CLHEP implementation of the MixMax generator and integration in Geant4

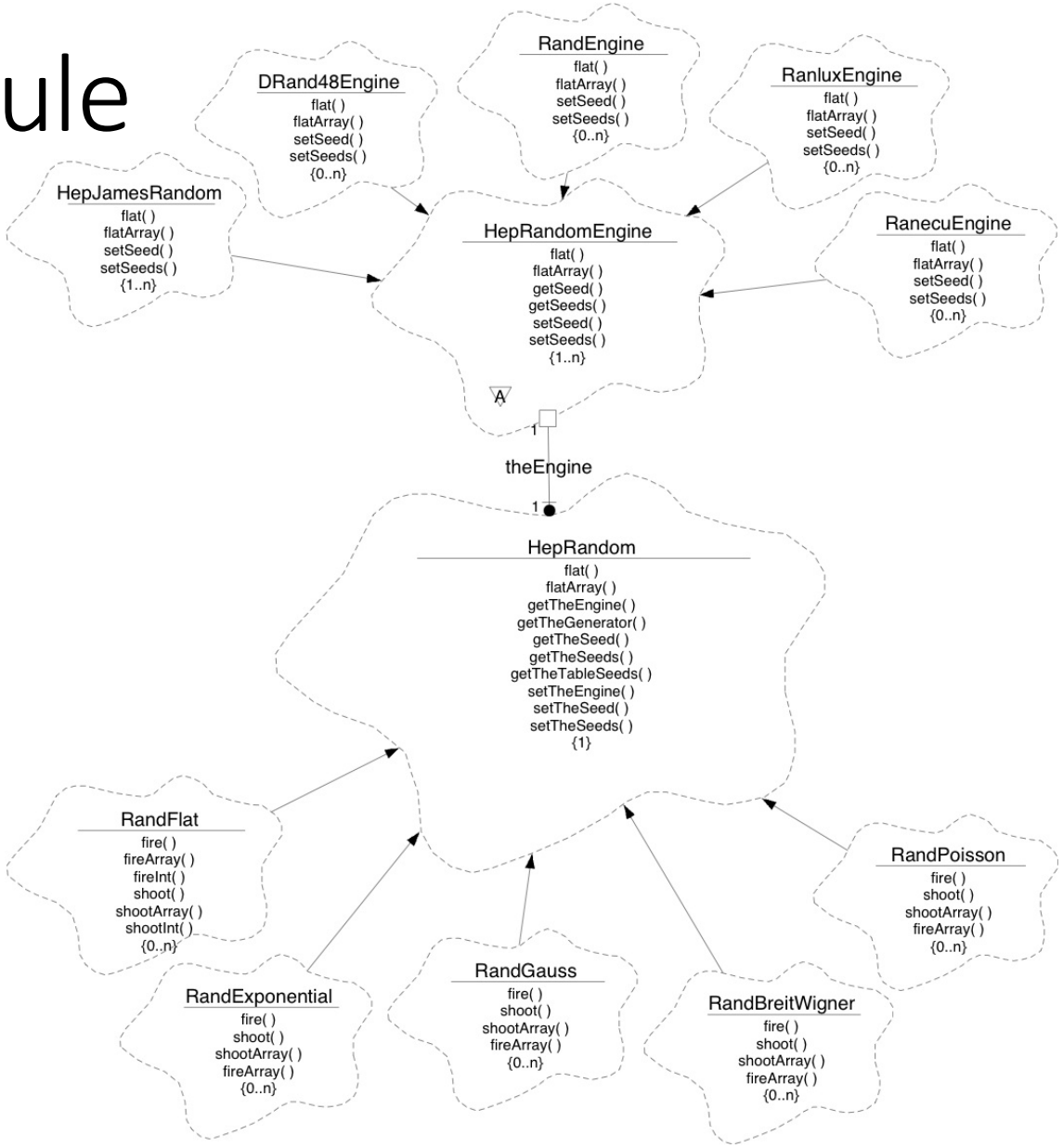
Gabriele Cosmo, CERN-EP/SFT

Contents

- The Random module in CLHEP
- Random engines available in CLHEP
- The testing suite
- MixMax in CLHEP, versions and current status

The CLHEP Random module

- Originally part of the Geant4 simulation toolkit
  - First implementation back to 1995
- Becomes part of CLHEP in 1997
  - http://cern.ch/clhep
- Set of C++ classes implementing
  - Random number generators (engines)
  - Random distribution classes
  - Common interfaces for seeding/shooting random values, saving/restoring engines' status, etc…



Random engines in CLHEP

- Groups 15 generator classes all written in C++
  - Most engines now rather obsolete…
    - working only on 32-bits precision
    - some could well be removed
    - others should be updated to more recent versions
  - Some of them now also available through the C++11 Standard or the GNU Scientific library
- Engines must obey to a well defined interface for
  - Seeding the generator (single/multiple seeds)
  - Saving/restoring/printing/streaming engines' internal status
  - Shooting random values (single/multiple values)
  - Engines can be statically and/or dynamically instantiated and used
    - Static interface can guarantee unique generation flow for reproducibility & thread-safe

The CLHEP testing suite for Random

- Set of fine-grained tests for provided interfaces
  - Sequence generation reproducibility (streams, files, engine's copies,…)
  - Reproducibility of distributions
  - Generation of vectors of numbers
  - Thread safety
- No tests for randomness quality

History of MixMax in CLHEP

- MixMax included since CLHEP version 2.3.1.0 (November 2015) – http://cern.ch/clhep
  - Implementation of MixMaxRng engine wrapper in C++
  - Based on mixmax v1.1 (N=256, s=4870132302560990 64, m=1)
  - Part of Geant4 release 10.2 series - http://cern.ch/geant4
  - Showing performance issues when setting seeds and shooting vectors of numbers (issues even more evident on multi-threaded applications)
- Corrections included in CLHEP version 2.3.3.0 (May 2016)
  - Based on mixmax v1.1 (N=17)
  - Corrected performance issues (thanks to smaller N and use of seed_spbox() for seeding MixMax with single seed); adopt simple loop for vector generation
  - Using wrong values for multiplier constants
- Corrected version included CLHEP version 2.3.4.0 (November 2016)
  - Based on mixmax v1.1 (N=17, s=0, m=2^36+1)
  - Included Geant4 release 10.3 series
- New MixMax C++ implementation included in CLHEP version 2.4.0.0 (November 2017)
  - Based on mixmax v2.0 (N=17, s=0, m=2^36+1)
  - CLHEP class MixMaxRng no longer a wrapper to C code; switched to become the DEFAULT engine
  - Included in Geant4 release 10.4 series as the DEFAULT random engine

Current MixMax in CLHEP

- MixMax included in CLHEP version 2.4.1.0 (June 2018)
  - https://gitlab.cern.ch/CLHEP/CLHEP/tags/CLHEP_2_4_1_0
  - Corrected issue for running on 32-bits platforms
  - Enabled use of thread-local storage on Windows platforms
  - Code ported on gcc-8.1.0 compiler
  - Part of Geant4 release 10.5-beta (https://github.com/Geant4/geant4/releases/tag/v10.5.0.beta)

The MixMaxRng class

Constructors & seeding:

- **`MixMaxRng(std::istream& is);`**
  - *Constructor taking a stream for the engine's state*
- **`MixMaxRng();`**
  - *Default destructor initialized on the engine instance number*
- **`MixMaxRng(long seed);`**
  - *Constructor initializing with a 64-bits seed*
- **`void setSeed(long seed, int dum=0);`**
  - *Sets the state of the algorithm according to a 64 bits seed*
- **`void setSeeds(const long* seeds, int seedNum=0);`**
  - *Sets the initial state of the engine according to the array of between one and four 32-bit seeds*
  - *If the size of long is greater on the platform, only the lower 32-bits are used*
  - *Streams created from seeds differing by at least one bit somewhere are guaranteed absolutely to be independent and non-colliding for at least the next 10^100 random numbers*

Shooting & status:

- **`double flat();`**
  - *Returns a pseudo random number between 0 and 1 (excluding the zero: in (0,1])*
- **`void flatArray (const int size, double* vect);`**
  - *Fills the array "vect" of specified size with flat random values*
- **`void saveStatus(const char filename[] = "MixMaxRngState.conf") const;`**
  - *Saves the the current engine state in the file given, by default MixMaxRngState.conf*
- **`void restoreStatus(const char filename[] = "MixMaxRngState.conf");`**
  - Reads a valid engine state from a given file, by default MixMaxRngState.conf and restores it
- **`void showStatus() const;`**
  - Dumps the engine status on the screen

Testing MixMax from CLHEP

- ✓ Passing all tests in the CLHEP testing suite
- ✓ Validated in Geant4 simulations through Geant4 testing suite
  - ✓ Including multi-threaded applications
- ✓ Smart seeding algorithm
- ✓ Excellent CPU performance
  - ❖ Shooting 1 billion numbers (sequential) grouped in vectors
    (MacOS 10.13.5, 2.8 GHz Intel Core i7)

HepJamesRamdom	- 32.49 s
Ranlux64Engine (luxury 2/3)	- 503.57 s
RanecuEngine	- 43.17s
RanluxEngine (luxury 4/5)	- 435.29 s
MTwistEngine	- 42.78 s
RanshiEngine	- 31.88 s
MixMaxRng	- 25.41 s

Next…

- Allow for run-time setting of N (keeping N=17 as the default)
  - Among the three “blessed” configurations: 8, 17 and 240
  - Instance of the engine parameterized on N
  - Skipping coefficients for N=8, 17, 240 already distributed in CLHEP
- Updated MixMax version
  - SIMD vectorized implementation?