# hipRAND Documentation

***Release 2.12.0***

**Advanced Micro Devices, Inc.**

**Jul 22, 2025**

# INSTALL

The hipRAND library is a wrapper library that lets you easily port NVIDIA CUDA applications that use the CUDA cuRAND library to the HIP layer. It sits between your application and the backend RAND library, where it marshals inputs to the backend and results to the application. hipRAND exports an interface that doesn't require the client to change, regardless of the chosen backend. It uses rocRAND in a ROCm environment and cuRAND in a CUDA environment and provides C, C++, and Python API wrappers.

The hipRAND public repository is located at https://github.com/ROCm/hipRAND.

Install

- *Installation guide*

How to

- *Use hipRAND interfaces*

Examples

- Examples

API reference

- *hipRAND data type support*
- *C/C++ API reference*
- *Python API reference*

To contribute to the documentation, see Contributing to ROCm.

You can find licensing information on the Licensing page.

CHAPTER

ONE

# INSTALLING AND BUILDING HIPRAND

To install hipRAND, choose between the following two methods:

- *Using prebuilt packages from the ROCm repositories*
- *Building from source*

## 1.1 Install using prebuilt packages

The prebuilt hipRAND packages require a ROCm-enabled platform. For information on installing ROCm, see the ROCm installation guide. After installing ROCm or enabling the ROCm repositories, use the system package manager to install hipRAND.

For Ubuntu and Debian:

```
sudo apt-get install hiprand
```

For CentOS-based systems:

```
sudo yum install hiprand
```

For SLES:

```
sudo dnf install hiprand
```

These commands install hipRAND in the `/opt/rocm` directory.

## 1.2 Build hipRAND from source

This section provides the information required to build hipRAND from source.

### 1.2.1 Requirements

To build hipRAND, CMake version 3.16 or later is required.

Additionally, to build hipRAND for the ROCm platform, the following components are required:

- ROCm Software (version 5.0.0 or later)
- rocRAND

To build hipRAND for the CUDA platform, the following applications are required:

- The CUDA toolkit (version 11.5.1 or newer)
- cuRAND (included in the CUDA Toolkit)

### 1.2.2 Downloading the source code

You can find the hipRAND source code in the hipRAND GitHub Repository. Use the branch that matches the ROCm version installed on the system. For example, on a system with ROCm 6.3 installed, use the following command to obtain the hipRAND version 6.3 source code:

```
git checkout -b release/rocm-rel-6.3 https://github.com/ROCm/hipRAND.git
```

### 1.2.3 Building the library

After obtaining the sources and dependencies, build hipRAND for ROCm software using the installation script:

```
cd hipRAND
./install --install
```

This automatically builds all required dependencies, excluding Git and the requirements listed above, and installs the project to `/opt/rocm`. For further information, run the `./install --help` command.

The clients, which are enabled using the `--clients` option, consist of the hipRAND tests and add the additional dependency of GoogleTest.

### 1.2.4 Building with CMake

For a more detailed installation process, build hipRAND manually using CMake. This enables certain configuration options that are not available through the `./install` script. To build hipRAND, use CMake with the following configuration:

```
cd hipRAND; mkdir build; cd build
# Configure the project
CXX=<compiler> cmake [options] ..
# Build
make -j$(nproc)
# Optionally, run the tests
ctest --output-on-failure
# Install
[sudo] make install
```

Where `<compiler>` should be set to `hipcc` or `amdclang` for ROCm or to a regular C++ compiler such as `g++` on a CUDA platform. The default build configuration is `Release`.

Here are the CMake options:

- `BUILD_WITH_LIB`: Determines whether to build hipRAND with the rocRAND or cuRAND backend. If it's set to `CUDA`, hipRAND is built using the cuRAND backend. Otherwise, the rocRAND backend is used.
- `BUILD_FORTRAN_WRAPPER`: Builds the Fortran wrapper when set to `ON`. Defaults to `OFF`.
- `BUILD_TEST`: Builds the hipRAND tests when set to `ON`. Defaults to `OFF`.
- `BUILD_BENCHMARK`: Builds the hipRAND benchmarks when set to `ON`. Defaults to `OFF`.
- `BUILD_ADDRESS_SANITIZER`: Builds with address sanitization enabled when set to `ON`. Defaults to `OFF`.
- `ROCRAND_PATH`: Specifies a rocRAND install other than the default system installed version.
- `DOWNLOAD_ROCRAND`: Downloads and installs rocRAND in the build directory when set to `ON`. Defaults to `OFF`.
- `DEPENDENCIES_FORCE_DOWNLOAD`: Downloads and builds the dependencies instead of using the system-installed dependencies when set to `ON`. Defaults to `OFF`.

If you are using `ROCRAND_PATH` or `DOWNLOAD_ROCRAND` when rocRAND is already installed in the default location, you must use the `CMAKE_NO_SYSTEM_FROM_IMPORTED=ON` option to configure the project. Failing to do so might process the rocRAND headers from the system-installed version instead of the specified version, leading to errors or missing functionality.

#### Common build errors

Use the following tips to troubleshoot build problems.

- `rocrand` package configuration file not found:

```
Could not find a package configuration file provided by "rocrand" with any of the␣
↪following names:

rocrandConfig.cmake
rocrand-config.cmake
```

  **Solution**: Install rocRAND.

- `ROCM` package configuration file not found:

```
Could not find a package configuration file provided by "ROCM" with any of the␣
↪following names:

ROCMConfig.cmake
rocm-config.cmake
```

  **Solution**: Install the ROCm CMake modules.

## 1.3 Building the Python API wrapper

This section provides the information required to build the hipRAND Python API wrapper.

### 1.3.1 Requirements

The hipRAND Python API Wrapper requires the following dependencies:

- hipRAND
- Python 3.5
- NumPy (This is installed automatically as a dependency, if necessary.)

**Note**

If hipRAND is built from source but is either not installed or installed in a non-standard directory, then set the `ROCRAND_PATH` or `HIPRAND_PATH` environment variable to the path containing `libhiprand.so` as shown below:

```
export HIPRAND_PATH=~/hipRAND/build/library/
```

### 1.3.2 Installation

To install the Python hipRAND module using `pip`, run these commands:

```
cd hipRAND/python/hiprand
pip install .
```

Use these commands to run the tests:

```
cd hipRAND/python/hiprand
python tests/hiprand_test.py
```

CHAPTER

# TWO

# USING HIPRAND INTERFACES

The hipRAND interface is compatible with the rocRAND and the NVIDIA CUDA cuRAND-v2 APIs. Porting a CUDA application that calls the cuRAND API to an application that calls the hipRAND API is relatively straightforward.

## 2.1 Host API

For example, to create a generator, follow this example:

```
hiprandStatus_t
hiprandCreateGenerator(
   hiprandGenerator_t* generator,
   hiprandRngType_t rng_type
)
```

## 2.2 Device API

Here is an example that generates a log-normally distributed float from a generator. These functions are templated for all generators.

```
__device__ double
hiprand_log_normal_double(
   hiprandStateSobol64_t* state,
   double mean,
   double stddev
)
```

CHAPTER

THREE

# HIPRAND DATA TYPE SUPPORT

The data type support in hipRAND is similar to NVIDIA CUDA cuRAND. On AMD hardware, the backend is provided by rocRAND. To see the data type comparison between rocRAND and cuRAND, see rocRAND data type support.

You can find a summary of the ROCm data type support at Supported data types in ROCm.

CHAPTER

FOUR

# C/C++ API REFERENCE

This document describes the hipRAND APIs available in C and C++.

## 4.1 Device functions

*group* **hipRAND device functions**

### Defines

**HIPRAND_PHILOX4x32_DEFAULT_SEED**

Default seed for PHILOX4x32 PRNG.

**HIPRAND_XORWOW_DEFAULT_SEED**

Default seed for XORWOW PRNG.

**HIPRAND_MRG32K3A_DEFAULT_SEED**

Default seed for MRG32K3A PRNG.

**HIPRAND_MTGP32_DEFAULT_SEED**

Default seed for MTGP32 PRNG.

**HIPRAND_MT19937_DEFAULT_SEED**

Default seed for MT19937 PRNG.

### Typedefs

typedef struct hiprandStateXORWOW **hiprandStateXORWOW_t**

XORWOW PRNG state type.

typedef *hiprandStateXORWOW_t* **hipRandState_t**

Deprecated alias of hiprandStateXORWOW_t.

*Deprecated:*

Please use hiprandStateXORWOW_t directly.

typedef struct hiprandStatePhilox4_32_10 **hiprandStatePhilox4_32_10_t**

PHILOX PRNG state type.

typedef struct hiprandStateMRG32k3a **hiprandStateMRG32k3a_t**

MRG32k3a PRNG state type.

typedef struct hiprandStateMtgp32 **hiprandStateMtgp32_t**

MTGP32 PRNG state type.

typedef struct hiprandStateScrambledSobol32 **hiprandStateScrambledSobol32_t**

Scrambled SOBOL32 QRNG state type.

typedef struct hiprandStateScrambledSobol64 **hiprandStateScrambledSobol64_t**

Scrambled SOBOL64 QRNG state type.

typedef struct hiprandStateSobol32 **hiprandStateSobol32_t**

SOBOL32 QRNG state type.

typedef struct hiprandStateSobol64 **hiprandStateSobol64_t**

SOBOL64 QRNG state type.

typedef mtgp32_params **mtgp32_kernel_params_t**

Opaque MTGP32 PRNG state parameters in the device format.

typedef struct mtgp32_params_fast **mtgp32_params_fast_t**

Opaque MTGP32 PRNG state parameters in the host format.

typedef mtgp32_fast_params **mtgp32_fast_param_t**

Deprecated alias of mtgp32_params_fast_t.

*Deprecated:*

please use mtgp32_params_fast_t directly.

typedef unsigned int **hiprandDirectionVectors32_t**[32]

Convenience typedef for Sobol 32 direction vector array.

typedef unsigned long long **hiprandDirectionVectors64_t**[64]

Convenience typedef for Sobol 64 direction vector array.

### Functions

void **hiprand_mtgp32_block_copy**(*hiprandStateMtgp32_t* *src, *hiprandStateMtgp32_t* *dest)

Copy MTGP32 state to another state using block of threads.

Copies a MTGP32 state `src` to `dest` using a block of threads efficiently. Example usage would be:

```
__global__
void generate_kernel(hiprandStateMtgp32_t * states, unsigned int * output,␣
↪const size_t size)
{
    const unsigned int state_id = hipBlockIdx_x;
    unsigned int index = hipBlockIdx_x * hipBlockDim_x + hipThreadIdx_x;
    unsigned int stride = hipGridDim_x * hipBlockDim_x;

    __shared__ GeneratorState state;
    hiprand_mtgp32_block_copy(&states[state_id], &state);

    while(index < size)
    {
        output[index] = rocrand(&state);
        index += stride;
    }

    hiprand_mtgp32_block_copy(&state, &states[state_id]);
}
```

**Parameters**

- **src** – - Pointer to a state to copy from
- **dest** – - Pointer to a state to copy to

void **hiprand_mtgp32_set_params**(*hiprandStateMtgp32_t* *state, *mtgp32_kernel_params_t* *params)

Changes parameters of a MTGP32 state.

**Parameters**

- **state** – - Pointer to a MTGP32 state
- **params** – - Pointer to new parameters

template<class **StateType**>
void **hiprand_init**(const unsigned long long seed, const unsigned long long subsequence, const unsigned long long offset, *StateType* *state)

Initializes a PRNG state.

> **Note**
>
> MTGP32 must be initialized from the host using *hiprandMakeMTGP32KernelState()*.

**Template Parameters**

**StateType** – - Pseudorandom number generator state type. `StateType` type must be one of following types: *hiprandStateXORWOW_t*, *hiprandStatePhilox4_32_10_t* or *hiprandStateMRG32k3a_t*

**Parameters**

- **seed** – - Pseudorandom number generator's seed
- **subsequence** – - Number of subsequence to skipahead

- **offset** – - Absolute subsequence offset, i.e. how many states from current subsequence should be skipped
- **state** – - Pointer to a state to initialize

void **hiprand_init**(*hiprandDirectionVectors32_t* direction_vectors, unsigned int offset, *hiprandStateSobol32_t* *state)

Initializes a Sobol32 state.

**Parameters**

- **direction_vectors** – - Pointer to array of 32 `unsigned int`s that represent the direction numbers.
- **offset** – - Absolute subsequence offset, i.e. how many states should be skipped.
- **state** – - Pointer to a state to initialize.

void **hiprand_init**(*hiprandDirectionVectors32_t* direction_vectors, unsigned int scramble_constant, unsigned int offset, *hiprandStateScrambledSobol32_t* *state)

Initializes a ScrambledSobol32 state.

**Parameters**

- **direction_vectors** – - Pointer to array of 32 `unsigned int`s that represent the direction numbers.
- **scramble_constant** – - Constant used for scrambling the sequence.
- **offset** – - Absolute subsequence offset, i.e. how many states should be skipped.
- **state** – - Pointer to a state to initialize.

void **hiprand_init**(*hiprandDirectionVectors64_t* direction_vectors, unsigned int offset, *hiprandStateSobol64_t* *state)

Initializes a Sobol64 state.

**Parameters**

- **direction_vectors** – - Pointer to array of 64 `unsigned long long int`s that represent the direction numbers.
- **offset** – - Absolute subsequence offset, i.e. how many states should be skipped.
- **state** – - Pointer to a state to initialize.

void **hiprand_init**(*hiprandDirectionVectors64_t* direction_vectors, unsigned long long int scramble_constant, unsigned int offset, *hiprandStateScrambledSobol64_t* *state)

Initializes a ScrambledSobol64 state.

**Parameters**

- **direction_vectors** – - Pointer to array of 64 `unsigned long long int`s that represent the direction numbers.
- **scramble_constant** – - Constant used for scrambling the sequence.
- **offset** – - Absolute subsequence offset, i.e. how many states should be skipped.
- **state** – - Pointer to a state to initialize.

template<class **StateType**>

void **skipahead**(unsigned long long n, *StateType* *state)

Updates RNG state skipping `n` states ahead.

**Template Parameters**

**StateType** – - Random number generator state type. `StateType` type must be one of following types: *hiprandStateXORWOW_t*, *hiprandStatePhilox4_32_10_t*, *hiprandStateMRG32k3a_t*, *hiprandStateSobol32_t*, *hiprandStateScrambledSobol32_t*, *hiprandStateSobol64_t* or *hiprandStateScrambledSobol64_t*.

**Parameters**

- **n** – - Number of states to skipahead
- **state** – - Pointer to a state to modify

template<class **StateType**>
void **skipahead_sequence**(unsigned long long n, *StateType* *state)

Updates PRNG state skipping `n` sequences ahead.

| PRNG | Sequence size [Number of elements] |
|---|---|
| XORWOW | $2^{67}$ |
| Philox | $4 \times 2^{64}$ |
| MRG32k3a | $2^{67}$ |

**Template Parameters**

**StateType** – - Random number generator state type. `StateType` type must be one of following types: *hiprandStateXORWOW_t*, *hiprandStatePhilox4_32_10_t*, or *hiprandStateMRG32k3a_t*

**Parameters**

- **n** – - Number of subsequences to skipahead
- **state** – - Pointer to a state to update

template<class **StateType**>
void **skipahead_subsequence**(unsigned long long n, *StateType* *state)

Updates PRNG state skipping `n` subsequences ahead.

| PRNG | Subsequence size [Number of elements] |
|---|---|
| XORWOW | $2^{67}$ |
| Philox | $4 \times 2^{64}$ |
| MRG32k3a | $2^{127}$ |

**Template Parameters**

**StateType** – - Random number generator state type. `StateType` type must be one of following types: *hiprandStateXORWOW_t*, *hiprandStatePhilox4_32_10_t* or *hiprandStateMRG32k3a_t*

**Parameters**

- **n** – - Number of subsequences to skipahead

- **state** – - Pointer to a state to update

template<class **StateType**>
unsigned int **hiprand**(*StateType* *state)

Generates uniformly distributed random `unsigned int` from [0; 2^32 - 1] range.

**Template Parameters**
**StateType** – - Random number generator state type. `StateType` type must be one of following types: *hiprandStateXORWOW_t*, *hiprandStatePhilox4_32_10_t*, *hiprandStateMRG32k3a_t*, *hiprandStateMtgp32_t*, *hiprandStateSobol32_t*, *hiprandStateScrambledSobol32_t*.

**Parameters**
**state** – - Pointer to a RNG state to use

**Returns**
Uniformly distributed random 32-bit `unsigned int`

uint4 **hiprand4**(*hiprandStatePhilox4_32_10_t* *state)

Generates four uniformly distributed random `unsigned int`s from [0; 2^32 - 1] range.

**Parameters**
**state** – - Pointer to a Philox state to use

**Returns**
Four uniformly distributed random 32-bit `unsigned int`s as `uint4`

template<class **StateType**>
unsigned long long int **hiprand_long_long**(*StateType* *state)

Generates uniformly distributed random `unsigned long long int` from [0; 2^64 - 1] range.

**Template Parameters**
**StateType** – - Random number generator state type. `StateType` type must be one of the following types: *hiprandStateSobol64_t* or *hiprandStateScrambledSobol64_t*.

**Parameters**
**state** – - Pointer to a RNG state to use

**Returns**
Uniformly distributed random 64-bit `unsigned long long int`

template<class **StateType**>
float **hiprand_uniform**(*StateType* *state)

Generates uniformly distributed random `float` value from (0; 1] range.

**Template Parameters**
**StateType** – - Random number generator state type.

**Parameters**
**state** – - Pointer to a RNG state to use

**Returns**
Uniformly distributed random `float` value

float4 **hiprand_uniform4**(*hiprandStatePhilox4_32_10_t* *state)

Generates four uniformly distributed random `float` value from (0; 1] range.

**Parameters**
**state** – - Pointer to a Philox state to use

**Returns**
Four uniformly distributed random `float` values as `float4`

template<class **StateType**>
double **hiprand_uniform_double**(*StateType* *state)

Generates uniformly distributed random `double` value from (0; 1] range.

> **Note**
>
> When `state` is of type: *hiprandStateMRG32k3a_t*, *hiprandStateMtgp32_t*, *hiprandStateSobol32_t* or *hiprandStateScrambledSobol32_t* then the returned `double` value is generated using only 32 random bits (one `unsigned int` value). In case of the Sobol types, this is done to guarantee the quasi-random properties.

**Template Parameters**
**StateType** – - Random number generator state type.

**Parameters**
**state** – - Pointer to a RNG state to use

**Returns**
Uniformly distributed random `double` value

double2 **hiprand_uniform2_double**(*hiprandStatePhilox4_32_10_t* *state)

Generates two uniformly distributed random `double` values from (0; 1] range.

**Parameters**
**state** – - Pointer to a Philox state to use

**Returns**
Two uniformly distributed random `double` values as `double2`

double4 **hiprand_uniform4_double**(*hiprandStatePhilox4_32_10_t* *state)

Generates four uniformly distributed random `double` values from (0; 1] range.

**Parameters**
**state** – - Pointer to a Philox state to use

**Returns**
Four uniformly distributed random `double` values as `double4`

template<class **StateType**>
float **hiprand_normal**(*StateType* *state)

Generates normally distributed random `float` value.

Mean value of normal distribution is equal to 0.0, and standard deviation equals 1.0.

**Template Parameters**
**StateType** – - Random number generator state type.

**Parameters**
**state** – - Pointer to a RNG state to use

**Returns**
Normally distributed random `float` value

template<class **StateType**>

float2 **hiprand_normal2**(*StateType* *state)

Generates two normally distributed random `float` values.

Mean value of normal distribution is equal to 0.0, and standard deviation equals 1.0.

**Template Parameters**
**StateType** – - Random number generator state type. `StateType` type must be one of following types: *hiprandStateXORWOW_t*, *hiprandStatePhilox4_32_10_t*, or *hiprandStateMRG32k3a_t*

**Parameters**
**state** – - Pointer to a RNG state to use

**Returns**
Two normally distributed random `float` values as `float2`

float4 **hiprand_normal4**(*hiprandStatePhilox4_32_10_t* *state)

Generates four normally distributed random `float` values.

Mean value of normal distribution is equal to 0.0, and standard deviation equals 1.0.

**Parameters**
**state** – - Pointer to a Philox state to use

**Returns**
Four normally distributed random `float` values as `float4`

template<class **StateType**>
double **hiprand_normal_double**(*StateType* *state)

Generates normally distributed random `double` value.

Mean value of normal distribution is equal to 0.0, and standard deviation equals 1.0.

**Template Parameters**
**StateType** – - Random number generator state type.

**Parameters**
**state** – - Pointer to a RNG state to use

**Returns**
Normally distributed random `double` value

template<class **StateType**>
double2 **hiprand_normal2_double**(*StateType* *state)

Generates two normally distributed random `double` values.

Mean value of normal distribution is equal to 0.0, and standard deviation equals 1.0.

**Template Parameters**
**StateType** – - Random number generator state type. `StateType` type must be one of following types: *hiprandStateXORWOW_t*, *hiprandStatePhilox4_32_10_t*, or *hiprandStateMRG32k3a_t*

**Parameters**
**state** – - Pointer to a RNG state to use

**Returns**
Two normally distributed random `double` values as `double2`

double4 **hiprand_normal4_double**(*hiprandStatePhilox4_32_10_t* *state)

Generates four normally distributed random `double` values.

Mean value of normal distribution is equal to 0.0, and standard deviation equals 1.0.

**Parameters**
**state** – - Pointer to a Philox state to use

**Returns**
Four normally distributed random `double` values as `double4`

template<class **StateType**>
float **hiprand_log_normal**(*StateType* *state, float mean, float stddev)

Generates log-normally distributed random `float` value.

**Template Parameters**
**StateType** – - Random number generator state type.

**Parameters**

- **state** – - Pointer to a RNG state to use
- **mean** – - Mean value of log-normal distribution
- **stddev** – - Standard deviation value of log-normal distribution

**Returns**
Log-normally distributed random `float` value

template<class **StateType**>
float2 **hiprand_log_normal2**(*StateType* *state, float mean, float stddev)

Generates two log-normally distributed random `float` values.

**Template Parameters**
**StateType** – - Random number generator state type. `StateType` type must be one of following types: *hiprandStateXORWOW_t*, *hiprandStatePhilox4_32_10_t*, or *hiprandStateMRG32k3a_t*

**Parameters**

- **state** – - Pointer to a RNG state to use
- **mean** – - Mean value of log-normal distribution
- **stddev** – - Standard deviation value of log-normal distribution

**Returns**
Two log-normally distributed random `float` values as `float2`

float4 **hiprand_log_normal4**(*hiprandStatePhilox4_32_10_t* *state, float mean, float stddev)

Generates four log-normally distributed random `float` values.

**Parameters**

- **state** – - Pointer to a Philox state to use
- **mean** – - Mean value of log-normal distribution
- **stddev** – - Standard deviation value of log-normal distribution

**Returns**
Four log-normally distributed random `float` values as `float4`

template<class **StateType**>
double **hiprand_log_normal_double**(*StateType* *state, double mean, double stddev)

Generates log-normally distributed random `double` value.

**Template Parameters**
**StateType** – - Random number generator state type.

**Parameters**

- **state** – - Pointer to a RNG state to use
- **mean** – - Mean value of log-normal distribution
- **stddev** – - Standard deviation value of log-normal distribution

**Returns**
Log-normally distributed random `double` value

template<class **StateType**>
double2 **hiprand_log_normal2_double**(*StateType* *state, double mean, double stddev)

Generates two log-normally distributed random `double` values.

**Template Parameters**
**StateType** – - Random number generator state type. `StateType` type must be one of following types: *hiprandStateXORWOW_t*, *hiprandStatePhilox4_32_10_t*, *hiprandStateMRG32k3a_t* or hiprandStateMtgp32_t.

**Parameters**

- **state** – - Pointer to a RNG state to use
- **mean** – - Mean value of log-normal distribution
- **stddev** – - Standard deviation value of log-normal distribution

**Returns**
Two log-normally distributed random `double` values as `double2`

double4 **hiprand_log_normal4_double**(*hiprandStatePhilox4_32_10_t* *state, double mean, double stddev)

Generates four log-normally distributed random `double` values.

**Parameters**

- **state** – - Pointer to a Philox state to use
- **mean** – - Mean value of log-normal distribution
- **stddev** – - Standard deviation value of log-normal distribution

**Returns**
Four log-normally distributed random `double` values as `double4`

template<class **StateType**>
unsigned int **hiprand_poisson**(*StateType* *state, double lambda)

Generates Poisson-distributed random `unsigned int` value.

**Template Parameters**
**StateType** – - Random number generator state type.

**Parameters**

- **state** – - Pointer to a RNG state to use
- **lambda** – - Lambda (mean) parameter of Poisson distribution

**Returns**
Poisson-distributed random `unsigned int` value

uint4 **hiprand_poisson4**(*hiprandStatePhilox4_32_10_t* *state, double lambda)

Generates four Poisson-distributed random `unsigned int` values.

**Parameters**

- **state** – - Pointer to a Philox state to use
- **lambda** – - Lambda (mean) parameter of Poisson distribution

**Returns**
Four Poisson-distributed random `unsigned int` values as `uint4`

template<class **StateType**>
unsigned int **hiprand_discrete**(*StateType* *state, hiprandDiscreteDistribution_t discrete_distribution)

Generates random `unsigned int` value according to given discrete distribution.

See also: *hiprandCreatePoissonDistribution()*

**Template Parameters**
**StateType** – - Random number generator state type.

**Parameters**

- **state** – - Pointer to a RNG state to use
- **discrete_distribution** – - Discrete distribution

**Returns**
Random `unsigned int` value

uint4 **hiprand_discrete4**(*hiprandStatePhilox4_32_10_t* *state, hiprandDiscreteDistribution_t discrete_distribution)

Generates four random `unsigned int` values according to given discrete distribution.

See also: *hiprandCreatePoissonDistribution()*

**Parameters**

- **state** – - Pointer to a Philox state to use
- **discrete_distribution** – - Discrete distribution

**Returns**
Four random `unsigned int` values as `uint4`

inline *hiprandStatus_t* **hiprandMakeMTGP32Constants**(const *mtgp32_params_fast_t* params[], *mtgp32_kernel_params_t* *p)

Loads parameters for MTGP32.

Loads parameters for use by kernel functions on the host-side and copies the results to the specified location in device memory.

**Parameters**

- **params** – **[in]** Pointer to an array of type mtgp32_params_fast_t allocated in host memory
- **p** – **[out]** Pointer to a mtgp32_kernel_params_t structure allocated in device memory

**Returns**

- HIPRAND_STATUS_ALLOCATION_FAILED if parameters could not be loaded
- HIPRAND_STATUS_SUCCESS if parameters are loaded

inline *hiprandStatus_t* **hiprandMakeMTGP32KernelState**(*hiprandStateMtgp32_t* *s, *mtgp32_params_fast_t* params[], *mtgp32_kernel_params_t* *k, int n, unsigned long long seed)

Initializes MTGP32 states.

Initializes MTGP32 states on the host-side by allocating a state array in host memory, initializes that array, and copies the result to device memory.

- On the cuRAND backend `k` must be initialized by calling *hiprandMakeMTGP32Constants()* with the the same `params`
- On the rocRAND backend `k` is ignored

For maximum portability application should initialize `k` as required by cuRAND, but when only targeting the rocRAND backend it may be left uninitialized.

**Parameters**

- **s** – **[out]** Pointer to an array of states in device memory
- **params** – **[in]** Pointer to an array of type mtgp32_params_fast_t in host memory
- **k** – **[in]** Pointer to a mtgp32_kernel_params_t structure allocated in device memory
- **n** – Number of states to initialize
- **seed** – Seed value

**Returns**

- HIPRAND_STATUS_ALLOCATION_FAILED if states could not be initialized
- HIPRAND_STATUS_SUCCESS if states are initialized

## 4.2 C host API

*group* **hipRAND host API**

### Defines

**HIPRAND_VERSION**

hipRAND library version

Version number may not be visible in the documentation.

`HIPRAND_VERSION % 100` is the patch level, `HIPRAND_VERSION / 100 % 1000` is the minor version, `HIPRAND_VERSION / 100000` is the major version.

For example, if `HIPRAND_VERSION` is `100500`, then the major version is `1`, the minor version is `5`, and the patch level is `0`.

**HIPRAND_DEFAULT_MAX_BLOCK_SIZE**

**HIPRAND_DEFAULT_MIN_WARPS_PER_EU**

### Typedefs

typedef enum *hiprandStatus* **hiprandStatus_t**

hipRAND function call status type

typedef enum *hiprandRngType* **hiprandRngType_t**

hipRAND generator type

typedef enum *hiprandOrdering* **hiprandOrdering_t**

hipRAND generator ordering

typedef enum *hiprandDirectionVectorSet* **hiprandDirectionVectorSet_t**

hipRAND vector set for quasirandom generators.

### Enums

enum **hiprandStatus**

hipRAND function call status type

*Values:*

enumerator **HIPRAND_STATUS_SUCCESS**

Success.

enumerator **HIPRAND_STATUS_VERSION_MISMATCH**

Header file and linked library version do not match.

enumerator **HIPRAND_STATUS_NOT_INITIALIZED**

Generator not created.

enumerator **HIPRAND_STATUS_ALLOCATION_FAILED**

Memory allocation failed.

enumerator **HIPRAND_STATUS_TYPE_ERROR**

Generator type is wrong.

enumerator **HIPRAND_STATUS_OUT_OF_RANGE**

Argument out of range.

enumerator **HIPRAND_STATUS_LENGTH_NOT_MULTIPLE**

Requested size is not a multiple of quasirandom generator's dimension, or requested size is not even (see *hiprandGenerateNormal()*), or pointer is misaligned (see *hiprandGenerateNormal()*)

enumerator **HIPRAND_STATUS_DOUBLE_PRECISION_REQUIRED**

GPU does not have double precision.

enumerator **HIPRAND_STATUS_LAUNCH_FAILURE**

Kernel launch failure.

enumerator **HIPRAND_STATUS_PREEXISTING_FAILURE**

Preexisting failure on library entry.

enumerator **HIPRAND_STATUS_INITIALIZATION_FAILED**

Initialization of HIP failed.

enumerator **HIPRAND_STATUS_ARCH_MISMATCH**

Architecture mismatch, GPU does not support requested feature.

enumerator **HIPRAND_STATUS_INTERNAL_ERROR**

Internal library error.

enumerator **HIPRAND_STATUS_NOT_IMPLEMENTED**

Feature not implemented yet.

enum **hiprandRngType**

hipRAND generator type

*Values:*

enumerator **HIPRAND_RNG_PSEUDO_DEFAULT**

Default pseudorandom generator.

enumerator **HIPRAND_RNG_PSEUDO_XORWOW**

XORWOW pseudorandom generator.

enumerator **HIPRAND_RNG_PSEUDO_MRG32K3A**

MRG32k3a pseudorandom generator.

enumerator **HIPRAND_RNG_PSEUDO_MTGP32**

Mersenne Twister MTGP32 pseudorandom generator.

enumerator **HIPRAND_RNG_PSEUDO_MT19937**

Mersenne Twister 19937.

enumerator **HIPRAND_RNG_PSEUDO_PHILOX4_32_10**

PHILOX_4x32 (10 rounds) pseudorandom generator.

enumerator **HIPRAND_RNG_QUASI_DEFAULT**

Default quasirandom generator.

enumerator **HIPRAND_RNG_QUASI_SOBOL32**

Sobol32 quasirandom generator.

enumerator **HIPRAND_RNG_QUASI_SCRAMBLED_SOBOL32**

Scrambled Sobol32 quasirandom generator.

enumerator **HIPRAND_RNG_QUASI_SOBOL64**

Sobol64 quasirandom generator.

enumerator **HIPRAND_RNG_QUASI_SCRAMBLED_SOBOL64**

Scrambled Sobol64 quasirandom generator.

enum **hiprandOrdering**

hipRAND generator ordering

*Values:*

enumerator **HIPRAND_ORDERING_PSEUDO_BEST**

Best ordering for pseudorandom results.

enumerator **HIPRAND_ORDERING_PSEUDO_DEFAULT**

Specific default thread sequence for pseudorandom results, same as HIPRAND_ORDERING_PSEUDO_BEST.

enumerator **HIPRAND_ORDERING_PSEUDO_SEEDED**

Specific seeding pattern for fast lower quality pseudorandom results.

enumerator **HIPRAND_ORDERING_PSEUDO_LEGACY**

Specific legacy sequence for pseudorandom results. This remains the same across releases, but not across backends.

enumerator **HIPRAND_ORDERING_PSEUDO_DYNAMIC**

Specific ordering adjusted to the device it is being executed on, provides the best performance.

enumerator **HIPRAND_ORDERING_QUASI_DEFAULT**

Specific n-dimensional ordering for quasirandom results.

enum **hiprandDirectionVectorSet**

hipRAND vector set for quasirandom generators.

*Values:*

enumerator **HIPRAND_DIRECTION_VECTORS_32_JOEKUO6**

enumerator **HIPRAND_SCRAMBLED_DIRECTION_VECTORS_32_JOEKUO6**

enumerator **HIPRAND_DIRECTION_VECTORS_64_JOEKUO6**

enumerator **HIPRAND_SCRAMBLED_DIRECTION_VECTORS_64_JOEKUO6**

### Functions

*hiprandStatus_t* **hiprandCreateGenerator**(hiprandGenerator_t *generator, *hiprandRngType_t* rng_type)

Creates a new random number generator.

Creates a new random number generator of type `rng_type`, and returns it in `generator`. That generator will use GPU to create random numbers.

Values for `rng_type` are:

- HIPRAND_RNG_PSEUDO_DEFAULT
- HIPRAND_RNG_PSEUDO_XORWOW
- HIPRAND_RNG_PSEUDO_MRG32K3A
- HIPRAND_RNG_PSEUDO_MTGP32
- HIPRAND_RNG_PSEUDO_MT19937
- HIPRAND_RNG_PSEUDO_PHILOX4_32_10
- HIPRAND_RNG_QUASI_DEFAULT
- HIPRAND_RNG_QUASI_SOBOL32
- HIPRAND_RNG_QUASI_SCRAMBLED_SOBOL32
- HIPRAND_RNG_QUASI_SOBOL64
- HIPRAND_RNG_QUASI_SCRAMBLED_SOBOL64

**Parameters**

- **generator** – - Pointer to generator
- **rng_type** – - Type of random number generator to create

**Returns**

- HIPRAND_STATUS_ALLOCATION_FAILED, if memory allocation failed
- HIPRAND_STATUS_INITIALIZATION_FAILED if there was a problem setting up the GPU
- HIPRAND_STATUS_VERSION_MISMATCH if the header file version does not match the dynamically linked library version
- HIPRAND_STATUS_TYPE_ERROR if the value for `rng_type` is invalid
- HIPRAND_STATUS_NOT_IMPLEMENTED if generator of type `rng_type` is not implemented yet
- HIPRAND_STATUS_SUCCESS if generator was created successfully

*hiprandStatus_t* **hiprandCreateGeneratorHost**(hiprandGenerator_t *generator, *hiprandRngType_t* rng_type)

Creates a new random number generator on host.

Creates a new host random number generator of type `rng_type` and returns it in `generator`. Created generator will use host CPU to generate random numbers.

Values for `rng_type` are:

- HIPRAND_RNG_PSEUDO_DEFAULT
- HIPRAND_RNG_PSEUDO_XORWOW

- HIPRAND_RNG_PSEUDO_MRG32K3A
- HIPRAND_RNG_PSEUDO_MTGP32
- HIPRAND_RNG_PSEUDO_MT19937
- HIPRAND_RNG_PSEUDO_PHILOX4_32_10
- HIPRAND_RNG_QUASI_DEFAULT
- HIPRAND_RNG_QUASI_SOBOL32
- HIPRAND_RNG_QUASI_SCRAMBLED_SOBOL32
- HIPRAND_RNG_QUASI_SOBOL64
- HIPRAND_RNG_QUASI_SCRAMBLED_SOBOL64

**Parameters**

- **generator** – - Pointer to generator
- **rng_type** – - Type of random number generator to create

**Returns**

- HIPRAND_STATUS_ALLOCATION_FAILED, if memory allocation failed
- HIPRAND_STATUS_VERSION_MISMATCH if the header file version does not match the dynamically linked library version
- HIPRAND_STATUS_TYPE_ERROR if the value for `rng_type` is invalid
- HIPRAND_STATUS_NOT_IMPLEMENTED if host generator of type `rng_type` is not implemented yet
- HIPRAND_STATUS_SUCCESS if generator was created successfully

*hiprandStatus_t* **hiprandDestroyGenerator**(hiprandGenerator_t generator)

Destroys random number generator.

Destroys random number generator and frees related memory.

**Parameters**

**generator** – - Generator to be destroyed

**Returns**

- HIPRAND_STATUS_NOT_INITIALIZED if the generator was not initialized
- HIPRAND_STATUS_SUCCESS if generator was destroyed successfully

*hiprandStatus_t* **hiprandGenerate**(hiprandGenerator_t generator, unsigned int *output_data, size_t n)

Generates uniformly distributed 32-bit unsigned integers.

Generates `n` uniformly distributed 32-bit unsigned integers and saves them to `output_data`.

Generated numbers are between `0` and `2^32`, including `0` and excluding `2^32`.

Note: `generator` must be not be of type `HIPRAND_RNG_QUASI_SOBOL64` or `HIPRAND_RNG_QUASI_SCRAMBLED_SOBOL64`.

**Parameters**

- **generator** – - Generator to use

- **output_data** – - Pointer to memory to store generated numbers
- **n** – - Number of 32-bit unsigned integers to generate

**Returns**

- HIPRAND_STATUS_NOT_INITIALIZED if the generator was not initialized
- HIPRAND_STATUS_LAUNCH_FAILURE if generator failed to launch kernel
- HIPRAND_STATUS_SUCCESS if random numbers were successfully generated

*hiprandStatus_t* **hiprandGenerateChar**(hiprandGenerator_t generator, unsigned char *output_data, size_t n)

Generates uniformly distributed 8-bit unsigned integers.

Generates `n` uniformly distributed 8-bit unsigned integers and saves them to `output_data`.

Generated numbers are between `0` and `2^8`, including `0` and excluding `2^8`.

**Parameters**

- **generator** – - Generator to use
- **output_data** – - Pointer to memory to store generated numbers
- **n** – - Number of 8-bit unsigned integers to generate

**Returns**

- HIPRAND_STATUS_NOT_INITIALIZED if the generator was not initialized
- HIPRAND_STATUS_LAUNCH_FAILURE if generator failed to launch kernel
- HIPRAND_STATUS_SUCCESS if random numbers were successfully generated

*hiprandStatus_t* **hiprandGenerateShort**(hiprandGenerator_t generator, unsigned short *output_data, size_t n)

Generates uniformly distributed 16-bit unsigned integers.

Generates `n` uniformly distributed 16-bit unsigned integers and saves them to `output_data`.

Generated numbers are between `0` and `2^16`, including `0` and excluding `2^16`.

**Parameters**

- **generator** – - Generator to use
- **output_data** – - Pointer to memory to store generated numbers
- **n** – - Number of 16-bit unsigned integers to generate

**Returns**

- HIPRAND_STATUS_NOT_INITIALIZED if the generator was not initialized
- HIPRAND_STATUS_LAUNCH_FAILURE if generator failed to launch kernel
- HIPRAND_STATUS_SUCCESS if random numbers were successfully generated

*hiprandStatus_t* **hiprandGenerateLongLong**(hiprandGenerator_t generator, unsigned long long *output_data, size_t n)

Generates uniformly distributed 64-bit unsigned integers.

Generates `n` uniformly distributed 64-bit unsigned integers and saves them to `output_data`.

Generated numbers are between `0` and `2^64`, including `0` and excluding `2^64`.

Note: `generator` must be of type `HIPRAND_RNG_QUASI_SOBOL64` or `HIPRAND_RNG_QUASI_SCRAMBLED_SOBOL64`.

**Parameters**

- **generator** – - Generator to use
- **output_data** – - Pointer to memory to store generated numbers
- **n** – - Number of 64-bit unsigned integers to generate

**Returns**

- HIPRAND_STATUS_NOT_INITIALIZED if the generator was not initialized
- HIPRAND_STATUS_LAUNCH_FAILURE if generator failed to launch kernel
- HIPRAND_STATUS_SUCCESS if random numbers were successfully generated

*hiprandStatus_t* **hiprandGenerateUniform**(hiprandGenerator_t generator, float *output_data, size_t n)

Generates uniformly distributed floats.

Generates `n` uniformly distributed 32-bit floating-point values and saves them to `output_data`.

Generated numbers are between `0.0f` and `1.0f`, excluding `0.0f` and including `1.0f`.

**Parameters**

- **generator** – - Generator to use
- **output_data** – - Pointer to memory to store generated numbers
- **n** – - Number of floats to generate

**Returns**

- HIPRAND_STATUS_NOT_INITIALIZED if the generator was not initialized
- HIPRAND_STATUS_LAUNCH_FAILURE if generator failed to launch kernel
- HIPRAND_STATUS_LENGTH_NOT_MULTIPLE if `n` is not a multiple of the dimension of used quasi-random generator
- HIPRAND_STATUS_SUCCESS if random numbers were successfully generated

*hiprandStatus_t* **hiprandGenerateUniformDouble**(hiprandGenerator_t generator, double *output_data, size_t n)

Generates uniformly distributed double-precision floating-point values.

Generates `n` uniformly distributed 64-bit double-precision floating-point values and saves them to `output_data`.

Generated numbers are between `0.0` and `1.0`, excluding `0.0` and including `1.0`.

Note: When `generator` is of type: `HIPRAND_RNG_PSEUDO_MRG32K3A`, `HIPRAND_RNG_PSEUDO_MTGP32`, `HIPRAND_RNG_QUASI_SOBOL32`, or `HIPRAND_RNG_QUASI_SCRAMBLED_SOBOL32` then the returned `double` values are generated from only 32 random bits each (one `unsigned int` value per one generated `double`).

**Parameters**

- **generator** – - Generator to use
- **output_data** – - Pointer to memory to store generated numbers
- **n** – - Number of floats to generate

**Returns**

- HIPRAND_STATUS_NOT_INITIALIZED if the generator was not initialized
- HIPRAND_STATUS_LAUNCH_FAILURE if generator failed to launch kernel
- HIPRAND_STATUS_LENGTH_NOT_MULTIPLE if `n` is not a multiple of the dimension of used quasi-random generator
- HIPRAND_STATUS_SUCCESS if random numbers were successfully generated

*hiprandStatus_t* **hiprandGenerateUniformHalf**(hiprandGenerator_t generator, half *output_data, size_t n)

Generates uniformly distributed half-precision floating-point values.

Generates `n` uniformly distributed 16-bit half-precision floating-point values and saves them to `output_data`.

Generated numbers are between `0.0` and `1.0`, excluding `0.0` and including `1.0`.

**Parameters**

- **generator** – - Generator to use
- **output_data** – - Pointer to memory to store generated numbers
- **n** – - Number of halfs to generate

**Returns**

- HIPRAND_STATUS_NOT_INITIALIZED if the generator was not initialized
- HIPRAND_STATUS_LAUNCH_FAILURE if generator failed to launch kernel
- HIPRAND_STATUS_LENGTH_NOT_MULTIPLE if `n` is not a multiple of the dimension of used quasi-random generator
- HIPRAND_STATUS_SUCCESS if random numbers were successfully generated

*hiprandStatus_t* **hiprandGenerateNormal**(hiprandGenerator_t generator, float *output_data, size_t n, float mean, float stddev)

Generates normally distributed floats.

Generates `n` normally distributed 32-bit floating-point values and saves them to `output_data`.

**Parameters**

- **generator** – - Generator to use
- **output_data** – - Pointer to memory to store generated numbers
- **n** – - Number of floats to generate
- **mean** – - Mean value of normal distribution
- **stddev** – - Standard deviation value of normal distribution

**Returns**

- HIPRAND_STATUS_NOT_INITIALIZED if the generator was not initialized
- HIPRAND_STATUS_LAUNCH_FAILURE if generator failed to launch kernel
- HIPRAND_STATUS_LENGTH_NOT_MULTIPLE if `n` is not even, `output_data` is not aligned to `sizeof(float2)` bytes, or `n` is not a multiple of the dimension of used quasi-random generator
- HIPRAND_STATUS_SUCCESS if random numbers were successfully generated

*hiprandStatus_t* **hiprandGenerateNormalDouble**(hiprandGenerator_t generator, double *output_data, size_t n, double mean, double stddev)

Generates normally distributed doubles.

Generates `n` normally distributed 64-bit double-precision floating-point numbers and saves them to `output_data`.

**Parameters**

- **generator** – - Generator to use
- **output_data** – - Pointer to memory to store generated numbers
- **n** – - Number of doubles to generate
- **mean** – - Mean value of normal distribution
- **stddev** – - Standard deviation value of normal distribution

**Returns**

- HIPRAND_STATUS_NOT_INITIALIZED if the generator was not initialized
- HIPRAND_STATUS_LAUNCH_FAILURE if generator failed to launch kernel
- HIPRAND_STATUS_LENGTH_NOT_MULTIPLE if `n` is not even, `output_data` is not aligned to `sizeof(double2)` bytes, or `n` is not a multiple of the dimension of used quasi-random generator
- HIPRAND_STATUS_SUCCESS if random numbers were successfully generated

*hiprandStatus_t* **hiprandGenerateNormalHalf**(hiprandGenerator_t generator, half *output_data, size_t n, half mean, half stddev)

Generates normally distributed halfs.

Generates `n` normally distributed 16-bit half-precision floating-point numbers and saves them to `output_data`.

**Parameters**

- **generator** – - Generator to use
- **output_data** – - Pointer to memory to store generated numbers
- **n** – - Number of halfs to generate
- **mean** – - Mean value of normal distribution
- **stddev** – - Standard deviation value of normal distribution

**Returns**

- HIPRAND_STATUS_NOT_INITIALIZED if the generator was not initialized
- HIPRAND_STATUS_LAUNCH_FAILURE if generator failed to launch kernel
- HIPRAND_STATUS_LENGTH_NOT_MULTIPLE if `n` is not even, `output_data` is not aligned to `sizeof(half2)` bytes, or `n` is not a multiple of the dimension of used quasi-random generator
- HIPRAND_STATUS_SUCCESS if random numbers were successfully generated

*hiprandStatus_t* **hiprandGenerateLogNormal**(hiprandGenerator_t generator, float *output_data, size_t n, float mean, float stddev)

Generates log-normally distributed floats.

Generates `n` log-normally distributed 32-bit floating-point values and saves them to `output_data`.

**Parameters**

- **generator** – - Generator to use
- **output_data** – - Pointer to memory to store generated numbers
- **n** – - Number of floats to generate
- **mean** – - Mean value of log normal distribution
- **stddev** – - Standard deviation value of log normal distribution

**Returns**

- HIPRAND_STATUS_NOT_INITIALIZED if the generator was not initialized
- HIPRAND_STATUS_LAUNCH_FAILURE if generator failed to launch kernel
- HIPRAND_STATUS_LENGTH_NOT_MULTIPLE if `n` is not even, `output_data` is not aligned to `sizeof(float2)` bytes, or `n` is not a multiple of the dimension of used quasi-random generator
- HIPRAND_STATUS_SUCCESS if random numbers were successfully generated

*hiprandStatus_t* **hiprandGenerateLogNormalDouble**(hiprandGenerator_t generator, double *output_data, size_t n, double mean, double stddev)

Generates log-normally distributed doubles.

Generates `n` log-normally distributed 64-bit double-precision floating-point values and saves them to `output_data`.

**Parameters**

- **generator** – - Generator to use
- **output_data** – - Pointer to memory to store generated numbers
- **n** – - Number of doubles to generate
- **mean** – - Mean value of log normal distribution
- **stddev** – - Standard deviation value of log normal distribution

**Returns**

- HIPRAND_STATUS_NOT_INITIALIZED if the generator was not initialized
- HIPRAND_STATUS_LAUNCH_FAILURE if generator failed to launch kernel
- HIPRAND_STATUS_LENGTH_NOT_MULTIPLE if `n` is not even, `output_data` is not aligned to `sizeof(double2)` bytes, or `n` is not a multiple of the dimension of used quasi-random generator
- HIPRAND_STATUS_SUCCESS if random numbers were successfully generated

*hiprandStatus_t* **hiprandGenerateLogNormalHalf**(hiprandGenerator_t generator, half *output_data, size_t n, half mean, half stddev)

Generates log-normally distributed halfs.

Generates `n` log-normally distributed 16-bit half-precision floating-point values and saves them to `output_data`.

**Parameters**

- **generator** – - Generator to use
- **output_data** – - Pointer to memory to store generated numbers

- **n** – - Number of halfs to generate
- **mean** – - Mean value of log normal distribution
- **stddev** – - Standard deviation value of log normal distribution

**Returns**

- HIPRAND_STATUS_NOT_INITIALIZED if the generator was not initialized
- HIPRAND_STATUS_LAUNCH_FAILURE if generator failed to launch kernel
- HIPRAND_STATUS_LENGTH_NOT_MULTIPLE if `n` is not even, `output_data` is not aligned to `sizeof(half2)` bytes, or `n` is not a multiple of the dimension of used quasi-random generator
- HIPRAND_STATUS_SUCCESS if random numbers were successfully generated

*hiprandStatus_t* **hiprandGeneratePoisson**(hiprandGenerator_t generator, unsigned int *output_data, size_t n, double lambda)

Generates Poisson-distributed 32-bit unsigned integers.

Generates `n` Poisson-distributed 32-bit unsigned integers and saves them to `output_data`.

**Parameters**

- **generator** – - Generator to use
- **output_data** – - Pointer to memory to store generated numbers
- **n** – - Number of 32-bit unsigned integers to generate
- **lambda** – - lambda for the Poisson distribution

**Returns**

- HIPRAND_STATUS_NOT_INITIALIZED if the generator was not initialized
- HIPRAND_STATUS_LAUNCH_FAILURE if generator failed to launch kernel
- HIPRAND_STATUS_OUT_OF_RANGE if lambda is non-positive
- HIPRAND_STATUS_LENGTH_NOT_MULTIPLE if `n` is not a multiple of the dimension of used quasi-random generator
- HIPRAND_STATUS_SUCCESS if random numbers were successfully generated

*hiprandStatus_t* **hiprandGenerateSeeds**(hiprandGenerator_t generator)

Initializes the generator's state on GPU or host.

Initializes the generator's state on GPU or host.

If *hiprandGenerateSeeds()* was not called for a generator, it will be automatically called by functions which generates random numbers like *hiprandGenerate()*, *hiprandGenerateUniform()*, *hiprandGenerateNormal()* etc.

**Parameters**

**generator** – - Generator to initialize

**Returns**

- HIPRAND_STATUS_NOT_INITIALIZED if the generator was never created
- HIPRAND_STATUS_PREEXISTING_FAILURE if there was an existing error from a previous kernel launch
- HIPRAND_STATUS_LAUNCH_FAILURE if the kernel launch failed for any reason

- HIPRAND_STATUS_SUCCESS if the seeds were generated successfully

*hiprandStatus_t* **hiprandSetStream**(hiprandGenerator_t generator, hipStream_t stream)

Sets the current stream for kernel launches.

Sets the current stream for all kernel launches of the generator. All functions will use this stream.

**Parameters**

- **generator** – - Generator to modify
- **stream** – - Stream to use or NULL for default stream

**Returns**

- HIPRAND_STATUS_NOT_INITIALIZED if the generator was not initialized
- HIPRAND_STATUS_SUCCESS if stream was set successfully

*hiprandStatus_t* **hiprandSetPseudoRandomGeneratorSeed**(hiprandGenerator_t generator, unsigned long long seed)

Sets the seed of a pseudo-random number generator.

Sets the seed of the pseudo-random number generator.

- This operation resets the generator's internal state.
- This operation does not change the generator's offset.

**Parameters**

- **generator** – - Pseudo-random number generator
- **seed** – - New seed value

**Returns**

- HIPRAND_STATUS_NOT_INITIALIZED if the generator was not initialized
- HIPRAND_STATUS_TYPE_ERROR if the generator is a quasi random number generator
- HIPRAND_STATUS_SUCCESS if seed was set successfully

*hiprandStatus_t* **hiprandSetGeneratorOffset**(hiprandGenerator_t generator, unsigned long long offset)

Sets the offset of a random number generator.

Sets the absolute offset of the random number generator.

- This operation resets the generator's internal state.
- This operation does not change the generator's seed.

Absolute offset cannot be set if generator's type is HIPRAND_RNG_PSEUDO_MTGP32 or HIPRAND_RNG_PSEUDO_MT19937.

**Parameters**

- **generator** – - Random number generator
- **offset** – - New absolute offset

**Returns**

- HIPRAND_STATUS_NOT_INITIALIZED if the generator was not initialized

- HIPRAND_STATUS_SUCCESS if offset was successfully set
- HIPRAND_STATUS_TYPE_ERROR if generator's type is HIPRAND_RNG_PSEUDO_MTGP32 or HIPRAND_RNG_PSEUDO_MT19937

*hiprandStatus_t* **hiprandSetGeneratorOrdering**(hiprandGenerator_t generator, *hiprandOrdering_t* order)

Sets the ordering of a random number generator.

Sets the ordering of the results of a random number generator.

- This operation resets the generator's internal state.
- This operation does not change the generator's seed.

The ordering choices for pseudorandom sequences are HIPRAND_ORDERING_PSEUDO_DEFAULT and HIPRAND_ORDERING_PSEUDO_LEGACY. The default ordering is HIPRAND_ORDERING_PSEUDO_DEFAULT, which is equal to HIPRAND_ORDERING_PSEUDO_LEGACY for now.

For quasirandom sequences there is only one ordering, HIPRAND_ORDERING_QUASI_DEFAULT.

**Parameters**

- **generator** – - Random number generator
- **order** – - New ordering of results

**Returns**

- HIPRAND_STATUS_NOT_INITIALIZED if the generator was not initialized
- HIPRAND_STATUS_OUT_OF_RANGE if the ordering is not valid
- HIPRAND_STATUS_SUCCESS if the ordering was successfully set
- HIPRAND_STATUS_TYPE_ERROR if generator's type is not valid

*hiprandStatus_t* **hiprandSetQuasiRandomGeneratorDimensions**(hiprandGenerator_t generator, unsigned int dimensions)

Set the number of dimensions of a quasi-random number generator.

Set the number of dimensions of a quasi-random number generator. Supported values of `dimensions` are 1 to 20000.

- This operation resets the generator's internal state.
- This operation does not change the generator's offset.

**Parameters**

- **generator** – - Quasi-random number generator
- **dimensions** – - Number of dimensions

**Returns**

- HIPRAND_STATUS_NOT_CREATED if the generator wasn't created
- HIPRAND_STATUS_TYPE_ERROR if the generator is not a quasi-random number generator

- HIPRAND_STATUS_OUT_OF_RANGE if `dimensions` is out of range
- HIPRAND_STATUS_SUCCESS if the number of dimensions was set successfully

*hiprandStatus_t* **hiprandGetVersion**(int *version)

Returns the version number of the cuRAND or rocRAND library.

Returns in `version` the version number of the underlying cuRAND or rocRAND library.

**Parameters**

**version** – - Version of the library

**Returns**

- HIPRAND_STATUS_OUT_OF_RANGE if `version` is NULL
- HIPRAND_STATUS_SUCCESS if the version number was successfully returned

*hiprandStatus_t* **hiprandCreatePoissonDistribution**(double lambda, hiprandDiscreteDistribution_t *discrete_distribution)

Construct the histogram for a Poisson distribution.

Construct the histogram for the Poisson distribution with lambda `lambda`.

**Parameters**

- **lambda** – - lambda for the Poisson distribution
- **discrete_distribution** – - pointer to the histogram in device memory

**Returns**

- HIPRAND_STATUS_ALLOCATION_FAILED if memory could not be allocated
- HIPRAND_STATUS_OUT_OF_RANGE if `discrete_distribution` pointer was null
- HIPRAND_STATUS_OUT_OF_RANGE if lambda is non-positive
- HIPRAND_STATUS_SUCCESS if the histogram was constructed successfully

*hiprandStatus_t* **hiprandDestroyDistribution**(hiprandDiscreteDistribution_t discrete_distribution)

Destroy the histogram array for a discrete distribution.

Destroy the histogram array for a discrete distribution created by hiprandCreatePoissonDistribution.

**Parameters**

**discrete_distribution** – - pointer to the histogram in device memory

**Returns**

- HIPRAND_STATUS_OUT_OF_RANGE if `discrete_distribution` was null
- HIPRAND_STATUS_SUCCESS if the histogram was destroyed successfully

*hiprandStatus_t* **hiprandGetDirectionVectors32**(*hiprandDirectionVectors32_t* **vectors, *hiprandDirectionVectorSet_t* set)

*hiprandStatus_t* **hiprandGetDirectionVectors64**(*hiprandDirectionVectors64_t* **vectors, *hiprandDirectionVectorSet_t* set)

*hiprandStatus_t* **hiprandGetScrambleConstants32**(const unsigned int **constants)

*hiprandStatus_t* **hiprandGetScrambleConstants64**(const unsigned long long **constants)

## 4.3 C++ host API wrapper

*group* **hipRAND host API C++ Wrapper**

### Typedefs

typedef philox4x32_10_engine **philox4x32_10**

Typedef of *hiprand_cpp::philox4x32_10_engine* PRNG engine with default seed (*HIPRAND_PHILOX4x32_DEFAULT_SEED*).

typedef xorwow_engine **xorwow**

Typedef of *hiprand_cpp::xorwow_engine* PRNG engine with default seed (*HIPRAND_XORWOW_DEFAULT_SEED*).

typedef mrg32k3a_engine **mrg32k3a**

Typedef of *hiprand_cpp::mrg32k3a_engine* PRNG engine with default seed (*HIPRAND_MRG32K3A_DEFAULT_SEED*).

typedef mtgp32_engine **mtgp32**

Typedef of *hiprand_cpp::mtgp32_engine* PRNG engine with default seed (*HIPRAND_MTGP32_DEFAULT_SEED*).

typedef mt19937_engine **mt19937**

Typedef of *hiprand_cpp::mt19937_engine* PRNG engine with default seed (*HIPRAND_MT19937_DEFAULT_SEED*).

typedef sobol32_engine **sobol32**

Typedef of *hiprand_cpp::sobol32_engine* QRNG engine with default number of dimensions (1).

typedef scrambled_sobol32_engine **scrambled_sobol32**

Typedef of *hiprand_cpp::scrambled_sobol32_engine* QRNG engine with default number of dimensions (1).

typedef sobol64_engine **sobol64**

Typedef of *hiprand_cpp::sobol64_engine* QRNG engine with default number of dimensions (1).

typedef scrambled_sobol64_engine **scrambled_sobol64**

Typedef of *hiprand_cpp::scrambled_sobol64_engine* QRNG engine with default number of dimensions (1).

typedef *xorwow* **default_random_engine**

Default random engine.

typedef std::random_device **random_device**

A non-deterministic uniform random number generator.

hiprand_cpp::random_device is non-deterministic uniform random number generator, or a pseudo-random number engine if there is no support for non-deterministic random number generation. It's implemented as a typedef of std::random_device.

For practical use hiprand_cpp::random_device is generally only used to seed a PRNG such as *hiprand_cpp::mtgp32_engine*.

Example:

```
#include <hiprand/hiprand.hpp>

int main()
{
    const size_t size = 8192;
    unsigned int * output;
    hipMalloc(&output, size * sizeof(unsigned int));

    hiprand_cpp::random_device rd;
    hiprand_cpp::mtgp32 engine(rd()); // seed engine with a real random value,␣
↪if available
    hiprand_cpp::normal_distribution<float> dist(0.0, 1.5);
    dist(engine, output, size);
}
```

### Functions

inline int **version**()

Returns hipRAND version.

**Returns**
hipRAND version number as an `int` value.

class **error** : public std::exception

*#include <hiprand.hpp>* A run-time hipRAND error.

The error class represents an error returned by a hipRAND function.

template<class **IntType** = unsigned int>

class **uniform_int_distribution**

*#include <hiprand.hpp>* Produces random integer values uniformly distributed on the interval [0, 2^(sizeof(IntType)*8) - 1].

**Template Parameters**
**IntType** – - type of generated values. Only `unsigned char`, `unsigned short`, `unsigned int`, `unsigned long long int` are supported.

template<class **RealType** = float>

class **uniform_real_distribution**

*#include <hiprand.hpp>* Produces random floating-point values uniformly distributed on the interval (0, 1].

**Template Parameters**
**RealType** – - type of generated values. Only `float`, `double` and `half` types are supported.

template<class **RealType** = float>

class **normal_distribution**

*#include <hiprand.hpp>* Produces random numbers according to a normal distribution.

**Template Parameters**
**RealType** – - type of generated values. Only `float`, `double` and `half` types are supported.

template<class **RealType** = float>

class **lognormal_distribution**

*#include <hiprand.hpp>* Produces positive random numbers according to a log-normal distribution.

**Template Parameters**
**RealType** – - type of generated values. Only `float`, `double` and `half` types are supported.

template<class **IntType** = unsigned int>

class **poisson_distribution**

*#include <hiprand.hpp>* Produces random non-negative integer values distributed according to Poisson distribution.

**Template Parameters**
**IntType** – - type of generated values. Only `unsinged int` type is supported.

template<unsigned long long **DefaultSeed** = HIPRAND_PHILOX4x32_DEFAULT_SEED>

class **philox4x32_10_engine**

*#include <hiprand.hpp>* Pseudorandom number engine based Philox algorithm.

*philox4x32_10_engine* implements a Counter-based random number generator called Philox, which was developed by a group at D. E. Shaw Research. It generates random numbers of type `unsigned int` on the interval [0; 2^32 - 1]. Random numbers are generated in sets of four.

template<unsigned long long **DefaultSeed** = HIPRAND_XORWOW_DEFAULT_SEED>

class **xorwow_engine**

*#include <hiprand.hpp>* Pseudorandom number engine based XORWOW algorithm.

*xorwow_engine* is a xorshift pseudorandom number engine based on XORWOW algorithm, which was presented by George Marsaglia in "Xorshift RNGs" paper published in Journal of Statistical Software. It produces random numbers of type `unsigned int` on the interval [0; 2^32 - 1].

template<unsigned long long **DefaultSeed** = HIPRAND_MRG32K3A_DEFAULT_SEED>

class **mrg32k3a_engine**

*#include <hiprand.hpp>* Pseudorandom number engine based MRG32k3a CMRG.

*mrg32k3a_engine* is an implementation of MRG32k3a pseudorandom number generator, which is a Combined Multiple Recursive Generator (CMRG) created by Pierre L'Ecuyer. It produces random 32-bit `unsigned int` values on the interval [0; 2^32 - 1].

template<unsigned long long **DefaultSeed** = HIPRAND_MTGP32_DEFAULT_SEED>

class **mtgp32_engine**

*#include <hiprand.hpp>* Pseudorandom number engine based on Mersenne Twister for Graphic Processors algorithm.

*mtgp32_engine* is a random number engine based on Mersenne Twister for Graphic Processors algorithm, which is a version of well-known Mersenne Twister algorithm. It produces high quality random numbers of type `unsigned int` on the interval [0; 2^32 - 1].

template<unsigned long long **DefaultSeed** = HIPRAND_MT19937_DEFAULT_SEED>

class **mt19937_engine**

*#include <hiprand.hpp>* Pseudorandom number engine based on Mersenne Twister.

*mt19937_engine* is a random number engine based on the well-known Mersenne Twister algorithm. It produces high quality random numbers of type `unsigned int` on the interval [0; 2^32 - 1].

template<unsigned int **DefaultNumDimensions** = 1>

class **sobol32_engine**

*#include <hiprand.hpp>* Sobol's quasi-random sequence generator.

*sobol32_engine* is quasi-random number engine which produced Sobol sequences. This implementation supports generating sequences in up to 20,000 dimensions. The engine produces random unsigned integers on the interval [0; 2^32 - 1].

template<unsigned int **DefaultNumDimensions** = 1>

class **scrambled_sobol32_engine**

*#include <hiprand.hpp>* Sobol's quasi-random sequence generator.

*scrambled_sobol32_engine* is a quasi-random number engine which produces scrambled Sobol sequences. This implementation supports generating sequences in up to 20,000 dimensions. The engine produces random unsigned integers on the interval [0; 2^32 - 1].

template<unsigned int **DefaultNumDimensions** = 1>

class **sobol64_engine**

*#include <hiprand.hpp>* Sobol's quasi-random sequence generator.

*sobol64_engine* is a quasi-random number engine which produced Sobol sequences. This implementation supports generating sequences in up to 20,000 dimensions. The engine produces random unsigned integers on the interval [0; 2^64 - 1].

template<unsigned int **DefaultNumDimensions** = 1>

class **scrambled_sobol64_engine**

*#include <hiprand.hpp>* Sobol's quasi-random sequence generator.

*scrambled_sobol64_engine* is a quasi-random number engine which produces scrambled Sobol sequences. This implementation supports generating sequences in up to 20,000 dimensions. The engine produces random unsigned integers on the interval [0; 2^64 - 1].

CHAPTER

FIVE

# PYTHON API REFERENCE

This document describes the hipRAND APIs in Python.

The APIs in this wrapper are similar to pyculib.rand.

## 5.1 class PRNG

**class** hiprand.**PRNG**(*rngtype=400*, *seed=None*, *offset=None*, *stream=None*, ***, *is_host=False*)

**DEFAULT = 400**

Default pseudo-random generator type, *XORWOW*

**MRG32K3A = 402**

MRG32k3a pseudo-random generator type

**MT19937 = 404**

Mersenne Twister 19937 pseudo-random generator type

**MTGP32 = 403**

Mersenne Twister MTGP32 pseudo-random generator type

**PHILOX4_32_10 = 405**

PHILOX_4x32 (10 rounds) pseudo-random generator type

**XORWOW = 401**

XORWOW pseudo-random generator type

**generate**(*ary*, *size=None*)

Generates uniformly distributed integers.

Generates **size** (if present) or **ary.size** uniformly distributed integers and saves them to **ary**.

**Supported dtype of ary for 32-bits generators:**
`numpy.uint32`, `numpy.int32`.

**Supported dtype of ary for 64-bits generators:**
`numpy.uint64`, `numpy.int64`.

**Parameters**

- **ary** – NumPy array (`numpy.ndarray`) or HIP device-side array (*DeviceNDArray*)
- **size** – Number of samples to generate, default to **ary.size**

`lognormal`(*ary*, *mean*, *stddev*, *size=None*)

Generates log-normally distributed floats.

Generates **size** (if present) or **ary.size** log-normally distributed floats and saves them to **ary**.

Supported **dtype** of **ary**: `numpy.float32`, `numpy.float64`.

**Parameters**

- **ary** – NumPy array (`numpy.ndarray`) or HIP device-side array (*DeviceNDArray*)
- **mean** – Mean value of log normal distribution
- **stddev** – Standard deviation value of log normal distribution
- **size** – Number of samples to generate, default to **ary.size**

`normal`(*ary*, *mean*, *stddev*, *size=None*)

Generates normally distributed floats.

Generates **size** (if present) or **ary.size** normally distributed floats and saves them to **ary**.

Supported **dtype** of **ary**: `numpy.float32`, `numpy.float64`.

**Parameters**

- **ary** – NumPy array (`numpy.ndarray`) or HIP device-side array (*DeviceNDArray*)
- **mean** – Mean value of normal distribution
- **stddev** – Standard deviation value of normal distribution
- **size** – Number of samples to generate, default to **ary.size**

`property offset`

Mutable attribute of the offset of random numbers sequence.

Setting this attribute resets the sequence.

`poisson`(*ary*, *lmbd*, *size=None*)

Generates Poisson-distributed integers.

Generates **size** (if present) or **ary.size** Poisson-distributed integers and saves them to **ary**.

Supported **dtype** of **ary**: `numpy.uint32`, `numpy.int32`.

**Parameters**

- **ary** – NumPy array (`numpy.ndarray`) or HIP device-side array (*DeviceNDArray*)
- **lmbd** – lambda for the Poisson distribution
- **size** – Number of samples to generate, default to **ary.size**

`property seed`

Mutable attribute of the seed of random numbers sequence.

Setting this attribute resets the sequence.

`uniform`(*ary*, *size=None*)

Generates uniformly distributed floats.

Generates **size** (if present) or **ary.size** uniformly distributed floats and saves them to **ary**.

Supported **dtype** of **ary**: `numpy.float32`, `numpy.float64`.

Generated numbers are between 0.0 and 1.0, excluding 0.0 and including 1.0.

**Parameters**

- **ary** – NumPy array (`numpy.ndarray`) or HIP device-side array (*DeviceNDArray*)
- **size** – Number of samples to generate, default to **ary.size**

## 5.2 class QRNG

**class** hiprand.**QRNG**(*rngtype=500*, *ndim=None*, *offset=None*, *stream=None*, ***, *is_host=False*)

**DEFAULT = 500**

Default quasi-random generator type, *SOBOL32*

**SCRAMBLED_SOBOL32 = 502**

Scrambled Sobol32 quasi-random generator type

**SCRAMBLED_SOBOL64 = 504**

Scrambled Sobol64 quasi-random generator type

**SOBOL32 = 501**

Sobol32 quasi-random generator type

**SOBOL64 = 503**

Sobol64 quasi-random generator type

**generate**(*ary*, *size=None*)

Generates uniformly distributed integers.

Generates **size** (if present) or **ary.size** uniformly distributed integers and saves them to **ary**.

**Supported dtype of ary for 32-bits generators:**
`numpy.uint32`, `numpy.int32`.

**Supported dtype of ary for 64-bits generators:**
`numpy.uint64`, `numpy.int64`.

**Parameters**

- **ary** – NumPy array (`numpy.ndarray`) or HIP device-side array (*DeviceNDArray*)
- **size** – Number of samples to generate, default to **ary.size**

**lognormal**(*ary*, *mean*, *stddev*, *size=None*)

Generates log-normally distributed floats.

Generates **size** (if present) or **ary.size** log-normally distributed floats and saves them to **ary**.

Supported **dtype** of **ary**: `numpy.float32`, `numpy.float64`.

**Parameters**

- **ary** – NumPy array (`numpy.ndarray`) or HIP device-side array (*DeviceNDArray*)
- **mean** – Mean value of log normal distribution
- **stddev** – Standard deviation value of log normal distribution
- **size** – Number of samples to generate, default to **ary.size**

**property ndim**

Mutable attribute of the number of dimensions of random numbers sequence.

Supported values are 1 to 20000. Setting this attribute resets the sequence.

**normal**(*ary*, *mean*, *stddev*, *size=None*)

Generates normally distributed floats.

Generates **size** (if present) or **ary.size** normally distributed floats and saves them to **ary**.

Supported **dtype** of **ary**: `numpy.float32`, `numpy.float64`.

**Parameters**

- **ary** – NumPy array (`numpy.ndarray`) or HIP device-side array (*DeviceNDArray*)
- **mean** – Mean value of normal distribution
- **stddev** – Standard deviation value of normal distribution
- **size** – Number of samples to generate, default to **ary.size**

**property offset**

Mutable attribute of the offset of random numbers sequence.

Setting this attribute resets the sequence.

**poisson**(*ary*, *lmbd*, *size=None*)

Generates Poisson-distributed integers.

Generates **size** (if present) or **ary.size** Poisson-distributed integers and saves them to **ary**.

Supported **dtype** of **ary**: `numpy.uint32`, `numpy.int32`.

**Parameters**

- **ary** – NumPy array (`numpy.ndarray`) or HIP device-side array (*DeviceNDArray*)
- **lmbd** – lambda for the Poisson distribution
- **size** – Number of samples to generate, default to **ary.size**

**uniform**(*ary*, *size=None*)

Generates uniformly distributed floats.

Generates **size** (if present) or **ary.size** uniformly distributed floats and saves them to **ary**.

Supported **dtype** of **ary**: `numpy.float32`, `numpy.float64`.

Generated numbers are between 0.0 and 1.0, excluding 0.0 and including 1.0.

**Parameters**

- **ary** – NumPy array (`numpy.ndarray`) or HIP device-side array (*DeviceNDArray*)
- **size** – Number of samples to generate, default to **ary.size**

## 5.3 Exceptions

**exception** hiprand.**HipRandError**(*value*)

Run-time hipRAND error.

**exception** hiprand.**HipError**(*value*)

Run-time HIP error.

## 5.4 Utilities

**class** hiprand.**DeviceNDArray**(*shape*, *dtype*, *data=None*)

Device-side array.

This class is a limited version of `numpy.ndarray` for device-side arrays.

See *empty()*

**copy_to_host**(*ary=None*)

Copy from data device memory to host memory.

If **ary** is passed then **ary** must have the same **dtype** and greater or equal **size** as **self** has.

If **ary** is not passed then a new `numpy.ndarray` will be created.

**Parameters**
**ary** – NumPy array (`numpy.ndarray`)

**Returns**
a new array or **ary**

hiprand.**empty**(*shape*, *dtype*)

Create a new device-side array of given shape and type, without initializing entries.

This function is a limited version of `numpy.empty()` for device-side arrays.

Example:

```
import hiprand
import numpy as np

gen = hiprand.QRNG(ndim=5)
d_a = hiprand.empty((5, 10000), dtype=np.float32)
gen.uniform(d_a)
a = d_a.copy_to_host()
print(a)
```

See *DeviceNDArray*

**Parameters**

- **shape** (*int or tuple of int*) – Shape of the array (see `numpy.ndarray.shape`)
- **dtype** – Type of the array (see `numpy.ndarray.dtype`)

hiprand.**get_version**()

Returns the version number of the rocRAND or cuRAND library.

CHAPTER

SIX

# LICENSE

# INDEX