# CommonRuby - Feature #12732

## An option to pass to `Integer`, `Float`, to return `nil` instead of raise an exception

09/07/2016 05:13 AM - tenderlovemaking (Aaron Patterson)

| | |
|---|---|
| **Status:** | Closed |
| **Priority:** | Normal |
| **Assignee:** | matz (Yukihiro Matsumoto) |
| **Target version:** | |

**Description**

I would like to be able to pass an option to Integer() and Float() so that they don't raise an exception, but return nil instead. For example:

```
Integer(string, exception: false)
```

The reason I want this function is so that I can convert strings from YAML or JSON to integers if they parse correctly, or just return strings if they can't be parsed.

**Related issues:**

| | |
|---|---|
| Related to Ruby master - Feature #12968: Allow default value via block for In... | **Open** |

### History

#### #1 - 09/07/2016 06:55 AM - tenderlovemaking (Aaron Patterson)

*- File integer-parse.pdf added*

Adding a slide to show code I'm actually writing vs want to write

#### #2 - 10/11/2016 11:16 AM - shyouhei (Shyouhei Urabe)

We looked at this issue in developer meeting today.

It seems originally, ruby was designed under assumption that string to integer conversion in general could be covered 100% by either to_s or Integer(). Truth is we need the proposed functionality.

People at the meeting was not sure about the API though. Is it a variant of Integer() or a separate new method? For instance an attendee suggested "Integer?()" but could not be popular.

#### #3 - 11/22/2016 10:08 AM - shyouhei (Shyouhei Urabe)

*- Related to Feature #12968: Allow default value via block for Integer(), Float() and Rational() added*

#### #4 - 11/25/2016 07:01 AM - matz (Yukihiro Matsumoto)

*- Assignee set to matz (Yukihiro Matsumoto)*

*- Status changed from Open to Feedback*

Is there any problem with the following code?

```
Integer(str) rescue default_value
```

Matz.

#### #5 - 11/25/2016 09:34 AM - naruse (Yui NARUSE)

Below is PoC; it may have a path which raises an exception.

```
diff --git a/object.c b/object.c
index 05bef4d..5d63803 100644
--- a/object.c
+++ b/object.c
@@ -2750,17 +2750,60 @@ static VALUE
 rb_f_integer(int argc, VALUE *argv, VALUE obj)
 {
     VALUE arg = Qnil;
+    VALUE opts = Qnil;
+    VALUE exception = Qnil;
```

```
+    VALUE vbase = Qundef;
     int base = 0;
+    static ID int_kwds[1];

-    switch (argc) {
-      case 2:
-   base = NUM2INT(argv[1]);
-      case 1:
-   arg = argv[0];
-   break;
-      default:
-   /* should cause ArgumentError */
-   rb_scan_args(argc, argv, "11", NULL, NULL);
+    rb_scan_args(argc, argv, "11:", &arg, &vbase, &opts);
+    if (!NIL_P(vbase)) {
+   base = NUM2INT(vbase);
+    }
+    if (!NIL_P(opts)) {
+   if (!int_kwds[0]) {
+       int_kwds[0] = rb_intern_const("exception");
+   }
+   if (rb_get_kwargs(opts, int_kwds, 0, 1, &exception)) {
+       VALUE tmp;
+       if (RB_FLOAT_TYPE_P(arg)) {
+       double f;
+       if (base != 0) goto arg_error;
+       f = RFLOAT_VALUE(arg);
+       if (FIXABLE(f)) return LONG2FIX((long)f);
+       return rb_dbl2big(f);
+       }
+       else if (RB_INTEGER_TYPE_P(arg)) {
+       if (base != 0) goto arg_error;
+       return arg;
+       }
+       else if (RB_TYPE_P(arg, T_STRING)) {
+       const char *s;
+       long len;
+       rb_must_asciicompat(arg);
+       RSTRING_GETMEM(arg, s, len);
+       tmp = rb_cstr_parse_inum(s, len, NULL, base);
+       if (NIL_P(tmp)) {
+           return exception;
+       }
+       return tmp;
+       }
+       else if (NIL_P(arg)) {
+       if (base != 0) goto arg_error;
+       return exception;
+       }
+       if (base != 0) {
+       tmp = rb_check_string_type(arg);
+       if (!NIL_P(tmp)) return rb_str_to_inum(tmp, base, TRUE);
+arg_error:
+       rb_raise(rb_eArgError, "base specified for non string value");
+       }
+       tmp = convert_type(arg, "Integer", "to_int", FALSE);
+       if (NIL_P(tmp)) {
+       return rb_to_integer(arg, "to_i");
+       }
+       return tmp;
+   }
     }
     return rb_convert_to_integer(arg, base);
 }
```

```
def assert(a, b)
  if a != b
    raise "'#{a}' != '#{b}'"
  end
end
def assert_raise(ex)
  begin
    yield
    raise "#{ex} is expected but not raised"
  rescue ex
```

```
    # correct
  rescue
    raise "#{ex} is expected but #{$!.inspect}"
  end
end
o = Object.new
assert 123, Integer("123")
assert 50, Integer("32", 16)
assert 16, Integer("10", 16, exception: o)
assert o, Integer("x", exception: o)
assert o, Integer("x", 16, exception: o)
assert_raise(ArgumentError){ Integer("x") }
assert_raise(ArgumentError){ Integer("x", 16) }

require'benchmark/ips'
Benchmark.ips{|x|
  x.report("rescue") {
    Integer('foo') rescue nil
  }
  x.report("kwarg") {
    Integer('foo', exception: nil)
  }
}

Warming up --------------------------------------
              rescue    36.258k i/100ms
               kwarg    64.004k i/100ms
Calculating -------------------------------------
              rescue    392.926k (± 8.9%) i/s -      1.958M in   5.025204s
               kwarg    844.563k (±14.9%) i/s -      4.096M in   5.017539s
```

#### #6 - 11/28/2016 11:01 PM - tenderlovemaking (Aaron Patterson)

Hi,

> Is there any problem with the following code?
> Integer(str) rescue default_value

2 problems

1. It's slower than it could be (as Naruse demonstrates)
2. It's very noisy when -d is enabled.

In Psych, I am trying to avoid noise from -d. That means I have to try to check if the string will work with Integer(), then actually call Integer(). It means the string has to be parsed twice. If Integer(str) rescue default_value didn't make noise, then I would be OK with that. :)

#### #7 - 02/07/2017 11:56 AM - rbjl (Jan Lelis)

Although it does not solve Aaron's use case, I would suggest to have a Integer.try_convert, Float.try_convert, Rational.try_convert, and Complex.try_convert which do not raise exceptions, but just return nil. To keep consistency, they would just call implicit, then explicit conversion (e.g. to_int, then to_i), instead of Integer()'s special parsing.

It only allows strict parsing, but is much cleaner, imho. Also, it would fill some empty spots in the core conversion table and make Ruby's conversion logic simpler. (.try_convert, which currently feels more like an implementation detail, could then be embraced more).

To allow Integer() special conversion, it would still need an exception: option, but also Float(), Rational(), and Complex() would need it (since they currently also lack this feature due to not having a try_convert). One idea is to give every of the uppercased Kernel methods an exception: option, but this does not make sense for Array() and just would not be needed if going for the broader try_convert support).

To summarize my suggestion in two bullet points:

- Create Integer.try_convert, Float.try_convert, Rational.try_convert, and Complex.try_convert which prefer implicit conversion (if available), then explicit conversion, but return nil instead of raising an exception
- Give Integer() an exception option to support special integer parsing without exceptions, but do not give Float() one

#### #8 - 01/24/2018 07:54 AM - matz (Yukihiro Matsumoto)

Aaron's comment in #note-6 sounds reasonable. Accepted.

Matz.

#### #9 - 01/24/2018 08:00 AM - knu (Akinori MUSHA)

Just for the record, Integer(x, rescue: default_value) might be an idea, if anything other than nil (like zero) would be desired.

#### #10 - 02/07/2018 07:17 PM - tenderlovemaking (Aaron Patterson)

Something like Integer(x, rescue: default_value) is fine for me too, (or Integer(x, ->() { default_value }), which is similar to [].find(->() { missing_value }) { ... }) Configuring with a default value seems more flexible.

#### #11 - 02/08/2018 12:49 PM - nobu (Nobuyoshi Nakada)

Since Integer() has radix optional argument, new optional argument might be confusing.
A keyword argument or a block would be better, I think.

#### #12 - 02/08/2018 05:47 PM - enebo (Thomas Enebo)

Two comments:

1. having block form only defeats any performance gain as executing blocks have a measurable cost. It may be nice to have though in addition to simple nil return form.
2. it would be really nice if Ruby had some API consistency for non-exception variants of the various callswhich have made this change. Doing each one of these as a one-off discussion almost destines these APIs to not be consistent.

#### #13 - 03/15/2018 07:19 AM - mrkn (Kenta Murata)

*- Status changed from Feedback to Closed*

Applied in changeset ruby-trunk:trunk|r62757.

---

Add exception: keyword in Kernel#Integer()

Support exception: keyword argument in Kernel#Integer().
If exception: is false, Kernel#Integer() returns nil if the given
value cannot be interpreted as an integer value.
The default value of exception: is true.
This is part of [Feature #12732].

#### #14 - 05/29/2018 11:35 AM - m_s__santos (Matheus Silva)

rbjl (Jan Lelis) wrote:

> Although it does not solve Aaron's use case, I would suggest to have a Integer.try_convert, Float.try_convert, Rational.try_convert, and Complex.try_convert which do not raise exceptions, but just return nil. To keep consistency, they would just call implicit, then explicit conversion (e.g. to_int, then to_i), instead of Integer()'s special parsing.
>
> It only allows strict parsing, but is much cleaner, imho. Also, it would fill some empty spots in the core conversion table and make Ruby's conversion logic simpler. (.try_convert, which currently feels more like an implementation detail, could then be embraced more).
>
> To allow Integer() special conversion, it would still need an exception: option, but also Float(), Rational(), and Complex() would need it (since they currently also lack this feature due to not having a try_convert). One idea is to give every of the uppercased Kernel methods an exception: option, but this does not make sense for Array() and just would not be needed if going for the broader try_convert support).
>
> To summarize my suggestion in two bullet points:
>
> - Create Integer.try_convert, Float.try_convert, Rational.try_convert, and Complex.try_convert which prefer implicit conversion (if available), then explicit conversion, but return nil instead of raising an exception
> - Give Integer() an exception option to support special integer parsing without exceptions, but do not give Float() one

It would be better if Integer.try_convert return the conversion or the value passed if it can't convert.

### Files

| | | | |
|---|---|---|---|
| integer-parse.pdf | 29 KB | 09/07/2016 | tenderlovemaking (Aaron Patterson) |