Ruby - Bug #17354

Module#const_source_location is misleading for constants awaiting autoload

11/29/2020 03:23 PM - tomstuart (Tom Stuart)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.7.2p137 (2020-10-01 revision 5445e04352) [x86_64-darwin20]	**Backport:**	2.5: UNKNOWN, 2.6: UNKNOWN, 2.7: UNKNOWN

Description

Feature #10771 added Module#const_source_location as a way to find the source location of a constant's definition. Bug #16764 reported that it didn't work correctly for autoloaded constants, instead giving the source location of the autoload call site. This was fixed in v3_0_0_preview1 in 92730810 and backported to v2_7_2 in c65aae11.

However, #const_source_location still returns the autoload call site for constants which have not yet been loaded:

```
% echo 'class Foo; end' > foo.rb

% irb
>> Module.const_defined?(:Foo)
=> false
>> Module.const_source_location(:Foo)
=> nil

>> autoload :Foo, './foo'
=> nil

>> Module.const_defined?(:Foo)
=> true
>> Module.const_source_location(:Foo)
=> ["(irb)", 3]

>> Module.const_get(:Foo)
=> Foo

>> Module.const_defined?(:Foo)
=> true
>> Module.const_source_location(:Foo)
=> ["./foo.rb", 1]
```

This edge case is undocumented and surprising. It looks like a bug to the programmer who receives the autoload location instead of one of the documented return values of #const_source_location (nil, [], or the definition's source location).

We could either:

- change the behaviour of #const_source_location to return [] for constants awaiting autoload, which is consistent with the return value of Module#const_defined? in this case ("if the constant is not present but there is an autoload for it, true is returned directly"), as well as the return value of #const_source_location for other constants whose source location is unknown ("if the constant is found, but its source location can not be extracted (constant is defined in C code), empty array is returned"); or
- document the current behaviour of #const_source_location to make it less surprising.

I recommend the first option — although the current behaviour was recently specified in source:spec/ruby/core/module/const_source_location_spec.rb@6d059674#L209, it doesn't seem intentional — but if that's not feasible, simply documenting this edge case would also be an improvement.

History

#1 - 11/30/2020 01:23 AM - mame (Yusuke Endoh)

Do you think what is the purpose of Module#const_source_location? Unfortunately, the original motivation is not expressed in #10771. IMO, it is for debugging. I think, when we are trying to find the definition of a constant, it is actually useful to see the line number that calls autoload, instead of seeing an empty array.

#2 - 11/30/2020 12:16 PM - ufuk (Ufuk Kayserilioglu)

@mame (Yusuke Endoh) In my use-case, I would very much like Module#const_source_location to tell me where the constant is loaded or *would be loaded from if it is autoloaded*. I am doing runtime reflection to discover types in gems and their associated methods, constants, etc in Tapioca, and my biggest problem so far has been detecting which file a constant is originally loaded from. I'd been waiting for Module#const_source_location to give me that information, even if the constant has not been loaded yet. The autoload file location, theoretically, could point to a file in a different gem or even to bootsnap if it is in play, which stops this use-case from working properly.

I would very much like Module#const_source_location to work like this:

```
% echo 'class Foo; end' > foo.rb

% irb
>> Module.const_defined?(:Foo)
=> false
>> Module.const_source_location(:Foo)
=> nil

>> autoload :Foo, './foo'
=> nil

>> Module.const_defined?(:Foo)
=> true
>> Module.const_source_location(:Foo)
=> ["./foo.rb", nil]

>> Module.const_get(:Foo)
=> Foo

>> Module.const_defined?(:Foo)
=> true
>> Module.const_source_location(:Foo)
=> ["./foo.rb", 1]
```

since Ruby basically knows where Foo *will be* loaded from, but cannot tell us what line number it will be from without actually loading the file. Thus, I propose the line number be nil in that case.

Do you think that works?

#3 - 11/30/2020 01:34 PM - tomstuart (Tom Stuart)

> Do you think what is the purpose of Module#const_source_location? Unfortunately, the original motivation is not expressed in #10771. IMO, it is for debugging. I think, when we are trying to find the definition of a constant, it is actually useful to see the line number that calls autoload, instead of seeing an empty array.

Like @ufuk (Ufuk Kayserilioglu), I'm trying to use #const_source_location to do runtime reflection in tooling, not for debugging, so the autoload call site is not useful.

#4 - 11/30/2020 04:14 PM - jeremyevans0 (Jeremy Evans)

ufuk (Ufuk Kayserilioglu) wrote in #note-2:

> @mame (Yusuke Endoh) In my use-case, I would very much like Module#const_source_location to tell me where the constant is loaded or *would be loaded from if it is autoloaded*. I am doing runtime reflection to discover types in gems and their associated methods, constants, etc in Tapioca, and my biggest problem so far has been detecting which file a constant is originally loaded from. I'd been waiting for Module#const_source_location to give me that information, even if the constant has not been loaded yet. The autoload file location, theoretically, could point to a file in a different gem or even to bootsnap if it is in play, which stops this use-case from working properly.
>
> I would very much like Module#const_source_location to work like this:
>
> ```
> % echo 'class Foo; end' > foo.rb
>
> % irb
> >> Module.const_defined?(:Foo)
> => false
> >> Module.const_source_location(:Foo)
> => nil
>
> >> autoload :Foo, './foo'
> => nil
>
> >> Module.const_defined?(:Foo)
> => true
> ```

```
>> Module.const_source_location(:Foo)
=> ["./foo.rb", nil]

>> Module.const_get(:Foo)
=> Foo

>> Module.const_defined?(:Foo)
=> true
>> Module.const_source_location(:Foo)
=> ["./foo.rb", 1]
```

since Ruby basically knows where Foo *will be* loaded from, but cannot tell us what line number it will be from without actually loading the file. Thus, I propose the line number be nil in that case.

This is not always true. Ruby does not know that the constant's location will be in that file. All it knows is that requiring that file should result in the constant being available after. The file mentioned may require or load another file with the constant definition. It may not even load it, you could end up with a NameError when referencing the constant.

If the location of the autoload definition is not useful, I think the main reasonable alternative would be [], as Ruby does not know where the constant will be defined. This is the same value used for constants defined in C.

#5 - 11/30/2020 04:28 PM - mame (Yusuke Endoh)

@ufuk (Ufuk Kayserilioglu) @tomstuart I'm unsure if I could understand your use case correctly, but maybe does Module#autoload? help you?

#6 - 11/30/2020 04:33 PM - mame (Yusuke Endoh)

An example. You can ignore the result of const_source_location if autoload? returns non-nil.

```
$ irb
irb(main):001:0> autoload(:Foo, "./foo")
=> nil
irb(main):002:0> Module.autoload?(:Foo)
=> "./foo"
irb(main):003:0> Module.const_source_location(:Foo)
=> ["(irb)", 1]
irb(main):004:0> Foo
=> Foo
irb(main):005:0> Module.autoload?(:Foo)
=> nil
irb(main):006:0> Module.const_source_location(:Foo)
=> ["/home/mame/work/ruby/foo.rb", 1]
```

#7 - 11/30/2020 06:58 PM - tenderlovemaking (Aaron Patterson)

jeremyevans0 (Jeremy Evans) wrote in #note-4:

> This is not always true. Ruby does not know that the constant's location will be in that file. All it knows is that requiring that file should result in the constant being available after. The file mentioned may require or load another file with the constant definition. It may not even load it, you could end up with a NameError when referencing the constant.
>
> If the location of the autoload definition is not useful, I think the main reasonable alternative would be [], as Ruby does not know where the constant will be defined. This is the same value used for constants defined in C.

I think asking for the const source location on something that hasn't been autoloaded yet should cause autoload to be triggered, then get the const source location. I don't think returning [] is reasonable because as you say, loading the constant could result in a NameError and the constant never being defined at all. When source location is [] we know it's for something that *is* defined, we just don't know *where* (it's implemented in C). In the autoload case it's something that is *potentially* defined, but we can't know unless the autoload is triggered.

It's weird that const_source_location would return a value for a constant that can never be defined:

```
irb(main):001:0> File.read "foo.rb"
=> "class Bar\nend\n"
irb(main):002:0> autoload(:Foo, "./foo.rb")
=> nil
irb(main):003:0> Module.const_source_location(:Foo)
=> ["(irb)", 2]
irb(main):004:0> Foo
Traceback (most recent call last):
        4: from /Users/aaron/.rubies/ruby-trunk/bin/irb:23:in `<main>'
        3: from /Users/aaron/.rubies/ruby-trunk/bin/irb:23:in `load'
        2: from /Users/aaron/.rubies/ruby-trunk/lib/ruby/gems/3.0.0/gems/irb-1.2.7/exe/irb:11:in `<top (requir
ed)>'
```

```
        1: from (irb):4
NameError (uninitialized constant Foo)
irb(main):005:0> Module.const_source_location(:Foo)
=> ["(irb)", 2]
```

#8 - 11/30/2020 07:06 PM - tenderlovemaking (Aaron Patterson)

We can probably never change this behavior due to backwards compatibility, but along the same lines, I think this behavior is weird too:

```
irb(main):001:0> File.read "foo.rb"
=> "class Bar\nend\n"
irb(main):002:0> autoload(:Foo, "./foo.rb")
=> nil
irb(main):003:0> Object.const_defined?(:Foo)
=> true
irb(main):004:0> Foo
Traceback (most recent call last):
        4: from /Users/aaron/.rubies/ruby-trunk/bin/irb:23:in `<main>'
        3: from /Users/aaron/.rubies/ruby-trunk/bin/irb:23:in `load'
        2: from /Users/aaron/.rubies/ruby-trunk/lib/ruby/gems/3.0.0/gems/irb-1.2.7/exe/irb:11:in `<top (requir
ed)>'
        1: from (irb):4
NameError (uninitialized constant Foo)
irb(main):005:0> Object.const_defined?(:Foo)
=> false
```

Foo was never defined and could never be defined, yet const_defined? returned true. It seems like const_defined? should return :not_yet or something □□

#9 - 11/30/2020 07:44 PM - ufuk (Ufuk Kayserilioglu)

mame (Yusuke Endoh) wrote in #note-6:

> An example. You can ignore the result of const_source_location if autoload? returns non-nil.
>
> ```
> $ irb
> irb(main):001:0> autoload(:Foo, "./foo")
> => nil
> irb(main):002:0> Module.autoload?(:Foo)
> => "./foo"
> irb(main):003:0> Module.const_source_location(:Foo)
> => ["(irb)", 1]
> irb(main):004:0> Foo
> => Foo
> irb(main):005:0> Module.autoload?(:Foo)
> => nil
> irb(main):006:0> Module.const_source_location(:Foo)
> => ["/home/mame/work/ruby/foo.rb", 1]
> ```

@mame (Yusuke Endoh) Thank you! That actually works for my use-case. In that case, I suggest that we leave the current implementation as is, and document the behaviour as @tomstuart had originally suggested.

#10 - 03/06/2021 02:50 AM - sawa (Tsuyoshi Sawada)

When I proposed this feature in #10771, the motivation was to create a tool that automatically reads Ruby code and documents its methods and constants together with their source code. To extract the relevant source code, knowing the start and end points of the definitions is necessary. The current feature gives the file name and the line number of the start point of the definition, lacking the character position (as well as information regarding the endpoint of the definition), which is only a part of what was necessary for me, but I still appreciate the developers for implementing the feature.

I agree with Tom that the current behaviour is wrong, and I think that returning the calling site does not make sense, nor is it useful. But I don't think [] should be returned, for the reason Aaron discusses.

Depending on whether we want to evaluate Ruby code statically or dynamically, the correct answer for the return value differs.

A. If we want a static picture, nil should be returned.
B. If we want a dynamic picture, I think we should adopt Aaron's proposal (to attempt) to load the constant on the spot.

B above seems a bit more practical than A, and seems to match my original motivation.

Still, both may make sense. We could have another option to switch the behaviour with an optional keyword argument trigger_autoload that has true for default:

```
autoload "Foo", "./foo"
```

```
Module.const_source_location("Foo", trigger_autoload: false) # => nil

Module.const_source_location("Foo") # => ["./foo.rb", 1]
```

In any case, I think that the behaviour should be in line with that of const_defined?. The latter should also be modified according to any modification being made to the current feature.

#11 - 03/15/2021 09:35 AM - mame (Yusuke Endoh)

I think that the current behavior is the most flexible.

1. If we want to check if a constant is set as autoload (and not actually loaded), we can use Module#autoload?.
2. If we want to identify where a constant is set as autoload, we can use the current Module#const_source_location.
3. If we want to make sure a constant is loaded and identify where a constant is defined, we can use Module#const_get and then Module#const_source_location.

If Module#const_source_location returns [] or nil when a constant is not actually loaded, it breaks the scenario 2. In fact I have used the method to identify the callsite of autoload for debugging.

#12 - 03/26/2021 05:56 PM - jeremyevans0 (Jeremy Evans)

This was discussed during the March 2021 developer meeting, but a conclusion was not reached:
https://github.com/ruby/dev-meeting-log/blob/master/DevelopersMeeting20210317Japan.md#bug-17354-moduleconst_source_location-is-misleading-for-constants-awaiting-autoload-jeremyevans0

#13 - 09/13/2023 04:07 AM - sawa (Tsuyoshi Sawada)

mame (Yusuke Endoh) wrote in #note-11:

> 1. If we want to identify where a constant is set as autoload, we can use the current Module#const_source_location.
>
> If Module#const_source_location returns [] or nil when a constant is not actually loaded, it breaks the scenario 2. In fact I have used the method to identify the callsite of autoload for debugging.

You cannot safely use const_source_location to extract the autoload call site since the return value depends on whether the constant is called or not:

```
autoload "A", "path_to_def_of_A"
autoload "B", "path_to_def_of_B"
p A
p Module.const_source_location("A") # => def site
p Module.const_source_location("B") # => autoload call site
```

Using const_source_location, you can know the call site of B, but not the call site of A. If you wanted to safely know the autoload call site for a constant name, you would need a different mechanism anyway.

Even worse, the return value for the same constant name may change:

```
autoload "A", File.expand_path("~/tmp/bin/a.rb")
p Module.const_source_location("A") # => autoload call site
p A
p Module.const_source_location("A") # => def site
```

Also, mame suggests to retain the current behavior of const_source_location and use it in combination with autoload? or const_get. That makes const_source_location useless by itself, and would have to be always used together with autoload? or const_get if there is a possibility of autoload and you wanted to safely know the definition site.

Hence, if there is a use case of extracting the autoload call site as mame discusses, (which I am neither for nor against), there should be a different method for doing that, and const_source_location should not return the autoload call site at all (at least by default).

#14 - 09/13/2023 05:11 AM - mame (Yusuke Endoh)

sawa (Tsuyoshi Sawada) wrote in #note-13:

> You cannot safely use const_source_location to extract the autoload call site since the return value depends on whether the constant is called or not:

As I said, you can use it safely in conjunction with Module.autoload?.

```
autoload "A", File.expand_path("./a.rb")
p Module.autoload?(:A)              #=> "/path/to/a.rb"
p Module.const_source_location(:A) #=> autoload call site
p A
```

```
p Module.autoload?(:A)              #=> nil
p Module.const_source_location(:A) #=> def site
```

As Aaron says, I think it is one reasonable design to load it immediately when calling Module.const_source_location.
But I think the current behavior is also reasonable enough. I don't know if we need to risk compatibility to change it.

#15 - 09/13/2023 05:55 AM - sawa (Tsuyoshi Sawada)

mame (Yusuke Endoh) wrote in #note-14:

> As I said, you can use it safely in conjunction with Module.autoload?.
>
> ```
> autoload "A", File.expand_path("./a.rb")
> p Module.autoload?(:A)              #=> "/path/to/a.rb"
> p Module.const_source_location(:A) #=> autoload call site
> p A
> p Module.autoload?(:A)              #=> nil
> p Module.const_source_location(:A) #=> def site
> ```

What I meant is that, once you are in a location beyond where A was called, you cannot reference the call site any more.

```
# b.rb
autoload "A", File.expand_path("./a.rb")
# short code
load "c.rb"

# c.rb
p A
# long code, or in a different file after a chain of file loading
load "d.rb"

# d.rb
# You can tell that you cannot achieve the call site from here:
p Module.autoload?(:A)              #=> nil
# And in fact, what is returned is not the call site:
p Module.const_source_location(:A) #=> def site
```

Provided you are currently working on d.rb, in order to use const_source_location to find the call site, you have to find a location where the constant is not yet called (all the way back to before the line in c.rb where A is called), and then use const_source_location there. If you are able to do that, you can probably find the call site directly without much more effort.

#16 - 09/13/2023 10:44 AM - mame (Yusuke Endoh)

Why do you want to know the location of the autoload call after it is actually loaded? I guess that information is already gone from the runtime as well.

#17 - 09/13/2023 12:38 PM - sawa (Tsuyoshi Sawada)

mame (Yusuke Endoh) wrote in #note-16:

> Why do you want to know the location of the autoload call after it is actually loaded?

I do not know. I do not understand in the first place your use case where you want to identify where a constant is set as autoload.

> I guess that information is already gone from the runtime as well.

I see.

#18 - 09/13/2023 02:30 PM - mame (Yusuke Endoh)

I said this could be useful for debugging. If a user finds a call to autoload on the line pointed to by const_source_location, they can guess that autoload is set. If they want to know the actual definition location, they can actually access it and then use const_source_location. The current behavior allows the user to choose these. In that sense, I think the current behavior is flexible. If const_source_location triggers autoload, not only is this flexibility lost, but loading may cause side effects, which may be inconvenient for a debugging purpose.

#19 - 09/14/2023 05:47 AM - matz (Yukihiro Matsumoto)

Currently, it gives the place we called autoload. If you want the place where the loaded module defined (after autoload), you can explicitly reference the module before calling const_source_location. So I vote for keeping it as it is now.

Matz.

#20 - 09/26/2023 11:07 PM - jeremyevans0 (Jeremy Evans)

- Status changed from Open to Closed