Ruby trunk - Feature #4840

Allow returning from require

06/06/2011 11:11 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Status:	Closed	
Priority:	Normal	
Assignee:	nobu (Nobuyoshi Nakada)	
Target version:		

Description

I have a situation where there is some code in Gitorious like:

```
unless defined? GitoriousConfig
  # tons of lines here
end
```

And I would it to look like

```
return if defined? GitoriousConfig
```

```
#tons of lines here
```

It would be great to allow return from a required file or some other keyword (break, etc or a new one)

Related issues:

Related to Ruby trunk - Bug #13678: toplevel return causes [BUG]	**Closed**
Related to Ruby trunk - Bug #13682: return inside of ensure causes [BUG]	**Closed**
Related to Ruby trunk - Bug #13755: Null pointer dereference in hash_table_in...	**Closed**
Related to Ruby trunk - Bug #14062: Top-level return allows an argument	**Open**

Associated revisions

Revision 342f10b9 - 12/21/2016 01:58 AM - nobu (Nobuyoshi Nakada)

compile.c: toplevel return

- compile.c (iseq_compile_each): stop execution of the current source by toplevel return. [ruby-core:36785] [Feature #4840]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@57132 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 57132 - 12/21/2016 01:58 AM - nobu (Nobuyoshi Nakada)

compile.c: toplevel return

- compile.c (iseq_compile_each): stop execution of the current source by toplevel return. [ruby-core:36785] [Feature #4840]

Revision 57132 - 12/21/2016 01:58 AM - nobu (Nobuyoshi Nakada)

compile.c: toplevel return

- compile.c (iseq_compile_each): stop execution of the current source by toplevel return. [ruby-core:36785] [Feature #4840]

Revision 57132 - 12/21/2016 01:58 AM - nobu (Nobuyoshi Nakada)

compile.c: toplevel return

- compile.c (iseq_compile_each): stop execution of the current source by toplevel return. [ruby-core:36785] [Feature #4840]

History

#1 - 06/07/2011 12:23 AM - judofyr (Magnus Holm)

I think using "return" is quite natural in this case, as long as we
can use it in module/class-definitions too.

// Magnus Holm

On Mon, Jun 6, 2011 at 16:18, Michael Edgar adgar@carboni.ca wrote:

> On Jun 6, 2011, at 10:11 AM, Rodrigo Rosenfeld Rosas wrote:
>
> > It would be great to allow return from a required file or some other keyword (break, etc or a new one)
>
> This could be implemented as a method that raises an exception that require and load rescue.
> I'm not sure how compatible that would be with custom require implementations (rubygems,
> polyglot, etc), but it would obviate the need for a new keyword or commandeering an existing one.
>
> Michael Edgar
> adgar@carboni.ca
> http://carboni.ca/

#2 - 06/07/2011 07:23 AM - cjheath (Clifford Heath)

On 07/06/2011, at 12:18 AM, Michael Edgar wrote:

> On Jun 6, 2011, at 10:11 AM, Rodrigo Rosenfeld Rosas wrote:
>
> > It would be great to allow return from a required file or some
> >
> > other keyword (break, etc or a new one)
>
> This could be implemented as a method that raises an exception that
>
> require and load rescue.
> I'm not sure how compatible that would be with custom require
>
> implementations (rubygems,
> polyglot, etc)

Polyglot will pass all exceptions except LoadError (or a subclass).
With a LoadError, if it has no further possibility to satisfy the

require,
the original exception is re-raised. Thus, polyglot should not impede
the implementation you propose.

Clifford Heath.

#3 - 06/07/2011 05:23 PM - rkh (Konstantin Haase)

How will that work with require? Remember it will only load the file once. Return false otherwise? (Which would be kinda compatible with the current behavior and using raise/throw). Or should those values be cached? If you want to use require CommonJS-style, it has to be cached. But what about return values that depend on or provoke side effects? Should files support early return?

Konstantin

On Jun 7, 2011, at 00:08 , Clifford Heath wrote:

> On 07/06/2011, at 12:18 AM, Michael Edgar wrote:
>
> > On Jun 6, 2011, at 10:11 AM, Rodrigo Rosenfeld Rosas wrote:
> >
> > > It would be great to allow return from a required file or some
> > >
> > > other keyword (break, etc or a new one)
> >
> > This could be implemented as a method that raises an exception that
> >
> > require and load rescue.
> > I'm not sure how compatible that would be with custom require
> >
> > implementations (rubygems,
> > polyglot, etc)
>
> Polyglot will pass all exceptions except LoadError (or a subclass).
> With a LoadError, if it has no further possibility to satisfy the

require,
the original exception is re-raised. Thus, polyglot should not impede
the implementation you propose.

Clifford Heath.

#4 - 06/07/2011 07:23 PM - mame (Yusuke Endoh)

Hello,

2011/6/6 Rodrigo Rosenfeld Rosas rr.rosas@gmail.com:

I have a situation where there is some code in Gitorious like:

unless defined? GitoriousConfig
Â # tons of lines here
end

And I would it to look like

return if defined? GitoriousConfig

#tons of lines here

It would be great to allow return from a required file or some other keyword (break, etc or a new one)

Agreed. It would be also useful to write platform-specific code:

require "test/unit"

return unless /mswin|cygwin|mingw|bccwin/

#5 - 06/07/2011 08:23 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Em 07-06-2011 07:10, Yusuke ENDOH escreveu:

Hello,

...

Agreed. It would be also useful to write platform-specific code:

```
require "test/unit"

return unless /mswin|cygwin|mingw|bccwin/ =~ RUBY_PLATFORM

class TestForWindowsEnv<  Test::Unit::TestCase
...
```

Here is an experimental patch:

```
diff --git a/compile.c b/compile.c
index 10d63bc..7b9c490 100644
--- a/compile.c
+++ b/compile.c
@@ -4291,10 +4291,6 @@ iseq_compile_each(rb_iseq_t *iseq, LINK_ANCHOR
*ret, NODE * node, int poped)
     rb_iseq_t *is = iseq;

     if (is) {
-        if (is->type == ISEQ_TYPE_TOP) {
-        COMPILE_ERROR((ERROR_ARGS "Invalid return"));
-        }
-        else {
         LABEL *splabel = 0;

         if (is->type == ISEQ_TYPE_METHOD) {
@@ -4321,7 +4317,6 @@ iseq_compile_each(rb_iseq_t *iseq, LINK_ANCHOR
*ret, NODE * node, int poped)
             ADD_INSN(ret, nd_line(node), pop);
             }
         }
```

```
-         }
      }
      break;
         }
diff --git a/vm_insnhelper.c b/vm_insnhelper.c
index f40dfdf..274f45d 100644
--- a/vm_insnhelper.c
+++ b/vm_insnhelper.c
@@ -1561,8 +1561,6 @@ vm_throw(rb_thread_t *th, rb_control_frame_t *reg_cfp,
              cfp = RUBY_VM_PREVIOUS_CONTROL_FRAME(cfp);
          }

-         rb_vm_localjump_error("unexpected return", throwobj, TAG_RETURN);
-
            valid_return:
          pt = dfp;
          }
```

This patch is so small, that it seems strange that it affects only
requires... Won't it have side effects? Notice that I didn't test it yet.

I agree with you about specific-platform tests use case too.

Thanks for your interest,

Rodrigo.

#6 - 06/07/2011 08:23 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Em 07-06-2011 05:14, Haase, Konstantin escreveu:

> How will that work with require? Remember it will only load the file once. Return false otherwise? (Which would be kinda compatible with the current behavior and using raise/throw). Or should those values be cached? If you want to use require CommonJS-style, it has to be cached. But what about return values that depend on or provoke side effects? Should files support early return?

I was thinking about that yesterday, but I have no idea how this should
work. If an aborted required should return true or false. Or a value
defined by the return, like:

```
return :aborted if should_abort?
```

Or if require should accept some block as:

```
require('some/file'){|return_value| do_something_with return_value }
```

Really, I have no idea about this!

Best regards,

Rodrigo.

#7 - 06/07/2011 08:53 PM - mame (Yusuke Endoh)

Hello,

2011/6/7 Rodrigo Rosenfeld Rosas rr.rosas@gmail.com:

> This patch is so small, that it seems strange that it affects only
> requires... Won't it have side effects? Notice that I didn't test it yet.

I'm not sure.
But surprisingly, the patch passes all tests except one.

--
Yusuke Endoh mame@tsg.ne.jp

#8 - 06/08/2011 09:53 AM - matz (Yukihiro Matsumoto)

Hi,

In message "Re: [ruby-core:36811] Re: [Ruby 1.9 - Feature #4840][Open] Allow returning from require"
on Tue, 7 Jun 2011 19:10:15 +0900, Yusuke ENDOH mame@tsg.ne.jp writes:

> Agreed.

Ah, I understand the request. But returning from outside of a method makes me so weird.

```
                          matz.
```

#9 - 06/08/2011 04:53 PM - headius (Charles Nutter)

On Tue, Jun 7, 2011 at 7:33 PM, Yukihiro Matsumoto matz@ruby-lang.org wrote:

> Ah, I understand the request. But returning from outside of a method
> makes me so weird.

I agree with both the feature and the fact that return outside a method feels weird. An early termination of a loading file would be welcome; I've wanted this many times, and always ended up doing the super-gross "giant if" to accomplish what could be done by a simple early exit.

I wonder if a core method that does the early return would be a better option, like Kernel#exit_script. It could be implemented to throw an exception all requires and loads expect to catch, like "ExitScriptError" or something. That would seem more consistent than having return end the script...but not actually be returning anything.

Another option would be to use a different keyword that isn't so tied to method/proc bodies, like "break"

```
break if defined? GitoriousConfig
```

I think I like the exit_script version better, though.

```
exit_script if defined? GitoriousConfig
```

- Charlie

#10 - 06/08/2011 05:53 PM - aprescott (Adam Prescott)

On Wed, Jun 8, 2011 at 8:38 AM, Charles Oliver Nutter headius@headius.comwrote:

> exit_script if defined? GitoriousConfig

This could be confusing, if you happen to view the entire program as a script. You might think that "exit_script" will do the same thing as "abort". I like the idea, though.

#11 - 06/08/2011 06:23 PM - zenspider (Ryan Davis)

On Jun 7, 2011, at 17:33 , Yukihiro Matsumoto wrote:

> Hi,
>
> In message "Re: [ruby-core:36811] Re: [Ruby 1.9 - Feature #4840][Open] Allow returning from require"
> on Tue, 7 Jun 2011 19:10:15 +0900, Yusuke ENDOH mame@tsg.ne.jp writes:
>
> |Agreed.
>
> Ah, I understand the request. But returning from outside of a method
> makes me so weird.

How about raising a specific exception that is rescued by #require instead?

```
class AbortRequire < StandardError; end

alias :original_require :require

def require f
  original_require f
rescue AbortRequire
```

```
  false
end

def

p require 'f' #
```

#12 - 06/08/2011 06:29 PM - headius (Charles Nutter)

On Wed, Jun 8, 2011 at 4:22 AM, Ryan Davis ryand-ruby@zenspider.com wrote:

> How about raising a specific exception that is rescued by #require instead?
>
> ```
> class AbortRequire < StandardError; end
>  ...
> raise AbortRequire if defined? GitoriousConfig
> ```

That's pretty clean too. Name needs work ;)

I almost suggested a special catch tag that all requires wrap, but catch/throw is a little arcane for most folks...

- Charlie

#13 - 06/08/2011 08:59 PM - regularfry (Alex Young)

On 08/06/11 01:33, Yukihiro Matsumoto wrote:

> Hi,
>
> In message "Re: [ruby-core:36811] Re: [Ruby 1.9 - Feature #4840][Open] Allow returning from require"
> on Tue, 7 Jun 2011 19:10:15 +0900, Yusuke ENDOH mame@tsg.ne.jp writes:
>
> > Agreed.
>
> Ah, I understand the request. But returning from outside of a method
> makes me so weird.

To me, return would imply that the returned value should be passed back by the original require call, like so:

```
$ cat a.rb
return 42

$ cat b.rb
p require("a")

$ ruby -I. b.rb
42
```

That could be *really* handy, although it's not compatible with my previous suggestion (currently languishing here: http://redmine.ruby-lang.org/issues/4523 - any comments?). An already-loaded file can still be signaled by a nil return value, and a file could pretend to be already loaded (if that's at all useful) by choosing nil as its return value...

Food for thought :-)

--
Alex

#14 - 06/08/2011 09:23 PM - rkh (Konstantin Haase)

One real use case I see would be avoiding global state (like CommonJS). However, if an already required file return nil, this is not possible.

#15 - 03/25/2012 04:10 PM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

#16 - 03/26/2012 12:07 AM - trans (Thomas Sawyer)

I've had occasion to use this as well, especially for RUBY_VERSION specific code.

I wonder if it is okay to be embedded in other code though. Would this work?

```
class Q
  def foo
    exit_script
  end
end

Q.new.foo
```

Is it a good idea for it to work? Or should exit_script only be allowed at toplevel?

#17 - 06/08/2012 02:38 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

- File feature-4840.odp added

#18 - 06/08/2012 08:00 AM - mame (Yusuke Endoh)

Received. Thanks for quick action!

But, matz said "returning from outside of a method makes me so weird" once.
I'm ok if you want to give it a second try with no change, but I guess matz is not likely to accept it for the same reason.
(Well, but, maybe it isn't so bad way because he sometimes changes his mind)

--
Yusuke Endoh mame@tsg.ne.jp

#19 - 06/08/2012 09:51 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

thanks for worrying, but I'll take the risk :)

#20 - 07/23/2012 10:12 PM - mame (Yusuke Endoh)

Rodrigo Rosenfeld Rosas,

At the developer meeting (7/21), Matz was basically positive
to this proposal, but it turned out that we still need to
discuss corner cases.

Here is a discussion summary.

- Matz said that he will accept "return from toplevel", but that reject "return from class definition".

```
return if cond # OK
class Foo
  return if cond # NG
end
```

- "return from toplevel" should always discard the argument.
- What's happen if it returns from the start script (the given
  script to interpreter as a cmdline)?
  - The argument should be discarded; it does NOT affect the process termination status code
  - The same as "exit", but cannot rescue SystemExit
- What's happen in the following corner cases?
  - eval("return") in toplevel
  - proc { return }.call in toplevel
- Matz prefered "return" to "a special exception that require
  and load rescue",
  - though some people (including ko1) prefered the latter.

--
Yusuke Endoh mame@tsg.ne.jp

#21 - 07/23/2012 11:59 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Em 23-07-2012 10:12, mame (Yusuke Endoh) escreveu:

> Issue #4840 has been updated by mame (Yusuke Endoh).
>
> Rodrigo Rosenfeld Rosas,
>
> At the developer meeting (7/21), Matz was basically positive
> to this proposal, but it turned out that we still need to
> discuss corner cases.
>
> Here is a discussion summary.
>
> - Matz said that he will accept "return from toplevel", but that reject "return from class definition".

```
return if cond # OK
class Foo
  return if cond # NG
end
```

Totally agree.

> - "return from toplevel" should always discard the argument.

You mean "return something", right? I agree. Maybe we could even issue an exception if anything other than just "return" is used.

> - What's happen if it returns from the start script (the given
>   script to interpreter as a cmdline)?
>
>   - The argument should be discarded; it does NOT affect the process termination status code
>   - The same as "exit", but cannot rescue SystemExit

Agreed. Specifically "return" should be equivalent to "exit 0", right? And "return -1" (or any other value, including 0) shouldn't be allowed in my opinion, raising an exception, in which case the return value wouldn't be 0 obviously :)

> - What's happen in the following corner cases?
>
>   - eval("return") in toplevel

I'd vote for just abandoning any subsequent code in the eval and nothing else. Maybe for that case "return" could use the arguments for the result of "eval".

> - proc { return }.call in toplevel

If it is the main program, "exit" would be the equivalent. Otherwise, no code would be interpreted after the "call" call.

I don't understand what makes this so special.

> - Matz prefered "return" to "a special exception that require
>   and load rescue",
>
>   - though some people (including ko1) prefered the latter.

I'm okay with raising an exception that would be rescued by require, require_relative and load.

Actually I guess this is the simpler approach and it fulfills all the real use cases I could think of in this moment.

#22 - 07/24/2012 12:44 AM - alexeymuranov (Alexey Muranov)

How about redefining __END__ to allow to call it as a method?

To avoid ambiguity with DATA constant, maybe __START_DATA__ keyword can be added?

#23 - 07/24/2012 10:29 AM - ko1 (Koichi Sasada)

(2012/07/24 0:44), alexeymuranov (Alexey Muranov) wrote:

> How about redefining __END__ to allow to call it as a method?

It has compatibility issue that __END__ is related to DATA.

--
// SASADA Koichi at atdot dot net

#24 - 07/24/2012 10:53 AM - ko1 (Koichi Sasada)

(2012/07/23 23:57), Rodrigo Rosenfeld Rosas wrote:

> - "return from toplevel" should always discard the argument.
>
> You mean "return something", right? I agree. Maybe we could even issue an exception if anything other than just "return" is used.
>
> - What's happen if it returns from the start script (the given script to interpreter as a cmdline)?
>   - The argument should be discarded; it does NOT affect the process termination status code
>   - The same as "exit", but cannot rescue SystemExit
>
> Agreed. Specifically "return" should be equivalent to "exit 0", right? And "return -1" (or any other value, including 0) shouldn't be allowed in my opinion, raising an exception, in which case the return value wouldn't be 0 obviously :)

matz proposed that ignore return argument completely. matz also said to avoid mistake, return expression with argument (example: "return foo") should be syntax error.

> - proc { return }.call in toplevel
>
> If it is the main program, "exit" would be the equivalent. Otherwise, no code would be interpreted after the "call" call.
>
> I don't understand what makes this so special.

(1)

```
pr = proc{return}
def foo(pr)
  pr.call # what happen?
end
```

(1')

```
1.times{
  return
}
```

(2)

```
# a.rb
$pr = proc{return}
```

```
# b.rb
require './a.rb'
$pr.call # waht happen?
```

(3)

```
begin
```

```
    ...
  rescue
    return
  ensure
    return
  end
```

matz proposed that "accept return on toplevel (not in block, classes, etc)".

- Matz prefered "return" to "a special exception that require and load rescue",
  - though some people (including ko1) prefered the latter.

> I'm okay with raising an exception that would be rescued by require, require_relative and load.
>
> Actually I guess this is the simpler approach and it fulfills all the real use cases I could think of in this moment.

I strongly recommended the exception approach because there are *NO* difficult corner cases.

However, matz disagreed the exception approach because of "raise StopLoading is too long". It is a kind of "name is not good".

--
// SASADA Koichi at atdot dot net

#25 - 07/24/2012 11:53 AM - trans (Thomas Sawyer)

> How about redefining __END__ to allow to call it as a method?

> It has compatibility issue that __END__ is related to DATA.

Then what about __end__?

Tangentially, why not deprecate __END__? Is there some really important use case that we just can't live without? The whole idea strikes me as rather hackish, especially considering it is limited to main file.

#26 - 07/24/2012 11:53 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Em 23-07-2012 22:37, SASADA Koichi escreveu:

> (2012/07/23 23:57), Rodrigo Rosenfeld Rosas wrote:
>
> - "return from toplevel" should always discard the argument.
>   You mean "return something", right? I agree. Maybe we could even issue an exception if anything other than just "return" is used.
>
> - What's happen if it returns from the start script (the given script to interpreter as a cmdline)?
>   - The argument should be discarded; it does NOT affect the process termination status code
>   - The same as "exit", but cannot rescue SystemExit Agreed. Specifically "return" should be equivalent to "exit 0", right? And "return -1" (or any other value, including 0) shouldn't be allowed in my opinion, raising an exception, in which case the return value wouldn't be 0 obviously :) matz proposed that ignore return argument completely. matz also said to avoid mistake, return expression with argument (example: "return foo") should be syntax error.

Better yet.

```
- proc { return }.call in toplevel
```

> If it is the main program, "exit" would be the equivalent. Otherwise, no

code would be interpreted after the "call" call.

I don't understand what makes this so special.
(1)

```
pr = proc{return}
def foo(pr)
  pr.call # what happen?
end
```

I wonder why someone would write code like this in the first place, but if someone did he is probably expecting the program to terminate with exit if this is the main file. For required files I have no idea what someone would expect from code like this. I'd probably raise an exception in such situation because it is most likely a logic error that could be hard to debug... I know what you mean, should that method "foo" be defined or not? What should be its definition? I have no idea because code like this simply doesn't make any sense to me.

(1')

```
1.times{
  return
}
```

This one I can think of. But something like:

```
['a', 'b'].each {|item|
   return if defined? item2module(item)
}
```

This would simply return from the require. I still don't understand why this would be so special.

(2)

```
# a.rb
$pr = proc{return}
```

Wow! I wouldn't ever think in something like this! Congratulations, you're really creative!

Again, why the hell would someone do something like this? I'd just raise an error when trying to call $pr because return has no more meaning after that file has been already required.

```
# b.rb
require './a.rb'
$pr.call # waht happen?
```

(3)

```
begin
  ...
rescue
  return
ensure
  return
end
```

matz proposed that "accept return on toplevel (not in block, classes, etc)".

I agree, this is way more simple to deal with. Even though I can think about someone using an approach like above, he could also rewrite it like:

```
# ['a', 'b'].each {|item|
#  return if defined? item2module(item)
#}
```

```
return if ['a', 'b'].any?{|item|  defined? item2module(item) }
```

- Matz prefered "return" to "a special exception that require and load rescue",
  - though some people (including ko1) prefered the latter. I'm okay with raising an exception that would be rescued by require, require_relative and load.

> Actually I guess this is the simpler approach and it fulfills all the real use cases I could think of in this moment.
> I strongly recommended the exception approach because there are *NO* difficult corner cases.

Agreed.

> However, matz disagreed the exception approach because of "raise StopLoading is too long". It is a kind of "name is not good".

So it would be just a matter of finding a better name, right? Not that I think it should make any difference because ideally only Ruby internals should see such errors in my opinion.

But if he thinks StopLoading is too long (while I find it short) it will be hard to find a shorter meaningful name I guess.

Maybe after a good night of sleep, who knows...

#27 - 07/24/2012 11:53 AM - shyouhei (Shyouhei Urabe)

On 2012□07□24□ 11:42, Trans wrote:

> > How about redefining __END__ to allow to call it as a method?
>
> It has compatibility issue that __END__ is related to DATA.

> Then what about __end__?
>
> Tangentially, why not deprecate __END__? Is there some really important use case that we just can't live without? The whole idea strikes me as rather hackish, especially considering it is limited to main file.

I'd be much appreciated if __END__ was usable from a library file.

I'd write a tiny loader script and place obfuscated script body on __END__ then.

#28 - 07/25/2012 02:53 AM - drbrain (Eric Hodel)

On Jul 23, 2012, at 7:42 PM, Trans transfire@gmail.com wrote:

> why not deprecate __END__? Is there some really important use case that we just can't live without? The whole idea strikes me as rather hackish, especially considering it is limited to main file.

I infrequently use __END__ for single-file scripts. It's easier to transport a single file with embedded data than two or more files.

#29 - 11/24/2012 01:39 PM - mame (Yusuke Endoh)

- Target version set to 2.0.0

#30 - 11/24/2012 03:26 PM - mame (Yusuke Endoh)

Anyone create a patch conformed to the spec written in [ruby-core:46648]?

I guess that my experimental patch ([ruby-core:36811]) is not confirmed completely;
perhaps it allows "return from class definition" (but I didn't tested yet).

--

Yusuke Endoh mame@tsg.ne.jp

#31 - 02/14/2013 06:59 PM - yhara (Yutaka HARA)

- Status changed from Assigned to Feedback

- Assignee changed from matz (Yukihiro Matsumoto) to mame (Yusuke Endoh)

- Target version changed from 2.0.0 to 2.6

#32 - 05/09/2013 11:14 AM - nobu (Nobuyoshi Nakada)

- File 0001-compile.c-toplevel-return.patch added

A simple patch.

#33 - 05/09/2013 09:41 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Pretty simple, indeed :) Not that I understand it, just that I appreciate its simplicity ;)

#34 - 05/09/2013 10:23 PM - phluid61 (Matthew Kerwin)

On May 9, 2013 12:14 PM, "nobu (Nobuyoshi Nakada)" nobu@ruby-lang.org
wrote:

> Issue #4840 has been updated by nobu (Nobuyoshi Nakada).
>
> File 0001-compile.c-toplevel-return.patch added
>
> A simple patch.

Line 369 has a typo in "rescue"

#35 - 05/31/2014 03:27 PM - nobu (Nobuyoshi Nakada)

- Description updated

#36 - 06/26/2014 12:33 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Yusuke, would the patch proposed by Nobu with the typo fix for rescue be good enough?

#37 - 06/26/2014 01:04 PM - mame (Yusuke Endoh)

- Assignee changed from mame (Yusuke Endoh) to nobu (Nobuyoshi Nakada)

I trust nobu more than myself. Thank you!

--
Yusuke Endoh mame@ruby-lang.org

#38 - 11/09/2015 06:34 AM - ko1 (Koichi Sasada)

Discussion: https://docs.google.com/document/d/1D0Eo5N7NE_unlySOKG9lVj_eyXf66BQPM4PKp7NvMyQ/pub

Feel free to continue discussion on this ticket.

#39 - 11/09/2015 10:54 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Thanks for the update, but I'd like to point out the description in that document is not accurate:

```
### Before
unless cond
  class C
  end
end
# … or …
class C
end unless cond
```

This is like the actual code looks like (the one that motivated this feature request):

https://github.com/grjones/gitorious-submodule-dependencies/blob/master/config/initializers/gitorious_config.rb

```
unless defined? GitoriousConfig
```

```
  GitoriousConfig = c = YAML::load_file(File.join(Rails.root,"config/gitorious.yml"))[RAILS_ENV]

  # make the default be publicly open
  GitoriousConfig["public_mode"] = true if GitoriousConfig["public_mode"].nil?

  # ...
```

As you can see, the assumption that we would be dealing with a conditional single class declaration is false, so, the alternative proposal for currently handling this situation does not really apply in several cases.

#40 - 08/02/2016 02:45 AM - nobu (Nobuyoshi Nakada)

Stalling due to unresolved bugs.

Current status: https://gist.github.com/nobu/e70b0c897b12b936e063

#41 - 08/03/2016 07:30 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

I'm not sure I understand that link. What is 1, 2, 3 and 4? What are the bugs?

#42 - 08/10/2016 02:51 AM - shyouhei (Shyouhei Urabe)

We looked at this issue at yesterday's developer meeting. No problem to introduce this feature was reported, except it is not implemented yet. Nobu tried this before and had technical difficulty; not sure if that could be rerouted. We are basically waiting for him (&others) to complete.

#43 - 12/21/2016 01:58 AM - nobu (Nobuyoshi Nakada)

- Status changed from Feedback to Closed

Applied in changeset r57132.

---

compile.c: toplevel return

- compile.c (iseq_compile_each): stop execution of the current source by toplevel return. [ruby-core:36785] [Feature #4840]

#44 - 07/20/2017 03:30 AM - nobu (Nobuyoshi Nakada)

- Related to Bug #13678: toplevel return causes [BUG] added

#45 - 07/20/2017 03:30 AM - nobu (Nobuyoshi Nakada)

- Related to Bug #13682: return inside of ensure causes [BUG] added

#46 - 07/20/2017 03:31 AM - nobu (Nobuyoshi Nakada)

- Related to Bug #13755: Null pointer dereference in hash_table_index() added

#47 - 08/28/2017 03:49 PM - headius (Charles Nutter)

Were any tests or specs added for this feature? I don't see anything recent added to language/return_spec.rb and the related commits don't show any additions to MRI's tests.

I believe we need at least one of these to be filled out. I'd be happy to do it but this is a very long thread and I'm not sure what is or is not considered spec.

#48 - 08/28/2017 04:49 PM - nobu (Nobuyoshi Nakada)

test/ruby/test_syntax.rb:test_return_toplevel

#49 - 08/28/2017 07:36 PM - headius (Charles Nutter)

Ahh great, thank you for the pointer, nobu!

#50 - 12/15/2017 02:21 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Could someone please explain me what is the current status of this feature? I thought I would be able to return from the top-level when requiring a file, but it still seems to be an invalid syntax to this day. Am I missing something?

#51 - 12/15/2017 03:00 PM - mame (Yusuke Endoh)

It works for me. How did you test?

```
$ cat t.rb
p 1
```

```
return
p 2
$ ruby -v t.rb
ruby 2.4.2p198 (2017-09-14 revision 59899) [x86_64-linux]
t.rb:3: warning: statement not reached
1
$ /tmp/local/bin/ruby -v t.rb
ruby 2.5.0rc1 (2017-12-14 trunk 61243) [x86_64-linux]
t.rb:3: warning: statement not reached
1
```

#52 - 02/20/2018 06:59 AM - matz (Yukihiro Matsumoto)

- Related to Bug #14062: Top-level return allows an argument added

Files

feature-4840.odp	114 KB	06/08/2012	rosenfeld (Rodrigo Rosenfeld Rosas)
0001-compile.c-toplevel-return.patch	1.71 KB	05/09/2013	nobu (Nobuyoshi Nakada)