

# *Random Architecture*

Version 2.0

By Gregor N. Purdy

Copyright (C) 1991, 1992 Contemporary Design Studios. All rights reserved.

---

The architecture of the Random system is really fairly simple. The main departure from typical systems is that Random doesn't force you to use any one random number generator. This allows you the freedom to choose the tradeoffs you need to make for a given application between speed and degree of randomness.

To allow you to use whatever pseudo-random number generation algorithm you choose, Random provides an abstract superclass called RandomEngine. Any time you instantiate a Random instance or an instance of one of its subclasses, you either specify a subclass of RandomEngine or RandomEngine itself as the class of generator to use, or you can specify an instance of one of these classes. Any operation you perform through the Random instance will then use the random number generator you provided.

The random class keeps two data buffers: One for bits and one for bytes. When the Random instance is initialized, the buffers are filled with the generator's output, whatever size that may be. Then, during usage, byte-sized chunks of random bits are used from the byte buffer, and bit-sized chunks are used from the bit buffer. This means that it is less expensive to ask for a boolean value than it is to ask for some 32-bit quantity, since the bit comes from the bit buffer, instead of the byte buffer.

Another feature of the Random class is that the percent method fills its result double with 52 random bits for the fraction, instead of starting with a random 32 bit quantity and dividing by the maximum 32 bit value. This provides finer granularity in random floating-point values than was previously available.

All classes included in the Random System are capable of archiving themselves, just as you would expect from any other AppKit-compatible classes. This allows you to save the entire state of your random number generation system, without regard for the details of any of the implementations.

The interface for the RandomEngine class is very simple, and you will find it easy to implement your own random number generators. All you do is implement four methods in your new class.

The class method **+unit** is used by Random to determine how large its buffers should be. Whatever the natural size of the value your generator makes (in multiples of whole bytes), just have +unit return it.

Implement the **±makeRandom:** instance method to do whatever is necessary to generate the next value in the sequence and stuff it into the buffer location provided.

Implement the **±read:** and **±write:** methods for archiving, so your class can save and restore its state.

---