# Deprecating `rand()` and Friends

*Document #:* WG21 N3775
*Date:* 2013-09-25
*Revises:* N3742
*Project:* JTC1.22.32 Programming Language C++
*Reply to:* Walter E. Brown <webrown.cpp@gmail.com>

## Contents



**Abstract**

This paper proposes to deprecate some **<cstdlib>** legacy interfaces in order to encourage programmers to migrate to the **<random>** component of the C++11 standard library.[1]

This proposal has been separated, at LEWG's request, from the others in [Bro13b] in order to advance only this part for early incorporation into C++14, leaving the remaining sections for a future Technical Specification and/or International Standard.

## 1 Proposal

*If a feature is not deprecated [I] don't see any point in not using it.*

— HARIHARAN SUBRAMANIAN

By common consensus at several consecutive WG21 meetings during which the C++11 random number facility was being discussed and shaped into its final form, it has for a number of years been the long-term plan to excise the legacy C random number facility from the **std** namespace. Indeed, obliquely acknowledging the quality[2,3] of C++11's **<random>** header, WG21 voted several years ago to insert a Note[4] into [c.math]/5 as a head start on this plan: "The random number generation . . . facilities in this standard are often preferable to **rand**."[5]

[1] Readers seeking greater familiarity with this component may find [Bro13a] to be a helpful source of background information and tutorial guidance with numerous usage examples.

[2] "[B]y and large, I think it's the best random number library design of all, by a mile. If I were a random number, I'd think I died and went to heaven" [Ale07].

[3] "The C++11 <random> is very STL-like in that it sets up requirements for random number generators. . . , and random distributions. . . , and then the client can mix and match the two. It's a really very cool design [Hin12].

[4] The language for this Note was proposed by Beman Dawes in [Daw08]; [Bec08] was the first Working Paper to incorporate it.

[5] See also Stephan T. Lavavej's recent talk, "**rand()** Considered Harmful," given at GoingNative 2013. Recorded on 2013-09-06; available at channel9.msdn.com/Events/GoingNative/2013/rand-Considered-Harmful.

We therefore now propose to execute the next step of this plan to discourage the use of the traditional C function **rand** as well as its associated seeding function **srand** and upper limit macro **RAND_MAX**.[6] In particular, we propose to begin this transition by formally deprecating:

- **rand**, **srand**, and **RAND_MAX** and
- algorithm **random_shuffle()** (keeping **shuffle**, however).

The rationale for deprecating **random_shuffle()** is that one overload is specified so as to depend on **rand**, while the other overload is specified so as to require a hard-to-produce distribution object from the user; such a distribution is already an implicit part of **shuffle**, which we retain.

## 2 Proposed wording

All proposed wording is relative to WG21 draft [DuT12]. It is recommended to apply the wording additions and ~~deletions~~ in the order shown. Editorial notes are displayed against a gray background.

(1) Create a new section in Annex D:

**D.x Rand** [depr.rand]

Use of function **rand**, function **srand**, and macro **RAND_MAX** is deprecated. [ *Note:* This deprecation holds in the global namespace as well as in namespace **std**. — *end note* ]

(2) Copy all of the current [alg.random.shuffle] to a new section in Annex D, applying to the copy the changes shown below.

**D.y Random shuffle** [depr.alg.random.shuffle]

The following templates are in addition to those specified in Clause [alg.random].

```
template<class RandomAccessIterator>
  void random_shuffle(RandomAccessIterator first, RandomAccessIterator last);

template<class RandomAccessIterator, class RandomNumberGenerator>
  void random_shuffle(RandomAccessIterator first, RandomAccessIterator last,
                      RandomNumberGenerator&& rand);

~~template<class RandomAccessIterator, class UniformRandomNumberGenerator>~~
~~  void shuffle(RandomAccessIterator first, RandomAccessIterator last,~~
~~                UniformRandomNumberGenerator&& g);~~
```

*Effects:* Permutes the elements in the range [**first**, **last**) such that each possible permutation of those elements has equal probability of appearance.

*Requires:* **RandomAccessIterator** shall satisfy the requirements of **ValueSwappable** (17.6.3.2). The random number generating function object **rand** shall have a return type that is convertible to **iterator_traits<RandomAccessIterator>::difference_type**, and the call **rand(n)** shall return a randomly chosen value in the interval [0, **n**), for **n** > 0 of type **iterator_traits<RandomAccessIterator>::difference_type**. ~~The type UniformRandomNumberGenerator shall~~

[6] These names are declared in the classic C header **<stdlib.h>** and the corresponding C++ header **<cstdlib>**.

~~meet the requirements of a uniform random number generator (26.5.1.3) type whose return type is convertible to `iterator_traits<RandomAccessIterator>::difference_type`.~~

*Complexity:* Exactly (**last** − **first**) − 1 swaps.

*Remarks:* To the extent that the implementation of these functions makes use of random numbers, the implementation shall use the following sources of randomness:

The underlying source of random numbers for the first form of the function is implementation-defined. An implementation may use the **rand** function from the standard C library.

In the second form of the function, the function object **rand** shall serve as the implementation's source of randomness.

~~In the third `shuffle` form of the function, the object `g` shall serve as the implementation's source of randomness.~~

(3) In the synopsis in [algorithms.general]:

- apply the comment ***//Deprecated*** to each of the two declarations of **random_shuffle**;
- at the Project Editor's discretion, append to these same declarations a cross-reference to the new Annex D section [depr.alg.random.shuffle]; and
- change the parameter name **rand** to **g** so as to make this declaration consistent with that in the later exposition of **shuffle**.

(4) Finally, excise vestiges of **std::random_shuffle** from [alg.random.shuffle] by adjusting as follows:

### 25.3.12 ~~Random s~~Shuffle [alg.~~random.~~shuffle]

```
~~template<class RandomAccessIterator>~~
~~  void random_shuffle(RandomAccessIterator first, RandomAccessIterator last);~~

~~template<class RandomAccessIterator, class RandomNumberGenerator>~~
~~  void random_shuffle(RandomAccessIterator first, RandomAccessIterator last,~~
~~                       RandomNumberGenerator&& rand);~~

template<class RandomAccessIterator, class UniformRandomNumberGenerator>
  void shuffle(RandomAccessIterator first, RandomAccessIterator last,
               UniformRandomNumberGenerator&& g);
```

*Effects:* Permutes the elements in the range [**first**, **last**) such that each possible permutation of those elements has equal probability of appearance.

*Requires:* **RandomAccessIterator** shall satisfy the requirements of **ValueSwappable** (17.6.3.2). ~~The random number generating function object `rand` shall have a return type that is convertible to `iterator_traits<RandomAccessIterator>::difference_type`, and the call `rand(n)` shall return a randomly chosen value in the interval~~ [0, **n**)~~, for n > 0 of type `iterator_traits<RandomAccessIterator>::difference_type`.~~ The type **UniformRandomNumberGenerator** shall meet the requirements of a uniform random number generator (26.5.1.3) type whose return type is convertible to **iterator_traits<RandomAccessIterator>::difference_type**.

*Complexity:* Exactly (**last** − **first**) − 1 swaps.

*Remarks:* To the extent that the implementation of ~~these~~ this function~~s~~ makes use of random numbers, ~~the implementation shall use the following sources of randomness:~~

~~The underlying source of random numbers for the first form of the function is implementation-defined. An implementation may use the `rand` function from the standard C library.~~

~~In the second form of the function, the function object `rand` shall serve as the implementation's source of randomness.~~

~~In the third `shuffle` form of the function,~~ the object **`g`** shall serve as the implementation's source of randomness.

## 3 Acknowledgments

Many thanks to the readers of early drafts of this paper for their thoughtful comments.

## 4 Bibliography

[Ale07] Andrei Alexandrescu: "Re: Conveniently generating random numbers with TR1 random." `comp.std.c++`, 2007-06-11.

[Bec08] Pete Becker: "Working Draft, Standard for Programming Language C++." ISO/IEC JTC1/SC22/WG21 document N2691 (post-Sophia mailing), 2008-06-27. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2691.pdf.

[Bro13a] Walter E. Brown: "Random Number Generation in C++11." ISO/IEC JTC1/SC22/WG21 document N3551 (pre-Bristol mailing), 2013-03-12. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3551.pdf.

[Bro13b] Walter E. Brown: "Three **`<random>`**-related Proposals, v2." ISO/IEC JTC1/SC22/WG21 document N3742 (pre-Chicago mailing), 2013-08-30. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3742.pdf.

[Daw08] Beman Dawes et al.: "Thread-Safety in the Standard Library (Rev 2)." ISO/IEC JTC1/SC22/WG21 document N2669 (post-Sophia mailing), 2008-06-13. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2669.pdf.

[DuT12] Stefanus Du Toit: "Working Draft, Standard for Programming Language C++." ISO/IEC JTC1/SC22/WG21 document N3691 (mid-Bristol/Chicago mailing), 2013-05-16. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2012/n3691.pdf.

[Hin12] Howard Hinnant: Untitled response to posted query. 2012-31-07. http://stackoverflow.com/questions/11717433/tutorial-or-example-code-for-extending-c11-random-with-generators-and-distribu.

## 5 Revision history

| Version | Date | Changes |
|---|---|---|
| 1 | 2013-09-25 | • Extracted text from N3742 and adapted to be self-contained.<br>• Tweaked as requested by LWG at Chicago: (a) included a bit more rationale re the deprecation of **`random_shuffle`**, (b) moved **`rand()`**'s deprecation to a new Annex D section, (c) deprecated **`rand`** and friends in the global namespace as well as in **`std`**,[7] and (d) annotated **`random_shuffle`** as deprecated.<br>• Added a Note of explanation to the **`rand`** deprecation wording.<br>• Added footnote referring to STL's recent talk.<br>• Published as N3775. |

[7] Thank you, LWG!