mruby

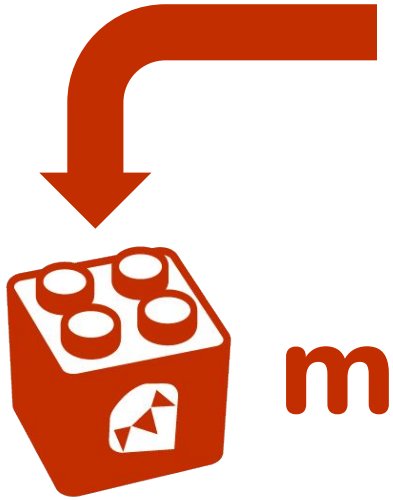
What to expect

What we’re going to see

- Intro to mruby and its ecosystem
- How to build mruby executables
- Presentation of a TUI framework built with mruby

This talk is beginner-friendly—no deep dives into C internals like GC, memory management, or cross-compilation

Wow a Lego block!
That must be easy

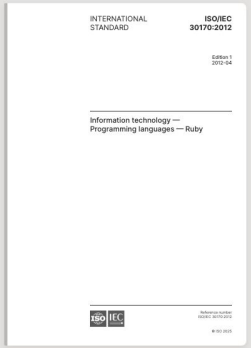


mruby

LOL

What is mruby?

mruby is the lightweight implementation of the Ruby language complying with part of the ISO standard.



Read sample

ISO/IEC 30170:2012

Information technology — Programming languages — Ruby

Published (Edition 1, 2012)

This publication was last reviewed and confirmed in 2021. Therefore this version remains current.

What is mruby?

- Short for Embedded Ruby
- Latest version: 3.4
- Lua equivalent
- Single threaded
- Only a thing in Japan
- Limitations:
  - https://github.com/mruby/mruby/blob/master/doc/limitations.md
  - No “require”
  - Anything outside the ISO standard probably not implemented
    - No pattern matching
    - No Struct keyword_init
    - …

Who is mruby for?

👨‍💻 **Embedded Developers (mruby & mruby/c)**

Microcontrollers, firmware, appliances

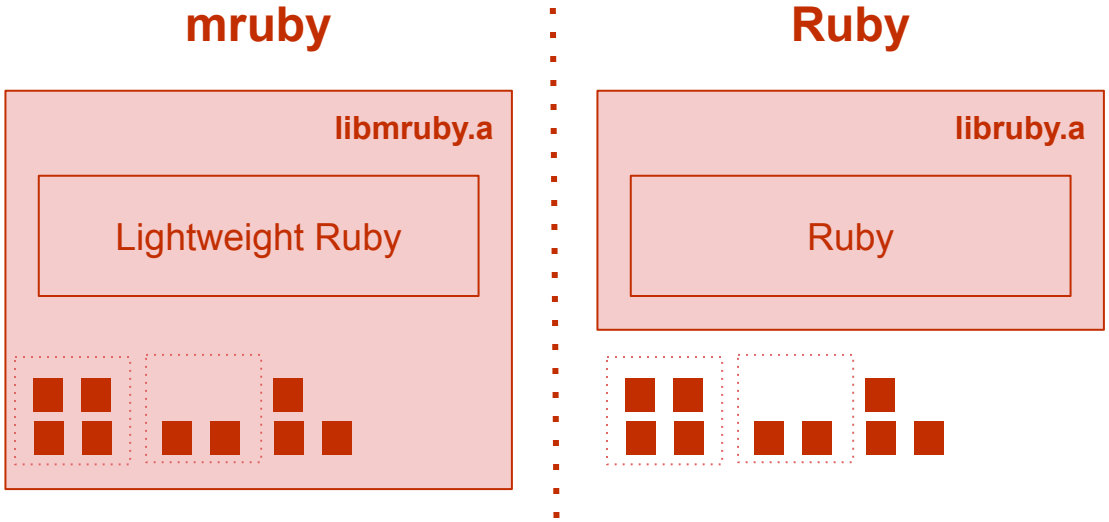
🛠️ **CLI & Tooling Developers**

Standalone executables

🎮 **Game & UI Developers**

2D games. Example: DragonRuby

mruby vs Ruby: Architecture and Philosophy



mruby is statically built

CRuby is dynamically extended

No require necessary

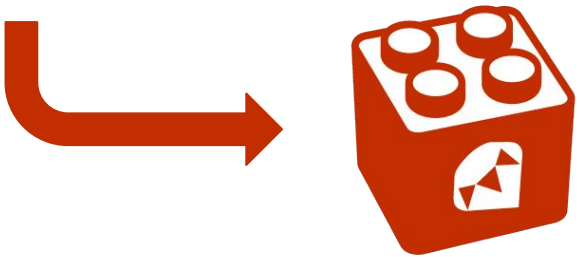
mrbgems

A mrbgem is a reusable component or library written in Ruby, C, or both. They are statically linked at build time, not dynamically loaded like regular Ruby gems.

It allows you to extend the mruby core:

- Create custom reusable Ruby code
- Create C extensions and bindings
- Configuration and initialization code

LIST OF MRBGEMS & DOCS



mrbgems

GEM Structure

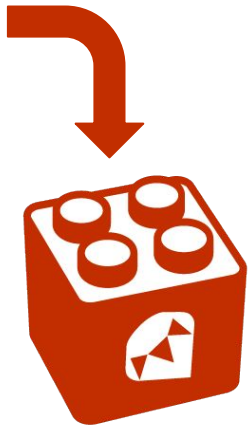
The maximal GEM structure looks like this:

```
+- GEM_NAME            <- Name of GEM
    |
    +- README.md       <- Readme for GEM
    |
    +- mrbgem.rake     <- GEM Specification
    |
    +- include/        <- Header for Ruby extension (will exported)
    |
    +- mrblib/         <- Source for Ruby extension
    |
    +- src/            <- Source for C extension
    |
    +- tools/          <- Source for Executable (in C)
    |
    +- test/           <- Test code (Ruby)
```

Gemboxes

A gembox is a Ruby file that defines a list of conf.gem entries—essentially a predefined gem bundle. https://github.com/mruby/mruby/tree/master/mrbgems

```
MRuby::GemBox.new do |conf|
  conf.gembox "stdlib"
  conf.gembox "stdlib-ext"
  conf.gembox "stdlib-io"
  conf.gembox "math"
  conf.gembox "metaprog"

  conf.gem :core => "mruby-bin-mrbc"     # Generate mrbc command
  conf.gem :core => "mruby-bin-debugger" # Generate mrdb command
  conf.gem :core => "mruby-bin-mirb"     # Generate mirb command
  conf.gem :core => "mruby-bin-mruby"    # Generate mruby command
  conf.gem :core => "mruby-bin-strip"    # Generate mruby-strip command
  conf.gem :core => "mruby-bin-config"   # Generate mruby-config command
end
```

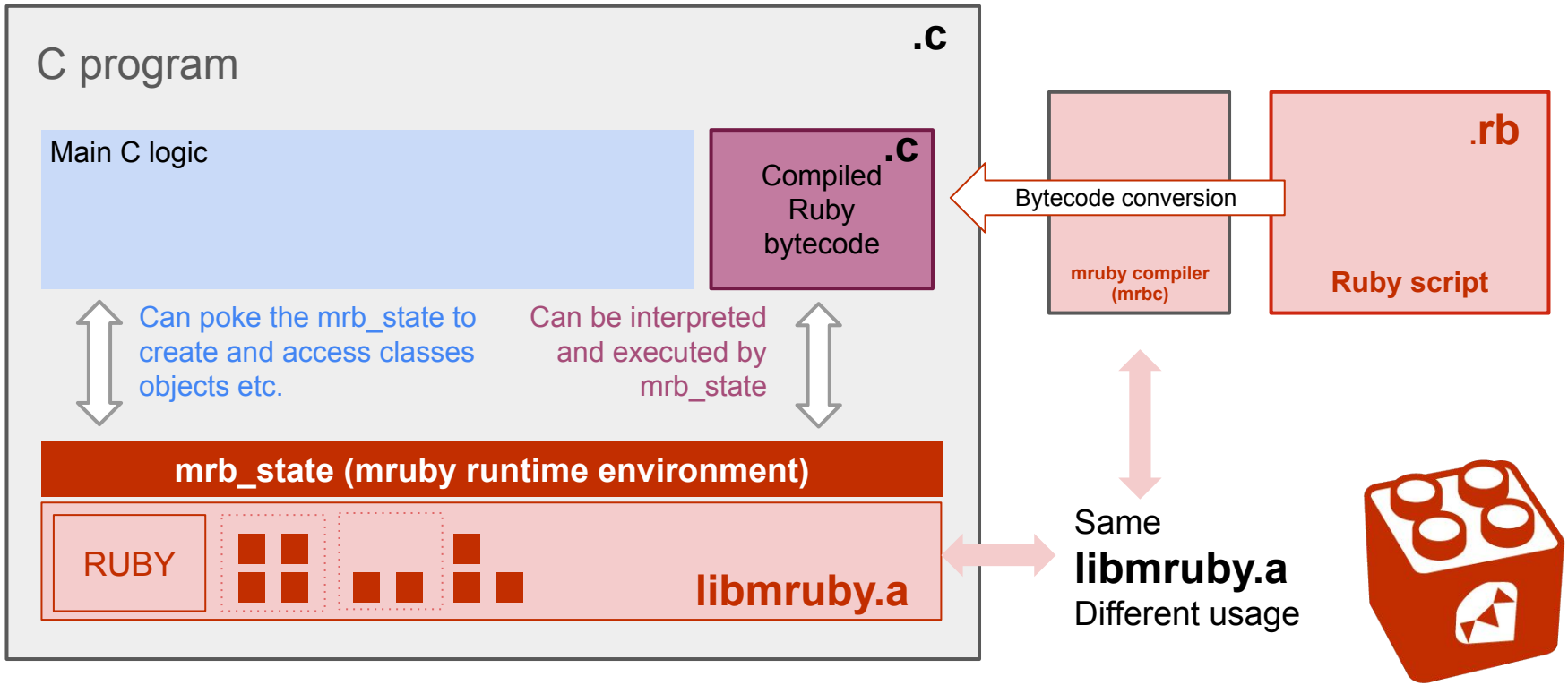


Building/Compiling mruby

```
MRuby::Build.new do |conf|
  toolchain :gcc

  conf.gembox 'default'
  # conf.gembox 'full-core'

  conf.gem github: 'iij/mruby-mtest'
  conf.gem "#{ MRUBY_ROOT }/.."

  # conf.gem "#{ MRUBY_ROOT }/../../mruby-termbox2"

  conf.cc.flags << '-g -O0 -fsanitize=address'
  conf.linker.flags << '-fsanitize=address'

  conf.enable_debug
  conf.enable_bintest
  conf.enable_test
end
```

Note, mruby can also be cross-compiled from one platform to another via a MRuby::CrossBuild

Embedding mruby in C



Embedding mruby in C

example-6.rb

```ruby
module Operations
  def self.add(n1, n2)
    n1 + n2
  end
end

puts "The value computed from the Ruby code is: #{Operations.add(12, 34)}"
```

```c
#include <mruby.h>
#include <mruby/irep.h>
#include "ruby_bytecode.c"

int main(void) {
  mrb_state* mrb = mrb_open();
  if (!mrb) { /* handle error */
  }

  mrb_load_irep(mrb, ruby_code);

  mrb_close(mrb);
  return 0;
}
```

example-6.rb | ruby_bytecode.c

```c
#include <stdint.h>
#ifdef __cplusplus
extern
#endif
const uint8_t ruby_code[] = {
0x52,0x49,0x54,0x45,0x30,0x33,0x30,0x30,0x00,0x00,0x01,0x07,0x4d,0x41,0x54,0x5a,
0x30,0x30,0x30,0x30,0x49,0x52,0x45,0x50,0x00,0x00,0x00,0xd1,0x30,0x33,0x30,0x30,
0x00,0x00,0x00,0x7d,0x00,0x01,0x00,0x07,0x00,0x01,0x00,0x00,0x00,0x00,0x00,0x21,
0x11,0x01,0x5d,0x01,0x00,0x5e,0x01,0x00,0x51,0x02,0x00,0x1d,0x03,0x00,0x03,0x04,
0x0c,0x03,0x05,0x22,0x2f,0x03,0x01,0x02,0x52,0x02,0x2d,0x01,0x02,0x01,0x38,0x01,
0x69,0x00,0x01,0x00,0x00,0x2a,0x54,0x68,0x65,0x20,0x76,0x61,0x6c,0x75,0x65,0x20,
0x63,0x6f,0x6d,0x70,0x75,0x74,0x65,0x64,0x20,0x66,0x72,0x6f,0x6d,0x20,0x74,0x68,
0x65,0x20,0x52,0x75,0x62,0x79,0x20,0x63,0x6f,0x64,0x65,0x20,0x69,0x73,0x3a,0x20,
0x00,0x00,0x03,0x00,0x0a,0x4f,0x70,0x65,0x72,0x61,0x74,0x69,0x6f,0x6e,0x73,0x00,
0x00,0x03,0x61,0x64,0x64,0x00,0x00,0x04,0x70,0x75,0x74,0x73,0x00,0x00,0x00,0x00,
0x26,0x00,0x01,0x00,0x03,0x00,0x01,0x00,0x00,0x00,0x00,0x00,0x0c,0x12,0x01,0x62,
0x01,0x58,0x02,0x00,0x5f,0x01,0x00,0x38,0x01,0x00,0x00,0x00,0x01,0x00,0x03,0x61,
0x64,0x64,0x00,0x00,0x00,0x00,0x22,0x00,0x04,0x00,0x07,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x0e,0x34,0x08,0x00,0x00,0x01,0x04,0x01,0x01,0x05,0x02,0x3c,0x04,0x38,
0x04,0x00,0x00,0x00,0x00,0x4c,0x56,0x41,0x52,0x00,0x00,0x00,0x1a,0x00,0x00,0x00,
0x02,0x00,0x02,0x6e,0x31,0x00,0x02,0x6e,0x32,0x00,0x00,0x00,0x01,0xff,0xff,0x45,
0x4e,0x44,0x00,0x00,0x00,0x00,0x08,
};
```

Creating a simple executable

1. Build mruby “image” with required dependencies
2. You create a ruby program (.rb)
3. You convert it into bytecode with mrbc
4. You inject the bytecode in your C program
5. You compile your C program
6. You run the executable

```c
#include <mruby.h>
#include <mruby/irep.h>
#include "ruby_bytecode.c"

int main(void) {
  mrb_state* mrb = mrb_open();
  if (!mrb) { /* handle error */
  }

  mrb_load_irep(mrb, ruby_code);

  mrb_close(mrb);
  return 0;
}
```

C Examples

Tui-rb framework

Built on top of Clay and Termbox2

Motivation: Pushing the use of Ruby to other platforms and use cases

- Termbox2
  - https://github.com/AlexB52/mruby-termbox2
  - https://github.com/termbox/termbox2
- Clay
  - https://github.com/AlexB52/mruby-clay
  - https://github.com/nicbarker/clay

Insights

- Testing and Debugging aren’t friendly
- Dependency management
- 1MB minimum size to include mruby in a binary
- AI helps a lot with onboarding
- References to mruby API is consistent (almost) but hard to navigate
  - A cheatsheet would be nice
  - Mruby-handbook is amazing

Aside: Everything is an object

```
/*
 * mrb_value representation:
 *
 * 64-bit word with inline float:
 *   nil   : ...0000 0000 (all bits are 0)
 *   false : ...0000 0100 (mrb_fixnum(v) != 0)
 *   true  : ...0000 1100
 *   undef : ...0001 0100
 *   symbol: ...0001 1100 (use only upper 32-bit as symbol value with MRB_64BIT)
 *   fixnum: ...IIII III1
 *   float : ...FFFF FF10 (51 bit significands; require MRB_64BIT)
 *   object: ...PPPP P000
 *
 * 32-bit word with inline float:
 *   nil   : ...0000 0000 (all bits are 0)
 *   false : ...0000 0100 (mrb_fixnum(v) != 0)
 *   true  : ...0000 1100
 *   undef : ...0001 0100
 *   symbol: ...SSS1 0100 (symbol occupies 20bits)
 *   fixnum: ...IIII III1
 *   float : ...FFFF FF10 (22 bit significands; require MRB_64BIT)
 *   object: ...PPPP P000
 *
 * and word boxing without inline float (MRB_WORDBOX_NO_FLOAT_TRUNCATE):
 *   nil   : ...0000 0000 (all bits are 0)
 *   false : ...0000 0100 (mrb_fixnum(v) != 0)
 *   true  : ...0000 1100
 *   undef : ...0001 0100
 *   fixnum: ...IIII III1
 *   symbol: ...SSSS SS10
 *   object: ...PPPP PP00 (any bits are 1)
 */
typedef struct mrb_value {
  uintptr_t w;
} mrb_value;
```

May contain immediate values (fixnums, symbols, true/false/nil), or **a pointer to a heap-allocated object** like `RString`, `RArray`, etc.

```
struct RVALUE {
  union {
    struct RVALUE_initializer init;
    struct free_obj free;
    struct RBasic basic;
    struct RObject object;
    struct RClass klass;
    struct RString string;
    struct RArray array;
    struct RHash hash;
    struct RRange range;
    struct RData data;
    struct RIStruct istruct;
    struct RProc proc;
    struct REnv env;
    struct RFiber fiber;
    struct RException exc;
    struct RBreak brk;
  } as;
};
```

Every object that needs GC management — like `RString`, `RArray`, `RClass`, etc. — is stored as an `RVALUE`.

Aside: mruby API

```c
// VALUES

MRB_INLINE mrb_value mrb_float_value(struct mrb_state *mrb, mrb_float f)
MRB_INLINE mrb_value mrb_nil_value(void)
MRB_INLINE mrb_value mrb_false_value(void)
MRB_INLINE mrb_value mrb_true_value(void)

// STRING

MRB_API mrb_value mrb_str_new(mrb_state *mrb, const char *p, size_t len)
MRB_API mrb_value mrb_str_new_cstr(mrb_state *mrb, const char *p)
_value mrb_str_new_lit(mrb_state *mrb, const char *p)

// SYMBOL

MRB_API mrb_sym mrb_intern_cstr(mrb_state *mrb, const char* str)
MRB_API mrb_sym mrb_intern(mrb_state *mrb, const char *name, size_t len)
MRB_API mrb_sym mrb_intern_lit(mrb_state *mrb, const char *name)
MRB_INLINE mrb_value mrb_symbol_value(mrb_sym i)


// ARRAYS

MRB_API mrb_value mrb_ary_new(mrb_state *mrb)
MRB_API mrb_value mrb_ary_new_capa(mrb_state *mrb, mrb_int capa)
MRB_API mrb_value mrb_ary_new_from_values(mrb_state *mrb, mrb_int size, const mrb_value *vals)

MRB_API void mrb_ary_push(mrb_state *mrb, mrb_value array, mrb_value value)
MRB_API mrb_value mrb_ary_pop(mrb_state *mrb, mrb_value ary)
MRB_API void mrb_ary_set(mrb_state *mrb, mrb_value ary, mrb_int n, mrb_value val)
MRB_API void mrb_ary_replace(mrb_state *mrb, mrb_value self, mrb_value other)

// HASHES

MRB_API mrb_value mrb_hash_new_capa(mrb_state *mrb, mrb_int capa)
MRB_API mrb_value mrb_hash_new(mrb_state *mrb)
MRB_API void mrb_hash_set(mrb_state *mrb, mrb_value hash, mrb_value key, mrb_value val)
MRB_API mrb_value mrb_hash_get(mrb_state *mrb, mrb_value hash, mrb_value key)
```

```c
// CLASSES & MODULES

MRB_API struct RClass* mrb_define_class(mrb_state *mrb, const char * name, struct RClass *super)
MRB_API struct RClass* mrb_define_module(mrb_state *mrb, const char *name)

MRB_API void mrb_define_method(mrb_state *mrb, struct RClass *c, const char *name, mrb_func_t func, mrb_aspec aspec)
MRB_API void mrb_define_class_method(mrb_state *mrb, struct RClass * c, const char *name, mrb_func_t func, mrb_aspec aspec)
MRB_API void mrb_define_singleton_method(mrb_state *mrb, struct RObject *o, const char *name, mrb_func_t func, mrb_aspec aspec)
MRB_API void mrb_define_module_function(mrb_state *mrb, struct RClass *c, const char *name, mrb_func_t func, mrb_aspec aspec)

MRB_API mrb_value mrb_obj_new(mrb_state *mrb, struct RClass *c, mrb_int argc, const mrb_value *argv) ;

// CONSTANTS

MRB_API mrb_value mrb_const_get(mrb_state *mrb, mrb_value mod, mrb_sym sym)
MRB_API void mrb_const_set(mrb_state *mrb, mrb_value mod, mrb_sym sym, mrb_value v)
MRB_API mrb_bool mrb_const_defined(mrb_state *mrb, mrb_value mod, mrb_sym id)
MRB_API void mrb_const_remove(mrb_state *mrb, mrb_value mod, mrb_sym sym)


// INSTANCE VARIABLES

MRB_API mrb_value mrb_obj_iv_get(mrb_state *mrb, struct RObject *obj , mrb_sym sym)
MRB_API mrb_value mrb_iv_get(mrb_state *mrb, mrb_value obj, mrb_sym sym)
MRB_API void mrb_obj_iv_set(mrb_state *mrb, struct RObject *obj, mrb_sym sym, mrb_value v)
MRB_API void mrb_iv_set(mrb_state *mrb, mrb_value obj, mrb_sym sym, mrb_value v)
MRB_API mrb_bool mrb_obj_iv_defined(mrb_state *mrb, struct RObject * obj, mrb_sym sym)
MRB_API mrb_bool mrb_iv_defined(mrb_state *mrb, mrb_value obj, mrb_sym sym)
MRB_API mrb_value mrb_iv_remove(mrb_state *mrb, mrb_value obj, mrb_sym sym)
```

Questions?