Ruby master - Feature #12906

do/end blocks work with ensure/rescue/else

11/07/2016 01:16 AM - josh.cheek (Josh Cheek)

Status:	Closed	
Priority:	Normal	
Assignee:	nobu (Nobuyoshi Nakada)	
Target version:	2.5	

Description

When you want to rescue in a block, you must do this:

```
lambda do
  begin
    raise 'err'
  rescue
    $! # => #<RuntimeError: err>
  end
end.call
```

I've wished on numerous occasions that I could omit the begin/end and not need the extra wrapper:

```
lambda do
  raise 'err'
rescue
  $! # => #<RuntimeError: err>
end.call
```

This would be consistent with how classes and methods work:

```
class C
  raise 'err'
rescue
  $! # => #<RuntimeError: err>
end

send def m
  raise 'err'
rescue
  $! # => #<RuntimeError: err>
end
```

It's not really clear to me how to submit this since it may require some discussion, but this is the diff:

```
diff --git a/parse.y b/parse.y
index 54ccc52..223e5d3 100644
--- a/parse.y
+++ b/parse.y
@@ -3757,7 +3757,7 @@ brace_body    : {$<vars>$ = dyna_push();}

 do_body    : {$<vars>$ = dyna_push();}
           {$<val>$ = cmdarg_stack >> 1; CMDARG_SET(0);}
-          opt_block_param compstmt
+          opt_block_param bodystmt
             {
             $$ = new_do_body($3, $4);
             dyna_pop($<vars>1);
```

I added tests for ensure to rubyspec, but there wasn't an obvious place to talk about rescue/else in this context (the spec for rescue only uses it in a begin/end block) It's probably fine as the spec for ensure does hit rescue, too, and they ultimately delegate to the same pieces. Not totally clear, though. I can do more with that if you need.

```
diff --git a/language/ensure_spec.rb b/language/ensure_spec.rb
index 13575fc..b14b0b5 100644
```

```
--- a/language/ensure_spec.rb
+++ b/language/ensure_spec.rb
@@ -124,3 +124,74 @@ describe "An ensure block inside a method" do
     @obj.explicit_return_in_method_with_ensure.should == :ensure
   end
 end
+
+describe "An ensure block inside a do block" do
+  before :each do
+    ScratchPad.record []
+  end
+
+  it "is executed when an exception is raised in it's corresponding do block" do
+    begin
+      lambda do
+        ScratchPad << :begin
+        raise "An exception occured!"
+      ensure
+        ScratchPad << :ensure
+      end.should raise_error(RuntimeError)
+
+      ScratchPad.recorded.should == [:begin, :ensure]
+    end
+  end
+
+  it "is executed when an exception is raised and rescued in it's corresponding do block" do
+    begin
+      lambda do
+        ScratchPad << :begin
+        raise "An exception occured!"
+      rescue
+        ScratchPad << :rescue
+      ensure
+        ScratchPad << :ensure
+      end.call
+
+      ScratchPad.recorded.should == [:begin, :rescue, :ensure]
+    end
+  end
+
+  it "is executed even when a symbol is thrown in it's corresponding do block" do
+    begin
+      catch(:symbol) do
+        lambda do
+          ScratchPad << :begin
+          throw(:symbol)
+        rescue
+          ScratchPad << :rescue
+        ensure
+          ScratchPad << :ensure
+        end.call
+      end
+
+      ScratchPad.recorded.should == [:begin, :ensure]
+    end
+  end
+
+  it "is executed when nothing is raised or thrown in it's corresponding do block" do
+    lambda do
+      ScratchPad << :begin
+    rescue
+      ScratchPad << :rescue
+    ensure
+      ScratchPad << :ensure
+    end.call
+
+    ScratchPad.recorded.should == [:begin, :ensure]
```

```
+  end
+
+  it "has no return value" do
+    lambda do
+      :begin
+    ensure
+      :ensure
+    end.call.should == :begin
+  end
+end
```

Related issues:

Is duplicate of Ruby master - Feature #7882: Allow rescue/else/ensure in do..end	**Closed**
Is duplicate of Ruby master - Feature #11337: Allow rescue without begin insi...	**Closed**
Is duplicate of CommonRuby - Feature #12623: rescue in blocks without begin/end	**Closed**
Is duplicate of Ruby master - Feature #13212: Syntax proposal: don't require ...	**Closed**

Associated revisions

Revision 57376 - 01/19/2017 09:54 AM - nobu (Nobuyoshi Nakada)

parse.y: rescue/else/ensure in do-end

- parse.y (do_body): allow rescue/else/ensure inside do/end blocks. [Feature #12906]

Revision 57376 - 01/19/2017 09:54 AM - nobu (Nobuyoshi Nakada)

parse.y: rescue/else/ensure in do-end

- parse.y (do_body): allow rescue/else/ensure inside do/end blocks. [Feature #12906]

Revision 57376 - 01/19/2017 09:54 AM - nobu (Nobuyoshi Nakada)

parse.y: rescue/else/ensure in do-end

- parse.y (do_body): allow rescue/else/ensure inside do/end blocks. [Feature #12906]

History

#1 - 11/07/2016 02:28 AM - shyouhei (Shyouhei Urabe)

- Is duplicate of Feature #7882: Allow rescue/else/ensure in do..end added

#2 - 11/07/2016 02:29 AM - shyouhei (Shyouhei Urabe)

- Is duplicate of Feature #11337: Allow rescue without begin inside blocks added

#3 - 11/07/2016 02:29 AM - shyouhei (Shyouhei Urabe)

- Is duplicate of Feature #12623: rescue in blocks without begin/end added

#4 - 11/07/2016 02:52 AM - shyouhei (Shyouhei Urabe)

- Assignee set to matz (Yukihiro Matsumoto)

- Status changed from Open to Assigned

Josh Cheek wrote:

> but this is the diff:
>
> ```
> diff --git a/parse.y b/parse.y
> index 54ccc52..223e5d3 100644
> --- a/parse.y
> +++ b/parse.y
> @@ -3757,7 +3757,7 @@ brace_body  : {$<vars>$ = dyna_push();}
>
>  do_body  : {$<vars>$ = dyna_push();}
>         {$<val>$ = cmdarg_stack >> 1; CMDARG_SET(0);}
> -       opt_block_param compstmt
> +       opt_block_param bodystmt
>           {
>           $$ = new_do_body($3, $4);
>           dyna_pop($<vars>1);
> ```

So from the patch you sent, I guess you are implicitly proposing to forget about {...}-style blocks for a while and focus on do ... end-style blocks. That is in fact a wise idea. To start small is a wisdom we learned in this forum.

#5 - 11/07/2016 03:19 AM - josh.cheek (Josh Cheek)

My error on the duplication, I tried searching a couple different ways but the first results weren't about this and I didn't see how to use the filters and the keywords together. Fortunately it seems congruent with https://bugs.ruby-lang.org/issues/7882

Shyouhei Urabe wrote:

> So from the patch you sent, I guess you are implicitly proposing to forget about {...}-style blocks for a while and focus on do ... end-style blocks. That is in fact a wise idea. To start small is a wisdom we learned in this forum.

Ty :) the existing places it works are begin/end, def/end, class/end, so it felt unifying to add do/end, but seemed less natural with curly braces. I'm sure I'd get used to it, if it were there, though.

#6 - 11/08/2016 02:57 PM - shevegen (Robert A. Heiler)

I like the idea. It would get rid of one begin/end clause if I understood
it correctly and be on the same level as class C/rescue/end definitions.

#7 - 11/30/2016 03:58 PM - josh.cheek (Josh Cheek)

Checking that I submitted this correctly, I usually do Github, so might have gotten this wrong.

#8 - 01/19/2017 08:27 AM - matz (Yukihiro Matsumoto)

- Target version set to 2.5

- Assignee changed from matz (Yukihiro Matsumoto) to nobu (Nobuyoshi Nakada)

Although I am not a big fan of this syntax, mostly because I don't like fine grain exception handling.
But I found out many developers prefer the syntax. After some consideration, I decided to accept this.

Matz.

#9 - 01/19/2017 09:55 AM - nobu (Nobuyoshi Nakada)

- Status changed from Assigned to Closed

Applied in changeset r57376.

---

parse.y: rescue/else/ensure in do-end

- parse.y (do_body): allow rescue/else/ensure inside do/end blocks. [Feature #12906]

#10 - 02/14/2017 06:50 AM - hsbt (Hiroshi SHIBATA)

- Is duplicate of Feature #13212: Syntax proposal: don't require begin-end to rescue exceptions inside do-end blocks added

#11 - 05/28/2018 12:00 PM - perlun (Per Lundberg)

Old issue, but still perhaps the right place to mention this: the new syntax ONLY works in do/end, not in {} blocks (as mentioned above.)

It also does not work in "block-like" places like a for loop. So this is not valid syntax:

```
for i in 1..100
  puts 'hello'
rescue # Syntax error; not valid code in existing Ruby versions like 2.5.0 and 2.5.1.
end
```

Suggested workaround: Use a (1..100).each do |i| construct instead. Then you can use a rescue or ensure within the block in Ruby 2.5 and newer.

(This is actually an arguably valid use case for "fine grained error handling", in cases where the loop can fail in n of the 100 cases and you want the failing ones to be ignored. I agree that too-finegrained error handling can otherwise become an antipattern.)