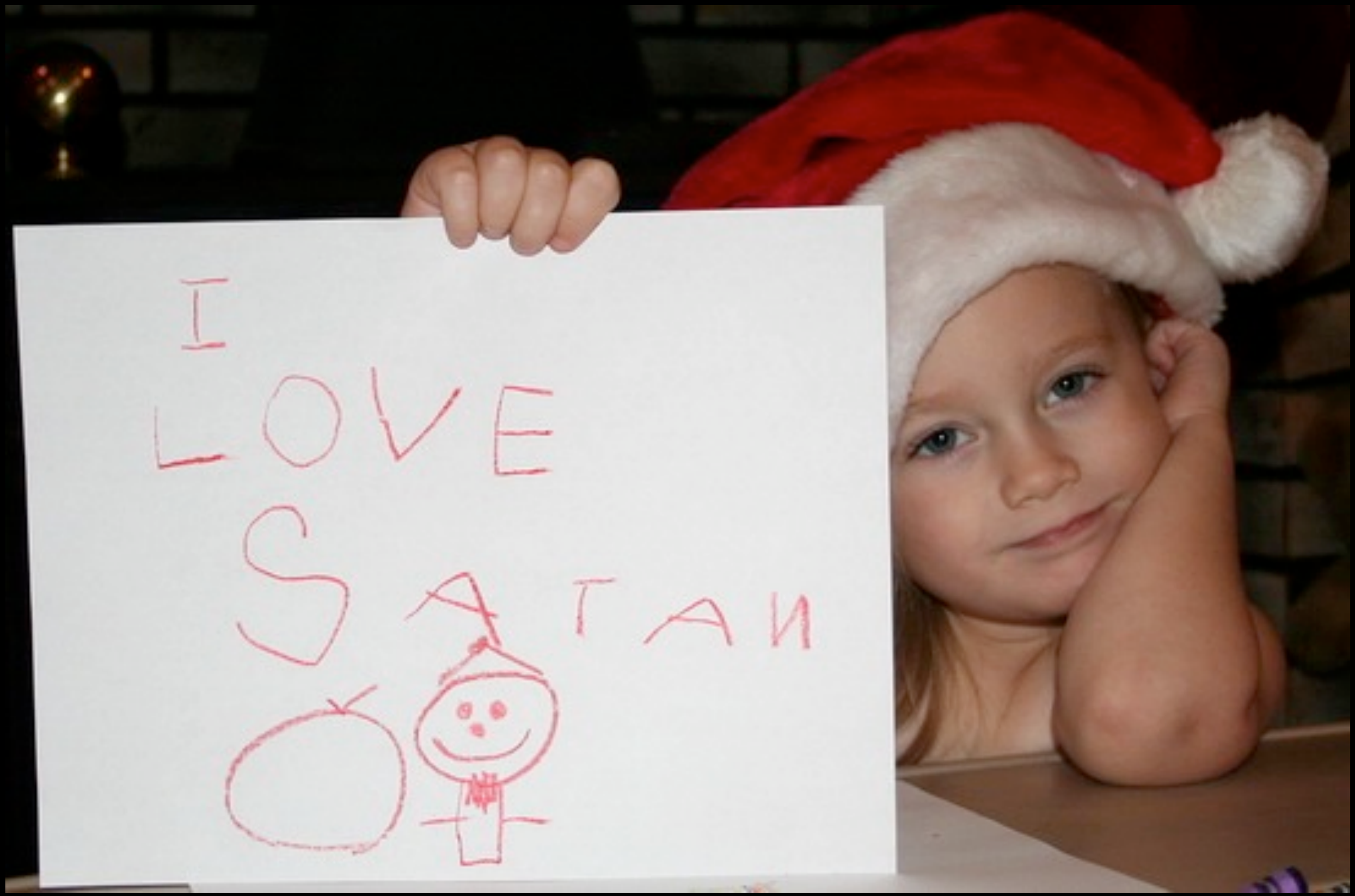
Happy Independence Day


I
LOVE
SATAN

Ruby

C Extensions

Matt Todd

GeoIP

```c
#include <GeoIP.h>

GeoIP *gi;
char *name = NULL;

gi = GeoIP_open("GeoIPOrg.dat", GEOIP_MEMORY_CACHE);
name = GeoIP_name_by_addr(gi, "24.24.24.24");
```

```ruby
db = GeoIP::Organization.new("GeoIPOrg.dat", :memory)
name = db.look_up('24.24.24.24')
```

Basics

```c
// include the Ruby objects, data types, macros, and mechanisms
#include "ruby.h"
```

```c
// define Ruby value variable
VALUE HelloWorld = Qnil;
```

```c
// prototype the init function that defines the extension class
void Init_hello_world();
```

```c
// extension initialization function
void Init_hello_world() {
  // class constant              Ruby constant name  parent class
  HelloWorld = rb_define_class("HelloWorld",        rb_cObject);

  // Init other classes/modules
}
```

```
// include the Ruby objects, data types, macros, and mechanisms
#include "ruby.h"
```

defines classes, keywords, values, functions, macros

```
// define Ruby value variable
VALUE HelloWorld = Qnil;
```

creates variable to hold class definition

```
// prototype the init function that defines the extension class
void Init_hello_world();
```

prototype for init function

init function defines Ruby class interface and any methods

```
// extension initialization function
void Init_hello_world() {
  // class constant              Ruby constant name  parent class
  HelloWorld = rb_define_class("HelloWorld",        rb_cObject);

  // Init other classes/modules
}
```

calls other init funcs

API

```
ReceiverClass =
rb_define_class(
   "RubyClassName",
   rb_cParentClass);
```

```
ReceiverModule =
rb_define_module(
   "RubyModuleName")
```

```
SubReceiverClass =
rb_define_class_under(
    ReceiverClass
    "SubRubyClassName",
    rb_cParentClass);
```

```
rb_define_method(
  ReceiverClass,
  "ruby_method_name",
  implementation_function,
  number_of_arguments);
```

implementation_function

Returns a Ruby value

the receiver

```
VALUE implementation_function(VALUE self) {
    return Qnil; return nil
}
```

number_of_arguments

```
0..17 =>
VALUE func(VALUE self, VALUE param1, ...);

-1 =>
VALUE func(int argc, VALUE *args, VALUE self);

-2 =>
VALUE func(VALUE self, VALUE *args);
```

```
rb_scan_args(
    int argc, VALUE *args,
    char *format,
vars...);
```

```
format  /(\d)?(\d)?(\*)?(&)?/
        number_of_mandatory_args
        number_of_optional_args
        splat?
        block?
```

```
"1"     => def meth(a)
"01"    => def meth(a = {})
"&"     => def meth(&block)
"11&"   => def meth(a, b = {}, &block)
"1*"    => def meth(a, *rest)
```

usage

```
rb_define_method(Foo, "bar", rb_foo_bar, -1);

VALUE rb_foo_bar(int argc, VALUE *args, VALUE self) {
  VALUE a, b;
  rb_scan_args(argc, args, "11", &a, &b);
  // default to 10 if b not supplied
  if(NIL_P(b)) { b = 10; }
  return a * b;
}
```

macros

```
NIL_P(VALUE v); // #nil?

// C => Ruby
VALUE INT2NUM(int i);
VALUE rb_str_new2(char *str);
VALUE rb_float_new(double d);

// Ruby => C
int NUM2INT(VALUE i);
double NUM2DBL(VALUE i);
char* STR2CSTR(VALUE str);
```

Ruby types

```
VALUE str = rb_str_new2("yo");

VALUE ary = rb_ary_new();
rb_ary_push(ary, VALUE a);
VALUE b = rb_ary_pop(ary);

VALUE hash = rb_hash_new();
rb_hash_aset(hash,
   VALUE key, VALUE a);
rb_hash_aref(hash, VALUE key);
```

evaluation

```
VALUE rb_eval_string(
    char *string);
```

```
VALUE rb_funcall(
    VALUE receiver,
    ID rb_intern("meth"),
    int argc, VALUE *args);
```

Examples

```c
#include "ruby.h"

VALUE ClassesAndModulesSample = Qnil;

VALUE Foo = Qnil;
VALUE Foo_Base = Qnil;
VALUE Bar = Qnil;

VALUE ActsAsFoo = Qnil;
VALUE ActsAsFoo_ClassMethods = Qnil;

void Init_classes_and_modules_sample();

// Foo

VALUE rb_Foo_Base_find_b(VALUE self) {
  return Qtrue;
}

void Init_foo() {
  Foo = rb_define_class("Foo", rb_cObject);

  Foo_Base = rb_define_class_under(Foo, "Base", rb_cObject);
  rb_define_method(Foo_Base, "foo!", rb_Foo_Base_find_b, 0);
}

// Bar

void Init_bar() {
  Bar = rb_define_class("Bar", Foo_Base);
}
```

```c
// ActsAsFoo

VALUE rb_ActsAsFoo_foo_q(VALUE self) {
  return Qtrue;
}

VALUE rb_ActsAsFoo_included(VALUE self, VALUE target) {
  rb_extend_object(target, ActsAsFoo_ClassMethods);
  return self;
}

void Init_acts_as_foo() {
  ActsAsFoo = rb_define_module("ActsAsFoo");
  rb_define_method(ActsAsFoo, "foo?", rb_ActsAsFoo_foo_q, 0);
  rb_define_singleton_method(ActsAsFoo, "included", rb_ActsAsFoo_included, 1);

  ActsAsFoo_ClassMethods = rb_define_module_under(ActsAsFoo, "ClassMethods");
  rb_define_method(ActsAsFoo_ClassMethods, "foo?", rb_ActsAsFoo_foo_q, 0);

  rb_include_module(Bar, ActsAsFoo);
  // call ActsAsFoo.included(Bar) manually since rb_include_module doesn't
  rb_funcall(ActsAsFoo, rb_intern("included"), 1, Bar);
}

////////////////////////////////////////////////////////////////////////

void Init_classes_and_modules_sample() {
  ClassesAndModulesSample = rb_define_class("ClassesAndModulesSample",
rb_cObject);

  Init_foo();
  Init_bar(); // class Bar < Foo::Base; end

  Init_acts_as_foo();
}
```

```ruby
class Foo
  class Base
    def foo!
      # deep, dark magic here
      true
    end
  end
end
```

```ruby
class Bar < Foo::Base; end

@bar = Bar.new
@bar.foo! #=> true
```

```c
#include "ruby.h"

VALUE ClassesAndModulesSample = Qnil;

VALUE Foo = Qnil;
VALUE Foo_Base = Qnil;
VALUE Bar = Qnil;

void Init_classes_and_modules_sample();

VALUE rb_Foo_Base_find_b(VALUE self) {
  return Qtrue;
}

void Init_foo() {
  Foo = rb_define_class("Foo", rb_cObject);

  Foo_Base = rb_define_class_under(Foo, "Base", rb_cObject);
  rb_define_method(Foo_Base, "foo!", rb_Foo_Base_find_b, 0);
}

void Init_bar() {
  Bar = rb_define_class("Bar", Foo_Base);
}

void Init_classes_and_modules_sample() {
  ClassesAndModulesSample = rb_define_class("ClassesAndModulesSample", rb_cObject);

  Init_foo();
  Init_bar(); // class Bar < Foo::Base; end
}
```

class Foo < Object

```c
#include "ruby.h"

VALUE ClassesAndModulesSample = Qnil;

VALUE Foo = Qnil;
VALUE Foo_Base = Qnil;
VALUE Bar = Qnil;

void Init_classes_and_modules_sample();

VALUE rb_Foo_Base_find_b(VALUE self) {
  return Qtrue;
}

void Init_foo() {
  Foo = rb_define_class("Foo", rb_cObject);

  Foo_Base = rb_define_class_under(Foo, "Base", rb_cObject);
  rb_define_method(Foo_Base, "foo!", rb_Foo_Base_find_b, 0);
}

void Init_bar() {
  Bar = rb_define_class("Bar", Foo_Base);
}

void Init_classes_and_modules_sample() {
  ClassesAndModulesSample = rb_define_class("ClassesAndModulesSample", rb_cObject);

  Init_foo();
  Init_bar(); // class Bar < Foo::Base; end
}
```

```c
#include "ruby.h"

VALUE ClassesAndModulesSample = Qnil;

VALUE Foo = Qnil;
VALUE Foo_Base = Qnil;
VALUE Bar = Qnil;

void Init_classes_and_modules_sample();

VALUE rb_Foo_Base_find_b(VALUE self) {
  return Qtrue;
}

void Init_foo() {
  Foo = rb_define_class("Foo", rb_cObject);

  Foo_Base = rb_define_class_under(Foo, "Base", rb_cObject);
  rb_define_method(Foo_Base, "foo!", rb_Foo_Base_find_b, 0);
}

void Init_bar() {
  Bar = rb_define_class("Bar", Foo_Base);
}

void Init_classes_and_modules_sample() {
  ClassesAndModulesSample = rb_define_class("ClassesAndModulesSample", rb_cObject);

  Init_foo();
  Init_bar(); // class Bar < Foo::Base; end
}
```

```
class Foo < Object
  class Base
```

```
#include "ruby.h"

VALUE ClassesAndModulesSample = Qnil;

VALUE Foo = Qnil;
VALUE Foo_Base = Qnil;
VALUE Bar = Qnil;

void Init_classes_and_modules_sample();

VALUE rb_Foo_Base_find_b(VALUE self) {
  return Qtrue;
}

void Init_foo() {
  Foo = rb_define_class("Foo", rb_cObject);

  Foo_Base = rb_define_class_under(Foo, "Base", rb_cObject);
  rb_define_method(Foo_Base, "foo!", rb_Foo_Base_find_b, 0);
}

void Init_bar() {
  Bar = rb_define_class("Bar", Foo_Base);
}

void Init_classes_and_modules_sample() {
  ClassesAndModulesSample = rb_define_class("ClassesAndModulesSample", rb_cObject);

  Init_foo();
  Init_bar(); // class Bar < Foo::Base; end
}
```

```
class Foo < Object
  class Base
    def foo!
```

```c
#include "ruby.h"

VALUE ClassesAndModulesSample = Qnil;

VALUE Foo = Qnil;
VALUE Foo_Base = Qnil;
VALUE Bar = Qnil;

void Init_classes_and_modules_sample();

VALUE rb_Foo_Base_find_b(VALUE self) {
  return Qtrue;
}

void Init_foo() {
  Foo = rb_define_class("Foo", rb_cObject);

  Foo_Base = rb_define_class_under(Foo, "Base", rb_cObject);
  rb_define_method(Foo_Base, "foo!", rb_Foo_Base_find_b, 0);
}

void Init_bar() {
  Bar = rb_define_class("Bar", Foo_Base);
}

void Init_classes_and_modules_sample() {
  ClassesAndModulesSample = rb_define_class("ClassesAndModulesSample", rb_cObject);

  Init_foo();
  Init_bar(); // class Bar < Foo::Base; end
}
```

```ruby
module ActsAsFoo
  def foo?
    true
  end

  def included(target)
    target.extend ClassMethods
  end
  module ClassMethods
    def foo?
      true
    end
  end
end
```

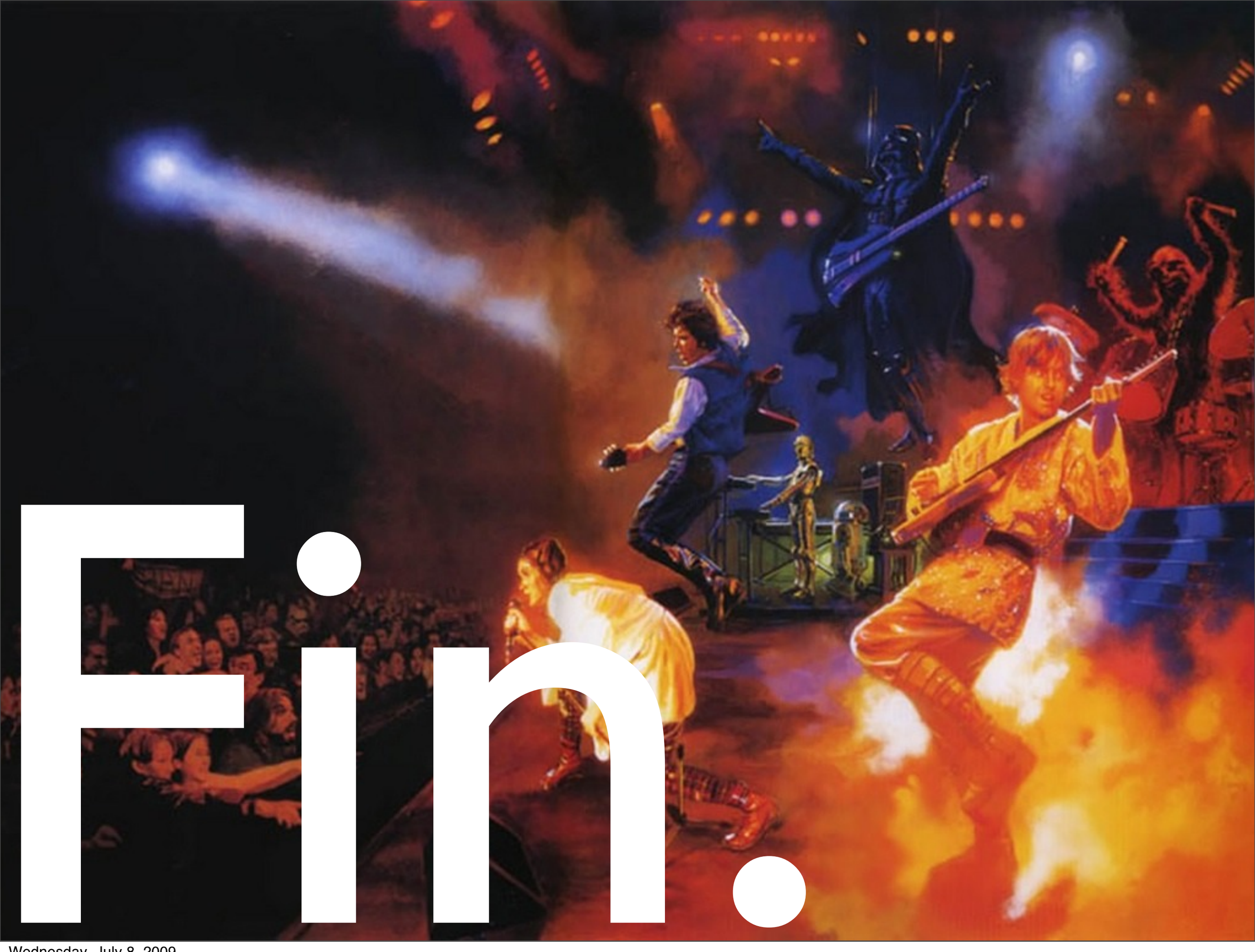
```ruby
class Bar < Foo
  include ActsAsFoo
end
```

```ruby
Bar.foo? #=> true
@bar = Bar.new
@bar.foo? #=> true
```

```c
#include "ruby.h"

VALUE ActsAsFoo = Qnil;
VALUE ActsAsFoo_ClassMethods = Qnil;

VALUE rb_ActsAsFoo_foo_q(VALUE self) {
  return Qtrue;
}

VALUE rb_ActsAsFoo_included(VALUE self, VALUE target) {
  rb_extend_object(target, ActsAsFoo_ClassMethods);
  return self;
}

void Init_acts_as_foo() {
  ActsAsFoo = rb_define_module("ActsAsFoo");
  rb_define_method(ActsAsFoo, "foo?", rb_ActsAsFoo_foo_q, 0);
  rb_define_singleton_method(ActsAsFoo, "included", rb_ActsAsFoo_included, 1);

  ActsAsFoo_ClassMethods = rb_define_module_under(ActsAsFoo, "ClassMethods");
  rb_define_method(ActsAsFoo_ClassMethods, "foo?", rb_ActsAsFoo_foo_q, 0);
}
```

```c
#include "ruby.h"

VALUE ActsAsFoo = Qnil;
VALUE ActsAsFoo_ClassMethods = Qnil;

VALUE rb_ActsAsFoo_foo_q(VALUE self) {
  return Qtrue;
}

VALUE rb_ActsAsFoo_included(VALUE self, VALUE target) {
  rb_extend_object(target, ActsAsFoo_ClassMethods);
  return self;
}

void Init_acts_as_foo() {
  ActsAsFoo = rb_define_module("ActsAsFoo");
  rb_define_singleton_method(ActsAsFoo, "included", rb_ActsAsFoo_included, 1);

  ActsAsFoo_ClassMethods = rb_define_module_under(ActsAsFoo, "ClassMethods");
  rb_define_method(ActsAsFoo_ClassMethods, "foo?", rb_ActsAsFoo_foo_q, 0);
}
```

```
rb_include_module(Bar, ActsAsFoo);
// call ActsAsFoo.included(Bar) manually since rb_include_module doesn't
rb_funcall(ActsAsFoo, rb_intern("included"), 1, Bar);
```



Fin.

http://github.com/mtodd/ruby-c

Resources

http://rubycentral.com/pickaxe/ext_ruby.html