.com/reference/random/seed_seq/.

## 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'. 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
Initial State: 3499211612
More information:

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

## Linear Congruential

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 implemented in APL+Win is 'MINSTD_RAND'. 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 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
Initial State: 15039276
More information:

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

### 4. ⎕rlx - Random Link Extended

**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*

**Domain:**
The following table specifies the ranges of values that can be used to seed the respective RNGs.

| Type of RNG | Min | Max |
|---|---|---|
| `1` | `0` | `4294967295 (¯1+2⋆32).` |

| | | |
|---|---|---|
| 2 | 1 | 2147483646 (¯2+2⋆31) |
| 3 | 0 | 16777215 (¯1+2⋆24). |

**Effect:**
The system uses the value of ⎕rlx to compute the result of the roll (monadic ?) and deal (dyadic ?) primitive functions when ⎕rng > 0. As the system generates each pseudo-random number in a sequence, it uses the ⎕rlx seed data in the computation and updates ⎕rlx as the seed for the next value in the sequence.

⎕rlx consists of at least 2 elements, the first being the discard value and the remaining values being the seed values.

The discard value advances the internal states of the RNG as if the new random numbers are being generated. By default the discard value is set to 0. For some RNSs increasing the discard value will avoid using the initial values of the sequence which may not be reasonably distributed.

The seed, if contains nonnegative value, will be used to seed the RNG in subsequent random number generation.

If the first element of the seed is ¯1, the remaining elements of ⎕rlx, if any, will be used as a seed generator sequence to produce a series of unsigned integers with 32 significant bits to seed the RNG in subsequent random number generation.

**Examples:**

```
      )clear
CLEAR WS
      ⎕rlx
0 0
```

Select the pseudo random number generator corresponding to '2':

```
      ⎕rng←2
      ⎕rlx
0 48271
```

Generate three random numbers in [1,100] (index origin 1):

```
      ?3⍴100
9 61 90
```

Use discard value of 0 and the vector 123 456 789 to set up a seed generator sequence value:

```
      ⎕rlx←5 ⍴ 0 ¯1 123 456 789
```

Generate three random numbers in [1,100] (index origin 1):

```
      ?3⍴100
1 9 61
      ⎕rlx
0 1291394886 123 456 789
```

If ⎕rlx is not modified by the APL+Win programmer, 1291394886 will be used in subsequent random number generation.


**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.