### Ruby master - Bug #9907

**Abbreviated method assignment with private attr_writer/attr_reader does not work.**

06/05/2014 03:50 PM - jwmittag (Jörg W Mittag)

<table>
<thead>
<tr>
<th>Status</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee</td>
<td></td>
</tr>
<tr>
<td>Target version</td>
<td>2.2.0</td>
</tr>
<tr>
<td>ruby -v</td>
<td>ruby 2.2.0dev (2014-06-05 trunk 46357) [x86_64-darwin13]</td>
</tr>
<tr>
<td>Backport</td>
<td>2.0.0: REQUIRED, 2.1: DONTNEED</td>
</tr>
</tbody>
</table>

**Description**

This looks like a hole in the specification:

```ruby
private def foo=(*) end
public def foo; 0 end

self.foo = 42

self.foo += 42
# private method `foo=' called for main:Object (NoMethodError)

private :foo

self.foo += 42
# private method `foo' called for main:Object (NoMethodError)
```

There is an exception for private writers in the rule for private message sends, but apparently that exception needs to broadened so that it also works in the case of abbreviated assignments. I'm not entirely sure what this rule would be, but I don't think it would break backwards compatibility, since all situations that would work differently with the changed rule would currently raise a NoMethodError anyway.

The rule should be something like:

- private methods can only be called without an explicit receiver.
- An exception is made for method assignments, where the literal receiver self is also allowed in the assignee method expression.
- This also applies to compound assignments: self.foo ω= bar shall always succeed if either or both of foo and foo= are private.

**Related issues:**

- Related to Ruby master - Bug #10060: private attr_accessor and NoMethodError
- Related to Ruby master - Bug #11096: 'private' access control bypassed when |...

**Associated revisions**

Revision 199f814f - 06/06/2014 08:07 AM - nobu (Nobuyoshi Nakada)

compile.c, parse.y: private op assign

- compile.c (iseq_compile_each), parse.y (new_attr_op_assign_gen): allow op assign to a private attribute. [ruby-core:62949] [Bug #9907]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@46365 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 46365 - 06/06/2014 08:07 AM - nobu (Nobuyoshi Nakada)

compile.c, parse.y: private op assign

- compile.c (iseq_compile_each), parse.y (new_attr_op_assign_gen): allow op assign to a private attribute. [ruby-core:62949] [Bug #9907]

Revision 46365 - 06/06/2014 08:07 AM - nobu (Nobuyoshi Nakada)

compile.c, parse.y: private op assign

03/20/2022
compile.c (iseq_compile_each), parse.y (new_attr_op_assign_gen): allow op assign to a private attribute. [ruby-core:62949] [Bug #9907]

Revision 46365 - 06/06/2014 08:07 AM - nobu (Nobuyoshi Nakada)
compile.c, parse.y: private op assign

compile.c (iseq_compile_each), parse.y (new_attr_op_assign_gen): allow op assign to a private attribute. [ruby-core:62949] [Bug #9907]

Revision 46365 - 06/06/2014 08:07 AM - nobu (Nobuyoshi Nakada)
compile.c, parse.y: private op assign

compile.c (iseq_compile_each), parse.y (new_attr_op_assign_gen): allow op assign to a private attribute. [ruby-core:62949] [Bug #9907]

Revision 46365 - 06/06/2014 08:07 AM - nobu (Nobuyoshi Nakada)
compile.c, parse.y: private op assign

compile.c (iseq_compile_each), parse.y (new_attr_op_assign_gen): allow op assign to a private attribute. [ruby-core:62949] [Bug #9907]

History

#1 - 06/06/2014 08:06 AM - nobu (Nobuyoshi Nakada)
- Backport changed from 2.0.0: UNKNOWN, 2.1: UNKNOWN to 2.0.0: REQUIRED, 2.1: REQUIRED

#2 - 06/06/2014 08:06 AM - nobu (Nobuyoshi Nakada)
- Description updated

#3 - 06/06/2014 08:07 AM - nobu (Nobuyoshi Nakada)
- Status changed from Open to Closed
- % Done changed from 0 to 100

Applied in changeset r46365.

compile.c, parse.y: private op assign

compile.c (iseq_compile_each), parse.y (new_attr_op_assign_gen): allow op assign to a private attribute. [ruby-core:62949] [Bug #9907]

#4 - 07/18/2014 05:17 PM - nobu (Nobuyoshi Nakada)
- Related to Bug #10060: private attr_accessor and NoMethodError added

#5 - 09/14/2014 02:47 PM - nagachika (Tomoyuki Chikanaga)
- Backport changed from 2.0.0: REQUIRED, 2.1: REQUIRED to 2.0.0: REQUIRED, 2.1: DONTNEED

This seems a spec change for me. Any comments?

#6 - 11/06/2014 04:41 PM - headius (Charles Nutter)
I'm confused about how private dispatch against self should behave. I expected that at least all self.x calls would be considered private dispatch, but that's not the case.

For this code:

```ruby
private
def foo; @a; end
def foo=(a); @a = a; end

puts begin; foo; rescue; 'foo failed'; else; 'foo worked'; end
puts begin; self.foo; rescue; 'self.foo failed'; else; 'self.foo worked'; end
puts begin; self.foo = 1; rescue; 'self.foo = 1 failed'; else; 'self.foo = 1 worked'; end
puts begin; self.foo += 1; rescue; 'self.foo += 1 failed'; else; 'self.foo += 1 worked'; end
puts begin; a = self; o.foo; rescue; 'o = self; o.foo failed'; else; 'o = self; o.foo worked'; end
```

03/20/2022
puts begin; o = self; o.foo = 1; rescue; 'o = self; o.foo = 1 failed'; else; 'o = self; o.foo = 1 worked'; end

I have the following behavior with 2.2.0-preview1:

```
foo worked
self.foo failed
self.foo = 1 worked
self.foo += 1 worked
o = self; o.foo failed
o = self; o.foo = 1 failed
```

On 2.1, the self.foo += 1 case failed, which is the purpose of this bug.

However, I don't see why that case should pass and self.foo should fail. self.foo += must do an implicit self.foo call, meaning the "self" exception for private dispatch propagates through. It makes no sense that directly calling self.foo would produce an error and implicitly calling self.foo would not.

I would also argue that the "o" cases should pass as well, but I'm more concerned about the inconsistent spec change in this bug.

#7 - 11/07/2014 01:22 AM - nobu (Nobuyoshi Nakada)
I think, += is a kind of assignment, so it should allow even private attributes as a whole.

#8 - 11/07/2014 06:13 PM - headius (Charles Nutter)
The following two pieces of code should both work, since they both do the same thing. VM-level exceptions hurt understandability, and it should come as a surprise to any Rubyist that self.foo+= can skip visibility checks but self.foo can't.

```
self.foo += 1 # ok
AND

self.foo = self.foo + 1 # visibility error
```

The first line essentially expands to the second line. The first line works, the second line doesn't. That makes no sense to me. Also, what if I had "self.foo += 1" (working fine after this bug) and later wanted to expand the logic to add some additional operations. It would break, and I wouldn't understand why.

```
self.foo = self.foo * 5 + 1 # visibility error
```

I think consistency wins here and self.foo should be able to skip visibility checks too.

#9 - 11/08/2014 09:31 PM - Eregon (Benoit Daloze)
Until now, the ability to call private methods has always been a property verifiable at parse time (no receiver or "receiver is self and methods ends with =").

I think it is good to remain that way so it is really easy to know whether you may call private method, as a Ruby user. So I think the "o" cases above should not be able to call private methods.

On the other hand, the current semantics seem less clear and more complex than they could be.

I agree with Charles, the "no explicit receiver or receiver is literally self" rule is simpler, more consistent and avoid problems with special cases such as assignments.

Having a hidden implicit visibility such as in the committed change seems harder for everyone to understand and may only solve a part of the special cases.

#10 - 04/27/2017 12:19 PM - similiser (Siim Liiser)
This issue seems to have resurfaced. The issue is fixed in 2.2, but broken in both 2.3 and 2.4.

#11 - 04/27/2017 03:11 PM - nobu (Nobuyoshi Nakada)
- Related to Bug #11096: 'private' access control bypassed when ||= is used added

#12 - 04/27/2017 03:12 PM - nobu (Nobuyoshi Nakada)
Intentional fix.