kiPrng.dll (tm) Kenneth Ives kenaso@tx.rr.com

I am open to ways to improve this application, please email me.

Visual Basic 6.0 with Service Pack 6 runtime files required.
To obtain required files (VBRun60sp6.exe):
http://www.microsoft.com/downloads/details.aspx?FamilyId=7B9BA261-7A9C-43E7-9117-F6730
77FFB3C

VBRun60sp6.exe installs Visual Basic 6.0 SP6 run-time files.
http://support.microsoft.com/kb/290887

This software has been tested on Windows XP through Windows 7.
Windows 9x, 2000 and NT4 are no longer supported.

NOTE: This application is slow due to the formatting for display purposes
and file creation, not the generation of the data. The primary
purpose of this application is to introduce you to more secure ways
of creating random values.

==========

All nine algorithms have output within these ranges:

-0.9999999999999 to 0.9999999999999	Double Precision
-2147483648 to 2147483647	Long Integer

My observations have been that any of the below listed random number generators
will pass or fail any particular test when values are generated because there
is no such thing as true randomness without an external reference such as
radioactive decay, noise, etc. However, these random number generators will
pass all or most of the Diehard and ENT tests the majority of the time. TT800
has been tweaked by me to enhance the quality of output values in order to pass
the Diehard test scenarios. See TestResults.zip for results of testing.

For cryptographic quality values, I have created a routine to convert a value
from a strong random value to a cryptographic value. Set the property value
CryptoQuality() to TRUE. The CryptoAPI module will ignore this request since
it already produces cryptographic values.

MT19937	Mersenne Twister (has a period of 2^19937-1)
MT11231A	Off-shoot of Mersenne Twister has a period of 2^11231-1
MT11231B	Off-shoot of Mersenne Twister has a period of 2^11231-1
TT800	Off-shoot of Mersenne Twister has a period of 2^800-1

==========
Testing software available
==========

The easiest way to create a test file is to check the Diehard checkbox
on the main screen of the demo program. This option will create an
11mb (approx) binary file with an extension of ".BIN". Use this binary
file as the input for your tests with Diehard, ENT or NIST.

Diehard software
http://stat.fsu.edu/pub/diehard/

Ent Software
http://www.fourmilab.ch/random/
download the file Random.zip

NIST (National Institute of Standards and Technology) testing software

http://csrc.nist.gov/groups/ST/toolkit/rng/index.html

1. Download source code
   http://csrc.nist.gov/groups/ST/toolkit/rng/documents/sts-2.1.zip
   or
   http://csrc.nist.gov/groups/ST/toolkit/rng/documents/sts-2.1.1.zip

2. Compile software using Cygwin. If anyone gets this to compile for
   Windows, please email me. I would like to get the binaries so I can
   start testing with the newer version of the NIST software. Thank you.
   http://sourceware.org/cygwin/

3. If you are not a C programmer then download an older version with
   the binaries at:
   http://www.cs.sunysb.edu/~algorith/implement/rng/distrib/sts-1.6.zip

4. Warning! The NIST testing suite is very thorough but time consuming.
   The process may take a few hours to longer than a day to complete.
   The reports are very detailed.

April 27, 2010: NIST Special Publication 800-22rev1a (dated April 2010),
A Statistical Test Suite for the Validation of Random Number Generators
and Pseudo Random Number Generators for Cryptographic Applications, that
describes the test suite.
http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf

Read PDF file Prng_Testing.pdf distributed with this application
concerning the parameters for NIST testing.

```
================================================================================
Available in cPRNG (clsRandom)
A cryptographically random number generator using Microsoft's CryptoAPI.
================================================================================

' ************************************************************************
' Enumerations
' ************************************************************************
  Public Enum enumPRNG_ReturnFormat
      ePRNG_ASCII           ' 0
      ePRNG_HEX             ' 1
      ePRNG_HEX_ARRAY       ' 2
      ePRNG_BYTE_ARRAY      ' 3
      ePRNG_LONG_ARRAY      ' 4
      ePRNG_DBL_ARRAY       ' 5
  End Enum

  Public Enum enumPRNG_HashAlgorithm
      ePRNG_MD2             ' 0
      ePRNG_MD4             ' 1
      ePRNG_MD5             ' 2
      ePRNG_SHA1            ' 3
      ePRNG_SHA256          ' 4
      ePRNG_SHA384          ' 5
      ePRNG_SHA512          ' 6
  End Enum

  Public Enum enumPRNG_Compare
      ePRNG_CaseSensitive   ' 0 - Exact byte match
      ePRNG_IgnoreCase      ' 1 - Uppercase/Lowercase considered same
  End Enum

' ************************************************************************
' ****                       Properties                               ****
' ************************************************************************

  StopProcessing - Input/Output - Boolean - True if user wants to stop processing

  AES_Ready - Output - Boolean - True if operating system can use SHA2 functionality

  CompareMethod - Input - Long Integer - Designates type of data comparison to be used

' ************************************************************************
' ****                       Methods                                  ****
' ************************************************************************
' Build random data using ASCII values 0-255.
Function BuildRndData(ByVal lngDataLength As Long, _
             Optional ByVal lngReturnFormat As enumPRNG_ReturnFormat =
ePRNG_BYTE_ARRAY, _
             Optional ByVal blnCreateExtraSeed As Boolean = True) As Variant

' Build random data that falls between two ASCII values, inclusive.
Function BuildWithinRange(ByVal lngDataLength As Long, _
                 Optional ByVal lngLowValue As Long = 0, _
                 Optional ByVal lngHighValue As Long = 255, _
                 Optional ByVal lngRetDataType As enumPRNG_ReturnFormat =
enuByteArray, _
                 Optional ByVal blnCreateExtraSeed As Boolean = True) As Variant

' The data will be SORTED.  This routine removes all duplicates based on
' user selection of case sensitivity. The number of duplicates removed
' are returned.
```

```
Function RemoveDupes(ByRef avntData As Variant, _
            Optional ByRef lngDupeCnt As Long = 0, _
            Optional ByVal blnReturnMixed As Boolean = False) As Boolean

' An array of data passed to this routine will be rearranged.
Sub ReshuffleData(ByRef avntData As Variant, _
         Optional ByVal lngMixCount As Long = 25)

' With this routine you can generate a series of non-repeating numbers.
' An array will be loaded starting with the base number (lngMinValue)
' requested up to the maximum value requested (lngMaxValue).  You can
' also enter the incremental step between the minimum and maximum value.
' This array is then passed to another routine ReshuffleData() to be
' throughly rearranged.  When it is returned, the requested number of
' elements (lngReturnQty) from the mixed array are transferred
' sequentially to the return array (alngMixed()).
'
' Syntax:  x = NonRepeatingNbrs(100, 0, 9999, 5)
'          Return 100 numbers, lowest = 0, highest = 9999,
'          incremental step = 5, Sort return data in
'          Ascending order (default)
Function NonRepeatingNbrs(ByVal lngReturnQty As Long, _
                          ByVal lngMinValue As Long, _
                          ByVal lngMaxValue As Long, _
                 Optional ByVal lngStep As Long = 1, _
                 Optional ByVal blnSortData As Boolean = True) As Long()

' CombSort is faster than all but QuickSort and close to it.  On the
' other hand, the code is much simpler than QuickSort and can be easily
' customized for any array type.  The CombSort was first published by
' Richard Box and Stephen Lacey in the April 1991 issue of Byte magazine.
Function CombSort(ByRef avntData As Variant, _
         Optional ByVal blnAscending As Boolean = True) As Boolean

' Generate a one-way hash string from a string of data. These are the
' algorithms to use:   MD2  MD4  MD5  SHA-1  SHA-256  SHA-384  SHA-512
'
' Special note:  SHA-224, SHA-512/224 and SHA-512/256 have not yet been
' implemented into the Microsoft crypto suite of hashes.
Function CreateHash(ByVal strInput As String, _
           Optional ByVal lngHashAlgo As enumPRNG_HashAlgorithm = ePRNG_SHA512, _
           Optional ByVal blnReturnAsHex As Boolean = True) As String

' Generate a random long integer between two input values.
Function GetRndValue(ByVal sngLow As Single, _
                     ByVal sngHigh As Single) As Long

' Convert a long integer to a double precision number.  Returns a decimal
' position of 14 places.
Function LongToDouble(ByVal lngValue As Long) As Double

' This is an ArrPtr function that determines if the passed array is
' initialized, and if so will return the pointer to the safearray header.
' If the array is not initialized, it will return zero.
' Syntax:   If CBool(IsArrayInitialized(array_being_tested)) Then ...
Function IsArrayInitialized(ByVal avntData As Variant) As Long

' Properly empty and deactivate a collection
Sub EmptyCollection(ByRef colData As Collection)

' Used to reseed Visual BASIC random number generator
Function RndSeed() As Double
```

```
' Swap data with each other.  Wrote this function since BASIC stopped
' having its own SWAP function.  Use this for swapping strings, type
' structures, numbers with decimal values, etc.
Sub SwapData(ByRef vntData1 As Variant, _
             ByRef vntData2 As Variant)

' Swap numeric data (byte, integer, or long) with each other
' without using a temporary holding variable.
Sub SwapLong(ByRef AA As Long, _
             ByRef BB As Long)

Sub SwapInt(ByRef AA As Integer, _
            ByRef BB As Integer)

Sub SwapBytes(ByRef AA As Byte, _
              ByRef BB As Byte)

' Converts a byte array to string data.
Function ByteArrayToString(ByRef abytData() As Byte) As String

' Converts string data to a byte array.
Function StringToByteArray(ByVal strData As String) As Byte()

' Creates a unique string of hex data using CryptoAPI hash functions.  Also,
' randomly select a starting position in hashed data string to capture two
' eight byte strings of data.  These will be converted into long integers
' for new carryover values.
Function CreateExtraSeed(Optional ByVal lngRetLength As Long = 0) As String
```

==========

Module: clsISAAC.cls

Description: ISAAC (Indirection, Shift, Accumulate, Add, and Count) generates 32-bit random numbers. Averaged out, it requires 18.75 machine cycles to generate each 32-bit value. Cycles are guaranteed to be at least 240 values long, and they are 28295 values long on average. The results are uniformly distributed, unbiased, and unpredictable unless you know the seed.

Reference: ISAAC Random number generator for Visual Basic 6.0 and VBA
by Kenneth Ives kenaso@tx.rr.com

Original C code by Bob Jenkins, March 1996
http://www.burtleburtle.net/bob/rand/isaacafa.html

Properties:

Version - Read only - String - Version information about this DLL

StopProcessing - Read/Write - Boolean - Switch that designates if the user opts to stop processing (True = Stop)

Methods:

```
' A quantity of random values will be generated based on the user request.
Function ISAAC_Prng(Optional ByVal lngArraySize As Long = 1, _
                    Optional ByVal blnReturnFloat As Boolean = True) As Variant
```

==========

Module: clsKISS

Description: The KISS generator, (Keep It Simple Stupid), is designed to combine the two multiply-with-carry generators in MWC with the 3-shift register SHR3 and the congruential generator CONG, using addition and exclusive-or. Has a period about 2^123.

The MWC generator concatenates two 16-bit multiply-with-carry generators, x(n)=36969x(n-1)+carry, y(n)=18000y(n-1)+carry mod 2^16, has period about 2^60 and seems to pass all tests of randomness. A favorite stand-alone generator---faster than KISS, which contains it.

SHR3 is a 3-shift-register generator with period 2^32-1. It uses y(n)=y(n-1)(I+L^17)(I+R^13)(I+L^5), with the y's viewed as binary vectors, L the 32x32 binary matrix that shifts a vector left 1, and R its transpose. SHR3 seems to pass all except those related to the binary rank test, since 32 successive values, as binary vectors, must be linearly independent, while 32 successive truly random 32-bit integers, viewed as binary vectors, will be linearly independent only about 29% of the time. The leading half of its 32 bits seem to pass tests, but bits in the last half are too regular.

SHR3 is a congruential generator with the widely used 69069 multiplier: x(n)=69069x(n-1)+1234567. It has period 2^32. The leading half of its 32 bits seem to pass tests, but bits in the last half are too regular.

CONG is a congruential generator with the widely used 69069 multiplier: x(n)=69069x(n-1)+1234567. It has a period 2^32. The leading half of its 32 bits seem to pass tests, but bits in the last half are too regular.

The generators MWC and KISS seem to pass all Diehard tests. By themselves, CONG and SHR3 do not.

References: KISS Random number generator for Visual Basic
by Kenneth Ives kenaso@tx.rr.com

Original code in C by George Marsaglia
http://www.ciphersbyritter.com/NEWS4/RANDC.HTM

George Marsaglia geo@stat.fsu.edu
http://stat.fsu.edu/pub/diehard/

Properties:

Version - Read only - String - Version information about this DLL

StopProcessing - Read/Write - Boolean - Switch that designates if the user opts to stop processing (True = Stop)

Methods:

```
' A quantity of random values will be generated based on the user request.
Function KISS_Prng(Optional ByVal lngArraySize As Long = 1, _
                   Optional ByVal blnReturnFloat As Boolean = True) As Variant
```

```
================================================================================
Module:         clsMWC

Description:    The MWC generator concatenates two 16-bit multiply-
                with-carry generators, x(n)=36969x(n-1)+carry,
                y(n)=18000y(n-1)+carry  mod 2^16, has period about
                2^60 and seems to pass all tests of randomness. A
                favorite stand-alone generator---faster than KISS,
                which contains it.

                The generators MWC and KISS seem to pass all Diehard
                tests.  By themselves, CONG and SHR3 do not.

References:     MWC Random number generator for Visual Basic 6.0
                by Kenneth Ives  kenaso@tx.rr.com

                Original code in C by George Marsaglia
                http://www.ciphersbyritter.com/NEWS4/RANDC.HTM

                George Marsaglia geo@stat.fsu.edu
                http://stat.fsu.edu/pub/diehard/

Properties:

    Version - Read only - String - Version information about this DLL

    StopProcessing - Read/Write - Boolean - Switch that designates if the user
              opts to stop processing (True = Stop)

Methods:

' A quantity of random values will be generated based on the user request.
Function MWC_Prng(Optional ByVal lngArraySize As Long = 1, _
                      Optional ByVal blnReturnFloat As Boolean = True) As Variant
```

==========

Module: clsMother (MOA = Mother-of-All)

Description: George Marsaglia's comments:

Yet another Random Number Generator

Random number generators are frequently posted on the network; my colleagues and I posted ULTRA in 1992 and, from the number of requests for releases to use it in software packages, it seems to be widely used.

I have long been interested in Random Number Generator's and several of my early ones are used as system generators or in statistical packages.

So why another one? And why here?

Because I want to describe a generator, or rather, a class of generators, so promising I am inclined to Call it

"The Mother-of-All Random Number Generators"

and because the generator seems promising enough to justify shortcutting the many months, even years, before new developments are widely known through publication in a journal.

This new class leads to simple, fast programs that produce sequences with very long periods. They use multiplication, which experience has shown does a better job of mixing bits than do +,- or exclusive-or, and they do it with easily implemented arithmetic modulo a power of 2, unlike arithmetic modulo a prime. The latter, while satisfactory, is difficult to implement. But the arithmetic here modulo 2^16 or 2^32 does not suffer the flaws of ordinary congruential generators for those moduli: trailing bit too regular. On the contrary, all bits of the integers produced by this new method, whether leading or trailing, have passed extensive tests of randomness.

Here is an idea of how it works, using, say, integers of six decimal digits from which we return random 3-digit integers. Start with n=123456, the seed.

Then form a new n=672*456+123=306555 and return 555.
Then form a new n=672*555+306=373266 and return 266.
Then form a new n=672*266+373=179125 and return 125,

and so on. Got it? This is a multiply-with-carry sequence x(n)=672*x(n-1)+carry mod b=1000, where the carry is the number of b's dropped in the modulus reduction. The resulting sequence of 3-digit x's has period 335,999. Try it.

No big deal, but that's just an example to give the idea. Now consider the sequence of 16-bit integers produced by the two C statements:

```
k=30903*(k&65535)+(k>>16); return(k&65535);
```

Notice that it is doing just what we did in the example:

multiply the bottom half (by 30903, carefully chosen), add the top half and return the new bottom.

That will produce a sequence of 16-bit integers with a period greater than 2^29, and if we concatenate two such:

```
k=30903*(k&65535)+(k>>16);
j=18000*(j&65535)+(j>>16);
return((k<<16)+j);
```

we get a sequence of more than 2^59 32-bit integers before cycling.

The following segment in a (properly initialized) C procedure will generate more than 2^118 32-bit random integers from six random seed values I , j, k, l, m, n:

```
k=30903*(k&65535)+(k>>16);
j=18000*(j&65535)+(j>>16);
i=29013*(i&65535)+(i>>16);
l=30345*(l&65535)+(l>>16);
m=30903*(m&65535)+(m>>16);
n=31083*(n&65535)+(n>>16);
return((k+i+m)>>16)+j+l+n);
```

And it will do it much faster than any of several widely used generators designed to use 16-bit integer arithmetic, such as that of Wichman-Hill that combines congruential sequences for three 15-bit primes (Applied Statistics, v31, p188-190, 1982), period about 2^42.

I call these multiply-with-carry generators. Here is an extravagant 16-bit example that is easily implemented in C or Fortran. It does such a thorough job of mixing the bits of the previous eight values that it is difficult to imagine a test of randomness it could not pass:

```
x[n]=12013x[n-8]+1066x[n-7]+1215x[n-6]+1492x[n-5]+1776x[n-4]
     +1812x[n-3]+1860x[n-2]+1941x[n-1]+carry mod 2^16.
```

The linear combination occupies at most 31 bits of a 32-bit integer. The bottom 16 is the output, the top 15 the next carry. It is probably best to implement with 8 case segments. It takes 8 microseconds on my PC. Of course it just provides 16-bit random integers, but awfully good ones. For 32 bits you would have to combine it with another, such as:

```
x[n]=9272x[n-8]+7777x[n-7]+6666x[n-6]+5555x[n-5]+4444x[n-4]
     +3333x[n-3]+2222x[n-2]+1111x[n-1]+carry mod 2^16.
```

Concatenating those two gives a sequence of 32-bit random integers (from 16 random 16-bit seeds), period about 2^250. It is so awesome it may merit the Mother of All Random Number Generator's title.

The coefficients in those two linear combinations suggest that it is easy to get long-period sequences, and that is true. The result is due to Cemal Kac, who extended the theory we gave for add-with-carry sequences: Choose a base b and give r seed values x[1],...,x[r] and an initial 'carry' c. Then the multiply-with-carry sequence:

```
x[n]=a1*x[n-1]+a2*x[n-2]+...+ar*x[n-r]+carry mod b,
```

where the new carry is the number of b's dropped in the modulus reduction, will have period the order of b in the group of residues relatively prime to m=ar*b^r+...+a1b^1-1. Furthermore, the x's are, in reverse order, the digits in the expansion of k/m to the base b, for some 0<k<m.

In practice b=2^16 or b=2^32 allows the new integer and the new carry to be the bottom and top half of a 32- or 64-bit linear combination of 16-or 32-bit integers. And it is easy to find suitable m's if you have a primality test:

just search through candidate coefficients until you get an m that is a safeprime---both m and (m-1)/2 are prime. Then the period of the multiply-with-carry sequence will be the prime (m-1)/2. (It can't be m-1 because b=2^16 or 2^32 is a square.)

Here is an interesting simple MWC generator with period> 2^92, for 32-bit arithmetic:

```
x[n]=1111111464*(x[n-1]+x[n-2]) + carry mod 2^32.
```

Suppose you have functions, say top() and bot(), that give the top and bottom halves of a 64-bit result. Then, with initial 32-bit x, y and carry c, simple statements such as:

```
y = bot(1111111464 * (x + y) + c)
x = y
c = Top(y)
```

will, repeated, give over 2^92 random 32-bit y's.

Not many machines have 64 bit integers yet. But most assemblers for modern CPU's permit access to the top and bottom halves of a 64-bit product.

References: Mother_Of_All Random number generator for Visual Basic
by Kenneth Ives kenaso@tx.rr.com

George Marsaglia code in C
ftp://ftp.taygeta.com/pub/c/mother.c

George Marsaglia geo@stat.fsu.edu
http://stat.fsu.edu/pub/diehard/

Properties:

Version - Read only - String - Version information about this DLL

StopProcessing - Read/Write - Boolean - Switch that designates if the user opts to stop processing (True = Stop)

Methods:

```
' A quantity of random values will be generated based on the user request.
Function MOA_Prng(Optional ByVal lngArraySize As Long = 1, _
                  Optional ByVal blnReturnFloat As Boolean = True) As Variant
```

```
================================================================================
Module:        clsMT19937

               MT19937 is the Mersenne Twister algorithm.
               It has a seed value of 2^19937-1.

               This code has been modified for only two types of output.
                   Long Integer       -2147483648 to 2147483647
                   Double precision   -0.9999999999999 to 0.9999999999999

               For the unabridged VBA code, visit the Mersenne Twister Home Page
               http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
               Look under Mersenne Twister "Various versions>languages links>codes".

               I am using Pablo Ronchi's VBA code to create my version of the
               Mersenne Twister algorithm.

http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/VERSIONS/BASIC/basic.html

Properties:

    Version - Read only - String - Version information about this DLL

    StopProcessing - Read/Write - Boolean - Switch that designates if the user
              opts to stop processing (True = Stop)

Methods:

' A quantity of random values will be generated based on the user request.
Function MT_Prng(Optional ByVal lngArraySize As Long = 1, _
                 Optional ByVal blnReturnFloat As Boolean = True) As Variant
```

```
================================================================================
Module:         clsMT11231A

                MT11231A is a variation of the Mersenne Twister algorithm.
                It has a seed value of 2^11231-1.

                This code has been modified for only two types of output.
                    Long Integer       -2147483648 to 2147483647
                    Double precision   -0.9999999999999 to 0.9999999999999

                For the unabridged VBA code, visit the Mersenne Twister Home Page
                http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
                Look under Mersenne Twister "Various versions>languages links>codes".

                I am using Pablo Ronchi's VBA code to create my version of the
                Mersenne Twister algorithm.

http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/VERSIONS/BASIC/basic.html

                This is one of three additional versions of the Mersenne Twister
algorithm
                named MT11231A, MT11231B & TT800.

Reference:      The Mersenne Twister (variations)
                http://www.quadibloc.com/crypto/co4814.htm

Properties:

    Version - Read only - String - Version information about this DLL

    StopProcessing - Read/Write - Boolean - Switch that designates if the user
              opts to stop processing (True = Stop)

Methods:

' A quantity of random values will be generated based on the user request.
Function MTA_Prng(Optional ByVal lngArraySize As Long = 1, _
                  Optional ByVal blnReturnFloat As Boolean = True) As Variant
```

```
================================================================================
Module:         clsMT11231B

                MT11231B is a variation of the Mersenne Twister algorithm.
                It has a seed value of 2^11231-1.

                This code has been modified for only two types of output.
                    Long Integer       -2147483648 to 2147483647
                    Double precision   -0.9999999999999 to 0.9999999999999

                For the unabridged VBA code, visit the Mersenne Twister Home Page
                http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
                Look under Mersenne Twister "Various versions>languages links>codes".

                I am using Pablo Ronchi's VBA code to create my version of the
                Mersenne Twister algorithm.

http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/VERSIONS/BASIC/basic.html

                This is one of three additional versions of the Mersenne Twister
algorithm
                named MT11231A, MT11231B & TT800.

Reference:      The Mersenne Twister (variations)
                http://www.quadibloc.com/crypto/co4814.htm

Properties:

    Version - Read only - String - Version information about this DLL

    StopProcessing - Read/Write - Boolean - Switch that designates if the user
              opts to stop processing (True = Stop)

Methods:

' A quantity of random values will be generated based on the user request.
Function MTB_Prng(Optional ByVal lngArraySize As Long = 1, _
                  Optional ByVal blnReturnFloat As Boolean = True) As Variant
```

```
================================================================================
Module:        clsTT800

               TT800 is a smaller variation of the Mersenne Twister algorithm.
               It has a seed value of 2^800-1.

               This code has been modified for only two types of output.
                   Long Integer       -2147483648 to 2147483647
                   Double precision   -0.9999999999999 to 0.9999999999999

               For the unabridged VBA code, visit the Mersenne Twister Home Page
               http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
               Look under Mersenne Twister "Various versions>languages links>codes".

               I am using Pablo Ronchi's VBA code to create my version of the
               Mersenne Twister algorithm.

http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/VERSIONS/BASIC/basic.html

               This is one of three additional versions of the Mersenne Twister
algorithm
               named MT11231A, MT11231B & TT800.

Reference:     A C-program for TT800 : July 8th 1996 Version
               by M. Matsumoto, email: m-mat @ math.sci.hiroshima-u.ac.jp
               TT800 c source code, available from
               http://random.mat.sbg.ac.at/ftp/pub/data/tt800.c

               The Mersenne Twister (variations)
               http://www.quadibloc.com/crypto/co4814.htm

Properties:

    Version - Read only - String - Version information about this DLL

    StopProcessing - Read/Write - Boolean - Switch that designates if the user
              opts to stop processing (True = Stop)

Methods:

' A quantity of random values will be generated based on the user request.
Function TT800_Prng(Optional ByVal lngArraySize As Long = 1, _
                    Optional ByVal blnReturnFloat As Boolean = True) As Variant
```

==========
License

Kenneth Ives
kenaso@tx.rr.com

==========

Preamble

This License governs Your use of the Work. This License is intended to allow developers to use the Source Code and Executable Files provided as part of the Work in any application in any form.

The main points subject to the terms of the License are:

- Source Code and Executable Files can be used in commercial applications;
- Source Code and Executable Files can be redistributed; and
- Source Code can be modified to create derivative works.

No claim of suitability, guarantee, or any warranty whatsoever is provided. The software is provided "as-is".

This License is entered between You, the individual or other entity reading or otherwise making use of the Work licensed pursuant to this License and the individual or other entity which offers the Work under the terms of this License ("Author").

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CODE PROJECT OPEN LICENSE ("LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HEREIN, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE AUTHOR GRANTS YOU THE RIGHTS CONTAINED HEREIN IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS. IF YOU DO NOT AGREE TO ACCEPT AND BE BOUND BY THE TERMS OF THIS LICENSE, YOU CANNOT MAKE ANY USE OF THE WORK.

Definitions.

"Articles" means, collectively, all articles written by Author which describes how the Source Code and Executable Files for the Work may be used by a user.

"Author" means the individual or entity that offers the Work under the terms of this License.

"Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works.

"Executable Files" refer to the executables, binary files, configuration and any required data files included in the Work.

"Publisher" means the provider of the website, magazine, CD-ROM, DVD or other medium from or by which the Work is obtained by You.

"Source Code" refers to the collection of source code and configuration files used to create the Executable Files.

"Standard Version" refers to such a Work if it has not been modified, or has been modified in accordance with the consent of the Author, such consent being in the full discretion of the Author.

"Work" refers to the collection of files distributed by the Publisher, including the Source Code, Executable Files, binaries, data files, documentation, whitepapers and the Articles.

"You" is you, an individual or entity wishing to use the Work and exercise your rights under this License.

Fair Use/Fair Use Rights. Nothing in this License is intended to reduce, limit, or restrict any rights arising from fair use, fair dealing, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

License Grant. Subject to the terms and conditions of this License, the Author hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

You may use the standard version of the Source Code or Executable Files in Your own applications.

You may apply bug fixes, portability fixes and other modifications obtained from the Public Domain or from the Author. A Work modified in such a way shall still be considered the standard version and will be subject to this License.

You may otherwise modify Your copy of this Work (excluding the Articles) in any way to create a Derivative Work, provided that You insert a prominent notice in each changed file stating how, when and where You changed that file.

You may distribute the standard version of the Executable Files and Source Code or Derivative Work in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution.

The Articles discussing the Work published in any form by the author may not be distributed or republished without the Author's consent. The author retains copyright to any such Articles. You may use the Executable Files and Source Code pursuant to this License but you may not repost or republish or otherwise distribute or make available the Articles, without the prior written consent of the Author.

Any subroutines or modules supplied by You and linked into the Source Code or Executable Files this Work shall not be considered part of this Work and will not be subject to the terms of this License.

Patent License. Subject to the terms and conditions of this License, each Author hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, import, and otherwise transfer the Work.

Restrictions. The license granted in Section 3 above is expressly

made subject to and limited by the following restrictions:

You agree not to remove any of the original copyright, patent, trademark, and attribution notices and associated disclaimers that may appear in the Source Code or Executable Files.

You agree not to advertise or in any way imply that this Work is a product of Your own.

The name of the Author may not be used to endorse or promote products derived from the Work without the prior written consent of the Author.

You agree not to sell, lease, or rent any part of the Work.

You may distribute the Executable Files and Source Code only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy of the Executable Files or Source Code You distribute and ensure that anyone receiving such Executable Files and Source Code agrees that the terms of this License apply to such Executable Files and/or Source Code. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute the Executable Files or Source Code with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License.

You agree not to use the Work for illegal, immoral or improper purposes, or on pages containing illegal, immoral or improper material. The Work is subject to applicable export laws. You agree to comply with all such laws and regulations that may apply to the Work after Your receipt of the Work.

Representations, Warranties and Disclaimer. THIS WORK IS PROVIDED "AS IS", "WHERE IS" AND "AS AVAILABLE", WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS OR GUARANTEES. YOU, THE USER, ASSUME ALL RISK IN ITS USE, INCLUDING COPYRIGHT INFRINGEMENT, PATENT INFRINGEMENT, SUITABILITY, ETC. AUTHOR EXPRESSLY DISCLAIMS ALL EXPRESS, IMPLIED OR STATUTORY WARRANTIES OR CONDITIONS, INCLUDING WITHOUT LIMITATION, WARRANTIES OR CONDITIONS OF MERCHANTABILITY, MERCHANTABLE QUALITY OR FITNESS FOR A PARTICULAR PURPOSE, OR ANY WARRANTY OF TITLE OR NON-INFRINGEMENT, OR THAT THE WORK (OR ANY PORTION THEREOF) IS CORRECT, USEFUL, BUG-FREE OR FREE OF VIRUSES. YOU MUST PASS THIS DISCLAIMER ON WHENEVER YOU DISTRIBUTE THE WORK OR DERIVATIVE WORKS.

Indemnity. You agree to defend, indemnify and hold harmless the Author and the Publisher from and against any claims, suits, losses, damages, liabilities, costs, and expenses (including reasonable legal or attorneys' fees) resulting from or relating to any use of the Work by You.

Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL THE AUTHOR OR THE PUBLISHER BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK OR OTHERWISE, EVEN IF THE AUTHOR OR THE PUBLISHER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Termination.

This License and the rights granted hereunder will terminate automatically upon any breach by You of any term of this License. Individualss or entities who have received Derivative Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 6, 7, 8, 9, 10 and 11 will survive any termination of this License.

If You bring a copyright, trademark, patent or any other infringement claim against any contributor over infringements You claim are made by the Work, your License from such contributor to the Work ends automatically.

Subject to the above terms and conditions, this License is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, the Author reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

Publisher. The parties hereby confirm that the Publisher shall not, under any circumstances, be responsible for and shall not have any liability in respect of the subject matter of this License. The Publisher makes no warranty whatsoever in connection with the Work and shall not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. The Publisher reserves the right to cease making the Work available to You at any time without notice

Miscellaneous.

This License shall be governed by the laws of the location of the head office of the Author or if the Author is an individual, the laws of location of the principal place of residence of the Author.

If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this License, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

This License constitutes the entire agreement between the parties with respect to the Work licensed herein. There are no understandings, agreements or representations with respect to the Work not specified herein. The Author shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Author and You.

Kenneth Ives kenaso@tx.rr.com