
writing_otp_module Documentation

Release 0.1.1

Kenji Rikitake

Mar 18, 2017

CONTENTS:

1	Introduction	3
2	Deciding what to contribute	5
2.1	Erlang/OTP and the OTP Team need your help	5
2.2	What is considered as a contribution?	5
2.3	How <i>not</i> to write Erlang/OTP modules	6
2.4	Why writing Erlang/OTP modules then?	6
2.5	A case study: rand module	7
3	Prototyping the modules	9
3.1	Create an independent prototype repository	9
3.2	Building your own testing environment for Erlang/OTP	9
3.3	Setting up a project repository for your module	9
3.4	Execution path on loading modules	10
3.5	Write your own test code	10
4	Testing with Common Test	11
4.1	Read OTP source first to learn Common Test cases	11
4.2	Use groups for parallel and sequential tests	11
5	Adding type specifications with Dialyzer	13
5.1	Type specifications are required in OTP modules	13
5.2	Using Dialyzer	13
5.3	Dialyzer on improper lists	13
6	Writing documentation for Erlang/OTP Releases	15
6.1	Learning erl_docgen	15
6.2	Example of rand module documentation	15
7	Effectively issuing pull requests on GitHub	23
7.1	What is required in a pull request	23
7.2	How to work together on GitHub with the OTP Team	23
7.3	Rewriting a pull request	24
8	Communication with the OTP Team maintainers	25
8.1	Message channels with the OTP Team maintainers	25
8.2	Respect OTP Team working hours	25
9	Earning community support	27
9.1	User community support is essential	27
9.2	Things you can do for gaining community support	27

9.3	Release all you've got	27
9.4	Talk about your code	28
9.5	Write a PoC and show what the problem is	28
10	Maintenance after the initial release	29
10.1	Your responsibility on adding a module to Erlang/OTP	29
10.2	Move carefully for adding a new feature	29
11	Solving licensing issues of contributed code	31
11.1	License of Erlang/OTP: Apache License 2.0	31
11.2	Compatibility of Apache License 2.0 with other licenses	31
11.3	Relicensing your code to Erlang/OTP	32
12	Reference	33
13	Indices and tables	35

This document is about how to write and release an Erlang/OTP module as a part of Erlang/OTP.

INTRODUCTION

Erlang/OTP accepts many pieces of user-contributed code, but the contribution process is not necessarily trivial. In this document, I (the author) will describe possible issues for contributing a piece of code to Erlang/OTP, and how the issues can be effectively solved, based on my experience of writing [Erlang rand module](#).

This documentation is written for people who want to know

- how the practical tips about writing Erlang/OTP modules;
- how the OTP code is developed and released; and
- what to do for contributing their pieces of code to OTP.

This documentation is originally written for a presentation called [Writing A New Erlang/OTP Module for Beginners](#) to be held in March 2017, including the contents which are not necessarily suitable for the presentation.

DECIDING WHAT TO CONTRIBUTE

In this section, I will describe what a contributor should do and should not do for choosing what to contribute for Erlang/OTP.

2.1 Erlang/OTP and the OTP Team need your help

Before getting into the details, I would like to emphasize the fact that *the Erlang community and OTP Team always need your help* to fix the bugs and improve the software.

Erlang/OTP is a well-designed and highly stable product, as well as a language and a concurrent system. Most of the features are carefully designed and easy to use. Reading the [OTP online manual](#) will solve most of the problems you will encounter. (If you haven't, you should do it now.)

Erlang/OTP is actively maintained by the **OTP Team**. The OTP Team is *not* a group of volunteers; *they are paid developers* hired by Ericsson, and takes the final responsibility of the code quality. Erlang/OTP is not a hobby product; it is a mission-critical product for many parts of the telecommunication and internet infrastructure systems.

Unfortunately, even for such a great product, you might find *numerous* bugs. Sometimes those bugs are nasty enough to prevent writing further code, or may result in a serious vulnerability or a security hole. If you find any of the bugs, they *must be reported first to the OTP Team*, even if you cannot fix them. Erlang/OTP uses [the bug reporting site](#)³ solely for this purpose.

Also, OTP team has been chronically understaffed, so their bug fixing resources are limited. Lots of other people are helping the OTP team, such as [Erlang Solutions](#), [Industrial Erlang User Group](#), and the contributors of the [Erlang/OTP GitHub Repository](#). Many people from the [Elixir language](#) community also help a lot.

The development history of Erlang/OTP can be found here as a paper written by Joe Armstrong, the principal author of Erlang and one of the four leading developers of Erlang, as a [presentation called "A History of Erlang"](#) presented in [HOPS III conference held in 2007](#).

2.2 What is considered as a contribution?

Erlang/OTP has become an open source software (OSS) product since 1998. OSS is a community product, and every OSS has the own governance system. For Erlang/OTP, the OTP Team makes the final decision on what should be accepted or not as a contribution. It's more like a commercial product governance than other OSS products, based on the centralized approach of making decisions, while accepting broader contributions from the users through the GitHub repository and the Pull Request workflow.

Erlang/OTP has the own [contribution guideline page](#). It is mainly focused onto contributing code, but other fixes such as those for the documentation are also frequently accepted. Describing the *reason to change* is essential for proposing a change to Erlang/OTP. Here's a quote from the guideline:

³ You need to create your account first for [bugs.erlang.org](#). The bug tracking system is currently using Atlassian's JIRA.

It is important to write a good commit message explaining why the feature is needed. We [OTP Team] prefer that the information is in the commit message, so that anyone that want to know two years later why a particular feature can easily find out. It does no harm to provide the same information in the pull request (if the pull request consists of a single commit, the commit message will be added to the pull request automatically).

Note that a proposal which contains the Erlang language and semantic changes should be proposed as a part of [Erlang Enhancement Process \(EEP\)](#) as an *Erlang Extension Proposal* (also called EEP). You can find the archive of [Erlang Enhancement Process and the proposals in GitHub](#). In this document, I will not discuss the details on writing an EEP, because writing an EEP is for changing the syntax and semantics of Erlang and OTP, and it requires far deeper and broader knowledge than writing an OTP module.

2.3 How *not* to write Erlang/OTP modules

In Erlang, a group of functions is called *module*. Erlang/OTP is a set of modules¹. Writing a module is an essential part of the OTP software development process. And a set of modules is called *application*.

Making a decision of writing or not writing a module should be done carefully. Erlang/OTP provides various useful functions as their own modules. You will be astonished to find out the functions in the modules provided in the basic two applications, [kernel](#) and [stdlib](#). You can find many other functions and modules in OTP suitable for your work.

A general tip for choosing a function to use is to always consider using the existing functions in the OTP modules first *before* reinventing or even inventing your own ones. In most cases you can do what you want to do by choosing an existing function. In short: *read the OTP documentation first* before writing your own code; your problem can be solved by combining existing functions.

Development of Erlang software usually means writing the project-specific modules and make them the project-specific applications. The developed OTP applications can be combined with the OTP runtime environment as a *release*. In many cases having a *release* for your specific tasks will be sufficient for running your own software², even you have to include your own modules.

2.4 Why writing Erlang/OTP modules then?

In most cases you don't have to write the modules which should reside in OTP. For some cases, however, you need to write and contribute your code to the Erlang/OTP modules. You need to consider a few issues before thinking about making a contribution to OTP, as described in this section.

2.4.1 Is the module or new code really needed in OTP?

OTP is part of Erlang and cannot be separated from OTP⁴. If your module is accepted to OTP, that will affect *all* users of Erlang/OTP, including other Erlang-derived language users, such as those of Elixir and LFE. If your contribution is a critical bug fix, it should be merged as soon as possible. If it's a feature or an enhancement, it would take a much longer time to assess the impact of change.

2.4.2 Who needs the new code?

If what you want to do with your contribution can be done without changing other parts of Erlang/OTP, it should be separated from the OTP. Just because your code provides a cool feature does not necessarily justify that the code should be a part of Erlang/OTP.

¹ Erlang VM, or **BEAM**, has its own native functions called *built-in functions* (BIFs). BIFs are considered as a part of `erlang` module of OTP.

² You can even specify a version of Erlang/OTP when building a release, since a release is a complete set of executable program for an operating system.

⁴ OTP itself has been under restructuring and may be split into multiple parts of applications or modules which may be classified by OTP Team. The change, however, has not been happened yet as of March 2017.

2.4.3 Removing old code from OTP is hard

Removing an OTP module is extremely difficult after once it is accepted as a part of OTP. For example, the module `random`, which is being replaced by a new module `rand` which has become official since OTP 18.0, will remain for two major versions; it is now *deprecated* on OTP version 19, and will be *removed* on OTP version 20. The life of single OTP version is usually one year⁵, so *two years* will be required as minimum to replace an obsolete piece of code by the respective new code.

2.5 A case study: rand module

The module `rand`, a pseudo random number generator (PRNG) which I contributed with Dan Gudmundsson as the corresponding OTP maintainer for Erlang/OTP 18.0, had many specific reasons to be a part of OTP as follows:

- AS183, the algorithm of `random`, has been exploited less than a day with a modern computer⁶, so there was a strong need to provide the alternative to plug the security hole;
- Finer resolution of the output, which gives sufficient precision for Erlang float numbers;
- Much longer period for preventing prediction of *random* number values;
- Fully compatible or even simplified API for the programmer;
- Multiple choices of algorithms available for future extension and bugfix; and
- the execution speed for the default algorithm is as fast as `random` on a modern 64bit CPU.

There are also other situational factors on making `rand` module possible:

- [Xorshift*/+](#), a compact and well-tested PRNG implementation became available in the Public Domain;
- Awareness on improving algorithms of language-specific PRNGs increased (e.g, [JavaScript V8 Engine failure discovered on November 2015](#)); and
- I gained expertise of building PRNG module on Erlang/OTP by building prototypes for many different algorithms. On the other hand, it took *four* years to actually start implementing the `rand` module after [the first PRNG security incident of Erlang/OTP discovered by Geoff Cant on May 2011](#).

My lessons learned through the `rand` module development process are as follows:

- Software is a sticky being. You need to convince a lot of people to revise an old piece of software, even it has a critical vulnerability.
- Contribution of lots of people are needed to actualize revision of a piece of software. Having a good idea itself is *not enough*.
- You need to take time for the whole process unravels.

⁵ In the past R14, R15, and R16 releases of OTP, the length of the major version was two years. For the further details, check out the release dates of each OTP release.

⁶ See <https://github.com/jj1bdx/as183-c> for the proof-of-concept code to exploit all the possible values of AS183.

PROTOTYPING THE MODULES

3.1 Create an independent prototype repository

Erlang/OTP is a complex set of Erlang modules, largely dependent on each other. Building an independent prototype repository for each development project is much safer and more practical than tweaking the running system. Making a small independent source code repository for prototyping which can be integrated into other projects is a better choice than putting everything in a mixed-up run-time system.

3.2 Building your own testing environment for Erlang/OTP

Try-and-error debug on a running Erlang/OTP run-time system is not a good idea, especially when the system is being used in the production system. Building an independent BEAM on your own is often required, especially when you have to debug the BEAM or Erlang Run-Time System (ERTS) by yourself.

For maintaining and switching between multiple BEAMs, I use [kerl](#) because [kerl](#) provides a way to build your own Erlang/OTP environment from your own Git repository. The same principle is also applicable to other essential software packages, such as Elixir. For Elixir, I use [kiex](#) to use a separate environment from the system Elixir implementation.

3.3 Setting up a project repository for your module

In a modern software development procedure, you need to set up a building environment before writing code. For Erlang/OTP, there are a few building schemes to set up:

- [Rebar3](#) is an actively developed building and package managing tool for Erlang/OTP, which is chosen as a prospective package management tool for Erlang/OTP itself. Note that it is *not* backward compatible with the predecessor [Rebar 2](#), which is deprecated but still in use for many existing projects.
- [Erlang.mk](#) is a GNU Make based building tool, which also resolves dependencies for popular software packages. If you are an old-school make type of programmer like me, this tool is easy to use. If you need to embed non-Erlang code such as Native Interface Functions (NIFs) written in C or C++, [erlang.mk](#) will fit very well.
- [Mix](#) is the default building, dependency resolution, and package management tool for Elixir. The packaging system [Hex](#) is also well accepted in Erlang/OTP community, for relatively easy configuration and documentation support on [HexDocs](#). If you want to make your modules usable on Elixir (and you will surely want to do so), supporting mix is essential. You need to have some fluency on Elixir to use mix, because mix configuration language is based on Elixir¹.

¹ See [Introduction to Mix](#) as the primer for learning Mix.

3.4 Execution path on loading modules

For testing your own module, you need to guarantee the code of the module should be chosen and loaded *prior to* the system module of the same name².

When running BEAM, the loading path priority can be controlled by `-pa` and `-pz` path of the `erl` command. Some package tools have their own commands for setting the execution path, such as [Rebar3 shell command](#). On Elixir, the interactive shell `ix` has the similar functionality.

3.5 Write your own test code

Testing code and providing a regression test set is a critical part of modern software development. Unit testing is essential as a minimum requirement. [EUnit](#) is a good place to start. For Erlang/OTP module development, however, providing tests under the [Common Test](#) framework is mandatory.

Testing code should cover various aspects including checking the function arguments, value range, and error handling to reject illegal values. For a complex algorithm, you need to provide the reference output from a reference implementation which is not necessarily written in Erlang, and compare the output with the corresponding testing program.

For prototyping the *rand* module algorithms into Erlang code, I picked up the code from the reference implementations of each candidate algorithm in C, wrote a small testing value generation code, collected the computed results, and embedded them into the unit testing code of the Erlang module. Having this kind of full regression tests help finding a bug possibly caused by the version change or implementation glitches caused by the base Erlang/OTP distribution. Making the module for each algorithm an independent repository helped ensuring the correctness of the algorithm without worrying about interdependencies.

² Erlang module namespace is *not* hierarchical, so the module name collision may occur at any time.

TESTING WITH COMMON TEST

4.1 Read OTP source first to learn Common Test cases

All OTP module unit tests are performed under [The Common Test Framework](#). You need to learn the Common Test for obtaining quality assurance of including your code into OTP.

I think the easiest way to start writing Common Test cases is to read the ones already written in the OTP source. Each OTP module has the Common Test case source code under each application with the name of `test/*_SUITE.erl`¹. You can find a lot of useful patterns.

Common Test manual chapter for [Writing Test Suites](#) provides a lot of useful examples, including the [Test Case Groups](#).

4.2 Use groups for parallel and sequential tests

Common Test has the Test Case Groups, which allows programmers to enforce the sequence of executing the tests and to specify parallel execution of a set of tests.

The following code in `rand` module² explains the sequence of the tests to be performed, beginning from `seed`, with a group called `basic_stats` and the last group of `reference_jump`.

```
all() ->
[seed, interval_int, interval_float,
 api_eq,
 reference,
 {group, basic_stats},
 plugin, measure,
 {group, reference_jump}
].
```

You need to list all the test functions and groups in `all/0`.

The group properties are defined in `groups/0` as follows, setting *parallel execution* flag for each of the two groups with the test functions in the group:

```
groups() ->
[{{basic_stats, [parallel]},
  [basic_stats_uniform_1, basic_stats_uniform_2, basic_stats_normal]},
 {reference_jump, [parallel]},
 {reference_jump_state, reference_jump_procdict}}].
```

¹ For example, the Common Test cases for `rand` module is at `lib/stdlib/test/rand_SUITE.erl` in the OTP source code tree, since `rand` is a part of `stdlib` application.

² The code will be appeared on OTP 20. See <https://github.com/erlang/otp/commit/1c000086275c06596ad402081be12ef95db6ea40> for the details.

ADDING TYPE SPECIFICATIONS WITH DIALYZER

5.1 Type specifications are required in OTP modules

All OTP exported functions (i.e., those defined with `-export()` attribute) have [the type specifications](#). The specifications are essential to define the type constraints for enabling static analysis of the Erlang code. Some module also have their own types, and they are essential to get access to the data structures defined in the module.

Having a correct type specification for each of the exported/global modules and related types is mandatory to be accepted as a part of Erlang/OTP. *Without the necessary type specifications, your code will not be accepted by the OTP Team¹.*

5.2 Using Dialyzer

[Dialyzer](#) is the tool of Erlang for performing the static analysis with the source code and the BEAM binary code with the debugging option enabled. You can invoke dialyzer as:

```
dialyzer --build_plt --apps erts kernel stdlib
```

Note well that the build tools may have Dialyzer options as well, such as `rebar3 dialyzer` for `rebar3` or `make dialyze` of `erlang.mk`.

5.3 Dialyzer on improper lists

Erlang/OTP `rand` module has the `Xorshift116+` code, which extensively uses an improper list for an internal data structure. Dialyzer is picky on detecting the improper list, so the following attribute for the warning option has to be configured to prevent unnecessary warnings:

```
-dialyzer({no_improper_lists, exsplus_next/1}).
```

See the section of [Requesting or Suppressing Warnings in Source Files in the Dialyzer Reference Manual](#) for the further details.

¹ You should also be careful that providing the type specification for non-exported (private) functions or types are not necessary.

WRITING DOCUMENTATION FOR ERLANG/OTP RELEASES

6.1 Learning `erl_docgen`

Writing documentation for Erlang/OTP requires adherence to the standard and format of the [Erlang/OTP manual](#). To prepare the document, you need to use the tool called `erl_docgen`.

`erl_docgen` uses the own XML tags and notation for writing the manual. The XML definition for writing a documentation of a module or an application is listed as the [Reference Manual DTDs](#). I am not going into the details of the XML tag definitions, because they are given in the `erl_docgen` documentation, and rather can be easily understood by reading actual XML documentation.

6.2 Example of `rand` module documentation

For prototyping `rand` module, I set up a locally executable documentation build environment. See the [xml-doc directory of the prototype repository](#) for the further details. You can also try [doing full building of the documentation](#).

The following list is an example source XML file for Erlang/OTP `rand` module, as provided in the [prototype repository](#). The rendered result can be accessed as [an html file](#).

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE erlref SYSTEM "erlref.dtd">

<erlref>
  <header>
    <copyright>
      <year>2015</year>
      <holder>Ericsson AB. All Rights Reserved.</holder>
    </copyright>
    <legalnotice>
      Licensed under the Apache License, Version 2.0 (the "License");
      you may not use this file except in compliance with the License.
      You may obtain a copy of the License at

        http://www.apache.org/licenses/LICENSE-2.0

      Unless required by applicable law or agreed to in writing, software
      distributed under the License is distributed on an "AS IS" BASIS,
      WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
      See the License for the specific language governing permissions and
      limitations under the License.
```

(continues on next page)

(continued from previous page)

```

</legalnotice>

<title>rand</title>
<prepared></prepared>
<responsible></responsible>
<docno>1</docno>
<approved></approved>
<checked></checked>
<date></date>
<rev>A</rev>
<file>rand.xml</file>
</header>
<module>rand</module>
<modulesummary>Pseudo random number generation.</modulesummary>
<description>
  <p>This module provides a random number generator. The module contains
    a number of algorithms. The uniform distribution algorithms use the
    <a href="http://xorshift.di.unimi.it">scrambled Xorshift algorithms by
    Sebastiano Vigna</a>. The normal distribution algorithm uses the
    <a href="http://www.jstatsoft.org/v05/i08">Ziggurat Method by Marsaglia
    and Tsang</a>.</p>
  <p>For some algorithms, jump functions are provided for generating
    non-overlapping sequences for parallel computations.
    The jump functions perform calculations
    equivalent to perform a large number of repeated calls
    for calculating new states. </p>

  <p>The following algorithms are provided:</p>

  <taglist>
    <tag><c>exsplus</c></tag>
    <item>
      <p>Xorshift116+, 58 bits precision and period of  $2^{116}-1$ </p>
      <p>Jump function: equivalent to  $2^{64}$  calls</p>
    </item>
    <tag><c>exs64</c></tag>
    <item>
      <p>Xorshift64*, 64 bits precision and a period of  $2^{64}-1$ </p>
      <p>Jump function: not available</p>
    </item>
    <tag><c>exs1024</c></tag>
    <item>
      <p>Xorshift1024*, 64 bits precision and a period of  $2^{1024}-1$ </p>
      <p>Jump function: equivalent to  $2^{512}$  calls</p>
    </item>
  </taglist>

  <p>The default algorithm is <c>exsplus</c>. If a specific algorithm is
    required, ensure to always use <seealso marker="#seed-1">
    <c>seed/1</c></seealso> to initialize the state.</p>

  <p>Every time a random number is requested, a state is used to

```

(continues on next page)

(continued from previous page)

calculate it and a new state is produced. The state can either be implicit or be an explicit argument and return value.

The functions with implicit state use the process dictionary variable `rand_seed` to remember the current state.

If a process calls `rand.uniform/0` or `rand.uniform/1` without setting a seed first, `rand.seed/1` is called automatically with the default algorithm and creates a non-constant seed.

The functions with explicit state never use the process dictionary.

Examples:

Simple use; creates and seeds the default algorithm with a non-constant seed if not already done:

```
<pre>
R0 = rand:uniform(),
R1 = rand:uniform(),</pre>
```

Use a specified algorithm:

```
<pre>
_ = rand:seed(exs1024),
R2 = rand:uniform(),</pre>
```

Use a specified algorithm with a constant seed:

```
<pre>
_ = rand:seed(exs1024, {123, 123534, 345345}),
R3 = rand:uniform(),</pre>
```

Use the functional API with a non-constant seed:

```
<pre>
S0 = rand:seed_s(exsplus),
{R4, S1} = rand:uniform_s(S0),</pre>
```

Create a standard normal deviate:

```
<pre>
{SND0, S2} = rand:normal_s(S1),</pre>
```

note

This random number generator is not cryptographically strong. If a strong cryptographic random number generator is needed, use one of functions in the `crypto:crypto` module.

(continues on next page)

(continued from previous page)

```

    module, for example, <seealso marker="crypto:crypto">
    <c>crypto:strong_rand_bytes/1</c></seealso>.</p>
</note>

</description>
<datatypes>
  <datatype>
    <name name="alg"/>
  </datatype>
  <datatype>
    <name name="state"/>
    <desc><p>Algorithm-dependent state.</p></desc>
  </datatype>
  <datatype>
    <name name="export_state"/>
    <desc><p>Algorithm-dependent state that can be printed or saved to
    file.</p></desc>
  </datatype>
</datatypes>

<funcs>
  <func>
    <name name="export_seed" arity="0"/>
    <fsummary>Export the random number generation state.</fsummary>
    <desc><marker id="export_seed-0"/>
    <p>Returns the random number state in an external format.
    To be used with <seealso marker="#seed-1"><c>seed/1</c></seealso>.</p>
    </desc>
  </func>

  <func>
    <name name="export_seed_s" arity="1"/>
    <fsummary>Export the random number generation state.</fsummary>
    <desc><marker id="export_seed_s-1"/>
    <p>Returns the random number generator state in an external format.
    To be used with <seealso marker="#seed-1"><c>seed/1</c></seealso>.</p>
    </desc>
  </func>

  <func>
    <name name="jump" arity="0"/>
    <fsummary>Return the seed after performing jump calculation
    to the state in the process dictionary.</fsummary>
    <desc><marker id="jump-0" />
    <p>Returns the state
    after performing jump calculation
    to the state in the process dictionary.</p>
    <p>This function generates a <c>not_implemented</c> error exception
    when the jump function is not implemented for
    the algorithm specified in the state
    in the process dictionary.</p>
    </desc>

```

(continues on next page)

(continued from previous page)

```

</func>

<func>
  <name name="jump" arity="1"/>
  <fsummary>Return the seed after performing jump calculation.</fsummary>
  <desc><marker id="jump-1" />
    <p>Returns the state after performing jump calculation
      to the given state. </p>
    <p>This function generates a <c>not_implemented</c> error exception
      when the jump function is not implemented for
      the algorithm specified in the state.</p>
  </desc>
</func>

<func>
  <name name="normal" arity="0"/>
  <fsummary>Return a standard normal distributed random float.</fsummary>
  <desc>
    <p>Returns a standard normal deviate float (that is, the mean
      is 0 and the standard deviation is 1) and updates the state in
      the process dictionary.</p>
  </desc>
</func>

<func>
  <name name="normal_s" arity="1"/>
  <fsummary>Return a standard normal distributed random float.</fsummary>
  <desc>
    <p>Returns, for a specified state, a standard normal
      deviate float (that is, the mean is 0 and the standard
      deviation is 1) and a new state.</p>
  </desc>
</func>

<func>
  <name name="seed" arity="1"/>
  <fsummary>Seed random number generator.</fsummary>
  <desc>
    <marker id="seed-1"/>
    <p>Seeds random number generation with the specified algorithm and
      time-dependent data if <anno>AlgOrExpState</anno> is an algorithm.</p>
    <p>Otherwise recreates the exported seed in the process dictionary,
      and returns the state. See also
      <seealso marker="#export_seed-0"><c>export_seed/0</c></seealso>.</p>
  </desc>
</func>

<func>
  <name name="seed" arity="2"/>
  <fsummary>Seed the random number generation.</fsummary>
  <desc>
    <p>Seeds random number generation with the specified algorithm and

```

(continues on next page)

(continued from previous page)

```

    integers in the process dictionary and returns the state.</p>
</desc>
</func>

<func>
  <name name="seed_s" arity="1"/>
  <fsummary>Seed random number generator.</fsummary>
  <desc>
    <p>Seeds random number generation with the specified algorithm and
      time-dependent data if <anno>AlgOrExpState</anno> is an algorithm.</p>
    <p>Otherwise recreates the exported seed and returns the state.
      See also <seealso marker="#export_seed-0">
      <c>export_seed/0</c></seealso>.</p>
  </desc>
</func>

<func>
  <name name="seed_s" arity="2"/>
  <fsummary>Seed the random number generation.</fsummary>
  <desc>
    <p>Seeds random number generation with the specified algorithm and
      integers and returns the state.</p>
  </desc>
</func>

<func>
  <name name="uniform" arity="0"/>
  <fsummary>Return a random float.</fsummary>
  <desc><marker id="uniform-0"/>
    <p>Returns a random float uniformly distributed in the value
      range <c>0.0 &lt; <anno>X</anno> &lt; 1.0</c> and
      updates the state in the process dictionary.</p>
  </desc>
</func>

<func>
  <name name="uniform" arity="1"/>
  <fsummary>Return a random integer.</fsummary>
  <desc><marker id="uniform-1"/>
    <p>Returns, for a specified integer <c><anno>N</anno> >= 1</c>,
      a random integer uniformly distributed in the value range
      <c>1 &lt;= <anno>X</anno> &lt;= <anno>N</anno></c> and
      updates the state in the process dictionary.</p>
  </desc>
</func>

<func>
  <name name="uniform_s" arity="1"/>
  <fsummary>Return a random float.</fsummary>
  <desc>
    <p>Returns, for a specified state, random float
      uniformly distributed in the value range <c>0.0 &lt;

```

(continues on next page)

(continued from previous page)

```
<anno>X</anno> &lt; 1.0</c> and a new state.</p>
</desc>
</func>

<func>
  <name name="uniform_s" arity="2"/>
  <fsummary>Return a random integer.</fsummary>
  <desc>
    <p>Returns, for a specified integer <c><anno>N</anno> >= 1</c>
    and a state, a random integer uniformly distributed in the value
    range <c>1 &lt;= <anno>X</anno> &lt;= <anno>N</anno></c> and a
    new state.</p>
  </desc>
</func>
</funcs>
</erlref>
```


EFFECTIVELY ISSUING PULL REQUESTS ON GITHUB

Opening a pull request (PR) to [Erlang/OTP GitHub repository](#) is the default contribution procedure described in the [contribution guideline of Erlang/OTP](#).

7.1 What is required in a pull request

The commit message of the PR should contain the following elements:

- What you are going to do with the PR;
- How the PR changes the behavior of the code;
- What and how the changes induced by the PR will affect to the other modules and applications; and
- What and how the changes induced by the PR will *not* affect to the other modules and applications.

The changed code and documentation should meet the following standards:

- For a new feature, the new PR should be committed to the `master` branch.
- If the contribution is a bugfix, reporting the bug is mandatory.
- The code should be built and tested on all major platforms, such as Linux, Windows, macOS, and FreeBSD, unless implementing a platform-specific feature.
- The documentation should be updated as well.
- Backward compatibility is a requirement. Even if a non-compatible feature is accepted, the old code will remain for one or two major OTP releases.
- The coding style must meet the standard for OTP. In short: 4 spaces, use Emacs erlang-mode.
- Make separate commits for separate changes.
- Make sure that each commit can be compiled and that the commit works.

7.2 How to work together on GitHub with the OTP Team

OTP Team usually takes a conservative approach to new features and code. The team member may frequently and repeatedly review your PRs and will suggest you to change the PRs for improvement. You are supposed to follow their requests, unless you have a legitimate reason to oppose; you need to find an agreement with the OTP Team, rather than to make a confrontation, because time and other resources for the OTP Team is limited. In short: OTP Team members have *no time to waste for confrontational arguments*.

7.3 Rewriting a pull request

Rewriting a pull request often results in a large number of commits, where the purpose and meaning of each commit is hard to understand later. OTP Team often request you to squash the commits to simplify the committed code for easier understanding. Being able to handle `git rebase` command is required for simplifying and streamlining your commits. If you need to commit against the `master` branch, you may frequently need to perform `git rebase -i master`.

COMMUNICATION WITH THE OTP TEAM MAINTAINERS

8.1 Message channels with the OTP Team maintainers

OTP Team assigns a pull request (PR) to a team member as a maintainer. Close and frequent communication with the maintainer is a key for contributing good and high quality code. Many levels of communication may happen, as in the following list:

- GitHub issues (on the PR), for open and public exchange of ideas, soliciting broader participation from the community;
- Direct email exchange, mainly for the unofficial exchange of conceptual ideas and making private communication and decision; and
- Other informal channels such as face-to-face meeting in a conference, which is rare but important to find out the gut feelings of each other on various issues.

8.2 Respect OTP Team working hours

Most of OTP Team maintainers seem to work in the Sweden time zone, which is the same as [Central European Time Zone](#) as observed in many other nations and regions in Europe.

In the Central European Time Zone, CET, the standard time in the Winter is UTC + 1 hour. Central European Summer Time (CEST), which is observed during the summer, is UTC + 2 hours. Note well that the Summer Time or Daylight Saving Time period *differ between Europe and US/Canada*.

When you send and receive message to OTP Team, you need to respect their regular working hours. For many of the volunteer contributors including myself, writing Erlang/OTP code is often *for fun*, but for the OTP Team it is their *duty and business*. They rarely work in weekends and holidays¹. Sending messages will be OK even on their vacation, but don't expect a prompt reply during non-working hours. And remember OTP Team members have to handle a lot of different OTP modules and applications, so their time slot for each application is limited.

¹ Everyone needs rest to recuperate.

EARNING COMMUNITY SUPPORT

9.1 User community support is essential

An excellent proposal for an open source software product without the user community support will not be approved or chosen by the development team. User community support is essential and critical to get your proposal or changes accepted into the repository.

9.2 Things you can do for gaining community support

It took almost six years to deprecate and remove `random` module and let the Erlang/OTP users convert into `rand` module, from 2011 to 2017. During the process, I have attempted a few kinds of promoting the need to change for a new pseudo random number generator (PRNG).

In the following sections, I will explain the activities I've done to promote the problem, issues, and the solution regarding the `random` module.

9.3 Release all you've got

I've released all PRNG algorithm code which are the candidates for `rand` module. The repositories for the Erlang implementations are:

- `sfmt-erlang`, an implementation of SIMD-oriented Fast Mersenne Twister (SFMT);
- `tinymt-erlang`, an implementation of Tiny Mersenne Twister (TinyMT);
- `exsplus128`, an Xorshift128+ implementation;
- `exs64`, an Xorshift64* implementation;
- `exs1024`, an Xorshift1024* implementation; and
- `exsplus116`, an Xorshift116+ implementation.

For the `rand` module, OTP Team finally chose `exs64`, `exs1024`, and `exsplus116` (called as `exsplus` in the `rand` module, the default algorithm).

OTP Team developer Dan Gudmundsson, who is in charge of `rand` module, even proposed the author of Xorshift*/+ algorithm Sebastiano Vigna to invent a new algorithm *specifically for Erlang/OTP's limitation of 60-bit integers*, which is Xorshift116+, the algorithm for `exsplus`. Dan was so helpful for actualizing the module for Erlang/OTP.

Releasing the code has many advantages including the following:

- Being used by many people will increase the chance to obtain the feedbacks for improvement;
- Even if the module won't be merged as a part of Erlang/OTP, many people still want to use as a part of their application software, so releasing them itself will be of a great service for those people; and

- Making the code public available will increase the awareness of the issues, which is about PRNG quality in my case.

9.4 Talk about your code

Having a talk on your issues at a public conference or a workshop, including the problem analysis and the possible solutions, will help raising the awareness of the issues.

In my case, I gave PRNG-related presentations at influential conferences such as:

- Erlang Factory SF Bay Area 2011
- ACM Erlang Workshop 2011
- ACM Erlang Workshop 2012
- London Erlang User Group Meetup in September 2013
- Erlang Factory SF Bay Area 2015
- Erlang User Conference 2016

A talk proposal for the module you want to commit for Erlang/OTP will be a viable candidate to be presented in a Erlang-and-Elixir-related conference. In such a conference, you can talk to an influential person, or to a complete stranger, to personally obtain a direct feedback to your idea. Such an occasion is precious and important, especially when the person shows support to your idea.

9.5 Write a PoC and show what the problem is

I think the most influential activity for letting OTP Team make a decision to change the default PRNG algorithm is the fact that the old algorithm in `random` module, AS183, was practically easy to exploit in less than eight hours with the proof-of-concept (PoC) code in my `as183-c`, which I released in December 2014.

While writing and publishing a PoC code *do not* imply that the idea represented in the code is new or that nobody proved the practicality before the PoC code was published¹, a PoC will surely raise an awareness of the possible attack vectors against known vulnerabilities.

¹ An Erlang/OTP veteran (whose name is withheld for respecting his privacy) sent me a personal email that my PoC code had no new discovery either in research or engineering aspects and *absolutely meaningless*. A good news was, however, that he also praised me for publishing alternative implementations for public use.

MAINTENANCE AFTER THE INITIAL RELEASE

10.1 Your responsibility on adding a module to Erlang/OTP

When you add a module to Erlang/OTP, that means you are *also responsible* for maintaining the module, especially when the security issues or vulnerabilities are found. As in the case of typical Open Source Software, you are not legally bound to provide a help, but you are *morally and professionally bound* to provide a help when it is requested by the OTP Team. Also, you are expected to contribute a further enhancement or a new feature when they are available.

An unmaintained piece of code will be deprecated and removed from OTP by the OTP Team's discretion. Old code is prone to vulnerability or degradation leading to new security issues. For example, `random` module, the predecessor of `rand` module, survived for a very long time at least from February 1999¹, but finally to be phased out and removed from OTP 20, which will be released in June 2017². Generally speaking, an 18-year old code should be reviewed and revised simply because it is *old enough*.

10.2 Move carefully for adding a new feature

Adding a new feature to an existing Erlang/OTP module may require the same or even more effort to write a new module. OTP Team generally takes conservative approach to changes, and the users of Erlang/OTP do not want drastic changes, especially when they break the backward compatibility. So you need to be careful on moving for adding a new feature to an existing module, even if that is under your control.

¹ The oldest available source code archive of Open Source Erlang at http://erlang.org/download/OLD/erlang_base-47.4.1.src.tar.gz suggests the `random` module code resided at `erlang-47.4.1/lib/stdlib/src/random.erl`. The time stamp of the file was on Feb 24 1999.

² Quoted from Kenneth Lundin's presentation slides of News from The OTP Team, at Erlang User Conference 2016 in Stockholm, Sweden, September 2016.

SOLVING LICENSING ISSUES OF CONTRIBUTED CODE

Note: the author is not a lawyer and the contents in this section may not be legally complete. Ask lawyers for the legal advices.

11.1 License of Erlang/OTP: Apache License 2.0

Erlang/OTP has been licensed under [Apache License 2.0](#) since the release of Erlang/OTP 18.0¹ on May 2015.

Rami Sass explains the summary of how Apache License 2.0 works in the blog entry [Top 10 Apache License Questions Answered](#). Some of the points are quoted here as follows:

- Apache License 2.0 is a permissive open source license.
- You can freely use, modify, distribute and sell a software licensed under the Apache License without worrying about the use of software: *personal, internal or commercial*.
- If you redistribute software with any Apache licensed components, you must include a copy of the license, provide a clear Apache License attribution, and add modification notices to all the files that you modify.
- You can choose to release the modified or derived products under different licenses. The unmodified parts of the software, however, must retain the Apache License.
- You cannot name your modified version in any way that suggests that the final product is endorsed or created by the Apache Software Foundation.
- Every licensed file must contain any original copyright, patent, trademark, and attribution notices in its redistributed code.
- Each modified file must also contain a notice about all the changes made to the original file.

11.2 Compatibility of Apache License 2.0 with other licenses

Mixing source code which has other licenses than Apache License 2.0 should be done carefully.

11.2.1 GPL version 2 and 3

Software under GNU Public License (GPL) cannot be directly linked to Erlang/OTP, due to the license compatibility with Apache License 2.0.

According to the [document of Apache Software Foundation about Apache License 2.0 and GPL compatibility](#), the following issues must be considered:

¹ On versions 17.x and before, Erlang/OTP was licensed under Erlang Public License 1.1, a modified license from Mozilla Public License. This is no longer applicable.

- Software with Apache License 2.0 is compatible with GPL version 3 (GPLv3), but the derived product *must be licensed under GPLv3*.
- Software with GPL version 2 (GPLv2) is *incompatible* with Apache License 2.0, citing the patent termination and indemnification provisions as restrictions not present in the older GPL license.

11.2.2 MIT and BSD licenses

Including MIT and BSD license code in Erlang/OTP does not require any special process, because both license allow relicensing of the software, under the restriction of keeping the original notices.

11.2.3 CC0 license (Public Domain)

Code explicitly stated as [Creative Commons CC0 licensed](#), which means equivalent to be placed in the Public Domain, can be freely relicensed.

11.3 Relicensing your code to Erlang/OTP

When OTP Team decides to include your code as a part of OTP, you need to comply the following rules:

- The code is relicensed as Apache License 2.0, and no other licenses are allowed in the code.
- The copyright notice in the code must be a simple one like this:

Copyright (c) 2017 by Kenji Rikitake.

Note that the notice has no “All Rights Reserved” or other words except the year and the name.

OTP Team will let you know the details on the exact conditions before merging your code, so that the legal issues will be cleared.

If you are using other authors’ algorithms and code, they must be compatible with Apache License 2.0 of Erlang/OTP. You may need to obtain written permissions from the original algorithm authors for specific relicensing to Erlang/OTP.

REFERENCE

This is a small reference documentation list for further learning and studying.

I believe the following list is also applicable to Elixir programmers as well as Erlang/OTP programmers. While Erlang/OTP books and documentations are focusing onto rather low-level implementations, Elixir programmers also have to know the details if they want to fully utilize the functionality of Erlang/OTP and BEAM¹.

I will not quote the books listed in the [Erlang/OTP manual](#). They are all essential. Read them first if you haven't.

The following blogs and small books will help you **a lot** (and even worth paying for):

- [Fréd Hebert's blog](#)
- [Loïc Hoguin's blog at Nine Nines](#)
- [Jesper Louis Andersen at Medium](#)
- [Brujo Benavides at Medium](#)
- [Fréd Hebert's free ebook: Erlang in Anger](#)
- [Loïc Hoguin's excellent ebook product: The Erlanger Playbook](#)

I've listed all my Erlang-related article references under my [Diigo bookmarks](#) with `erlang` tag. The [Diigo bookmarks](#) with `elixirlang` tag will also be useful.

You can find a Japanese version of preliminary document about writing a new Erlang/OTP module for beginners here in [Qiita](#).

¹ I believe Erlang/OTP programmers should learn all about Elixir language too, because many useful Web applications are written in Elixir, and the ecosystem including Plug and Phoenix are now widely used in the real-world production systems. A good starting point is here: <http://elixir-lang.org/learning.html>

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`