Certification Report:	Metagalactic Club NV RNG Evaluation
Report Identification:	MLC-CR-230223-01-RC-R1
Certification Laboratory:	Gaming Associates Europe Ltd. www.gamingassociates.com •178 Merton High Street, London, UK, SW19 1AY. •Office 7, 82 London Road Leicester, UK, LE2 0QR •Hamngatan 27, Stockholm, Sweden
Supervisor:	Usman Vaseer
Signatures:	
Certifier:	Wajahat kashan
Dates of certification work:	01 February 2023 to 17 February 2023
Date of issue of certification report:	23 February 2023
Report prepared for:	Metagalactic Club NV Abraham de Veerstraat 9, Willemstad, P.O. Box 3421, Curaçao
Jurisdiction:	Curaçao eGaming
Technical Standard used for testing:	UK Remote Gambling and Software Technical Standards, February 2021

Gaming Associates

© GAMING ASSOCIATES

Certified RNG using Javascript for Gambling related Games

Installation

```
npm install xorshift
```

Example

```
var xorshift = require('xorshift');

for (var i = 0; i < 10; i++) {
  console.log(xorshift.random()); // number in range [0, 1)
}
```

Documentation

This module exports a default pseudo random generator. This generators seed have already been set (using `Date.now()`). If this is not suitable a custom generator can be initialized using the constructor function `xorshift.constructor`. In both cases random numbers can be generated using the two methods `.random` and `.randomint`.

```
var xorshift = require('xorshift');
```

xorshift.random()

This method returns a random 64-bit double, with its value in the range [0, 1). That means 0 is inclusive and 1 is exclusive. This is equivalent to `Math.random()`.

```
console.log(xorshift.random()); // number between 0 and 1
```

This method will serve most purposes, for instance to randomly select between 2, 3 and 4, this function can be used:

```
function uniformint(a, b) {
  return Math.floor(a + xorshift().random() * (b - a));
}

console.log(uniformint(2, 4));
```

xorshift.randomint()

This method returns a random 64-bit integer. Since JavaScript doesn't support 64-bit integers, the number is represented as an array with two elements in big-endian order.

This method is useful if high precision is required, the `xorshift.random()` method won't allow you to get this precision since a 64-bit IEEE754 double only contains the 52 most significant bits.

```
var bview = require('binary-view');
console.log(bview( new Uint32Array(xorshift.randomint()) ));
```

xorshift.constructor

This method is used to construct a new random generator, with a specific seed. This is useful when testing software where random numbers are involved and getting consistent results is important.

```
var XorShift = require('xorshift').constructor;
    var rng1 = new XorShift([1, 0, 2, 0]);
    var rng2 = new XorShift([1, 0, 2, 0]);

assert(rng1.random() === rng2.random());
```

A `XorShift` instance have both methods `random` and `randomint`. In fact the `xorshift` module is an instance of the `XorShift` constructor.

Reference

This module implements the xorshift128+ pseudo random number generator.

This is the fastest generator passing BigCrush without systematic errors, but due to the relatively short period it is acceptable only for applications with a very mild amount of parallelism; otherwise, use a xorshift1024* generator. – http://xorshift.di.unimi.it

This source also has a reference implementation for the xorshift128+ generator. A wrapper around this implementation has been created and is used for testing this module. To compile and run it:

```
gcc -O2 reference.c -o reference
./reference <numbers> <seed0> <seed1>
```

- `<numbers>` can be any number greater than zero, and it will be the number of random numbers written to `stdout`. The default value is `10`.
- `<seed0>` and `<seed1>` forms the 128bit seed that the algorithm uses. Default is `[1, 2]`.