Ruby - Feature #4824

Provide method Kernel#executed?

06/04/2011 07:59 PM - lazaridis.com (Lazaridis Ilias)

Status:	Assigned	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:		

Description

The current construct to execute main code looks not very elegant:

```
if __FILE__ == $0
  my_main()           # call any method or execute any code
end
```

With a Kernel#executed? method, this would become more elegant:

```
if executed?
  #do this
  #do that
  my_main()
end
```

or

```
main() if executed?
```

This addition would not break any existent behaviour.

History

#1 - 06/04/2011 11:16 PM - matz (Yukihiro Matsumoto)

I agree providing a method to tell whether it is loaded as a library or is executed as a command is more elegant than '**FILE** == $0', but I am still afraid #executed? can mean multiple ways so that the name can cause confusion. We need a better name.

#2 - 06/04/2011 11:53 PM - Cezary (Cezary Baginski)

On Sat, Jun 04, 2011 at 11:17:01PM +0900, Yukihiro Matsumoto wrote:

> I agree providing a method to tell whether it is loaded as a library
> or is executed as a command is more elegant than '**FILE** == $0',
> but I am still afraid #executed? can mean multiple ways so that the
> name can cause confusion. We need a better name.

How about the inverse or something similar:

```
unless required?  # top-level file, not require'd or loaded
  #main
  puts "hello from main"
end
```

This would also protect from main being run twice when the file
includes itself.

--
Cezary Baginski

#3 - 06/05/2011 03:53 AM - sdsykes (Stephen Sykes)

Some other suggestions:

```
if main?
if first_script?
if main_script?
if run_script?
```

-Stephen

#4 - 06/05/2011 03:01 PM - lazaridis.com (Lazaridis Ilias)

Some notes subjecting the naming:

The clarity of the method name should be rated in it's OO context, like this:

```
if self.executed? # or .started? .launched?, which means *not* .included? .required? .loaded?
  # do main code
end
```

similarly, one could write:

```
unless self.executed?
  # do inclusion code
end
```

"self" would be "the module" or "the file" or "the main object"

Possibly the most simple word would be "main?", but strictly from a OO view, this would be not correct.

Although, from a user view, it looks simple and recognizable:

```
if main?
  do main stuff
end

unless main?
  do inclusion stuff
end
```

In any way, the name should be a compact one, e.g. without "_", like is_started?

#5 - 06/05/2011 04:29 PM - cjheath (Clifford Heath)

On 05/06/2011, at 4:01 PM, Lazaridis Ilias wrote:

> The clarity of the method name should be rated in it's OO context,
> like this:

I dispute the need for a method. That just forces someone to the
documentation
to know what the code means.

The problem with the existing solution "if (__FILE__ == $0)" is that
the meaning is
hidden. If the variables were named differently, this would be exactly
the expression
that would best communicate the intent. To exaggerate, what we need is:

```
if __THIS_FILE__ == $THIS_PROGRAM
  ...
end
```

That is, the problem is that it's not obvious to a newcomer that __FILE__ means
the current source-code file, or that $0 means the name of the script
being
executed. Especially the latter...

Clifford Heath.

#6 - 06/05/2011 10:29 PM - Eregon (Benoit Daloze)

Clifford Heath wrote:

> That is, the problem is that it's not obvious to a newcomer that **FILE** means
> the current source-code file, or that $0 means the name of the script
> being executed. Especially the latter...

$PROGRAM_NAME is an alias for $0.
But "if __FILE__ == $PROGRAM_NAME" is quite long.

#7 - 06/06/2011 06:53 AM - cjheath (Clifford Heath)

On 05/06/2011, at 11:29 PM, Benoit Daloze wrote:

> Clifford Heath wrote:
>
> > That is, the problem is that it's not obvious to a newcomer that
> > __FILE__ means
> > the current source-code file, or that $0 means the name of the script
> > being executed. Especially the latter...
>
> $PROGRAM_NAME is an alias for $0.
> But "if __FILE__ == $PROGRAM_NAME" is quite long.

Length is not a problem, if the text includes the meaning.

Things that are used often should be succinct, and the reader's knowledge should be assumed. Things that are used only a few times in a program do not need to be succinct.

To use an API call requires that the user knows (or looks up) the meaning. This kind of semantic hiding is completely unnecessary and *counter-productive*.

Clifford Heath.

#8 - 06/06/2011 02:51 PM - lazaridis.com (Lazaridis Ilias)

Clifford Heath wrote:
[...]

> But "if __FILE__ == $PROGRAM_NAME" is quite long.
> [...]
> To use an API call requires that the user knows (or looks up) the meaning. This kind of semantic hiding is completely unnecessary and *counter-productive*.
> [...]

So maybe all rarely used methods should be written in long descriptive statements, to overcome non-semantic-hiding and become more productive?

There's one simple fact:

"if __FILE__ == $PROGRAM_NAME" is inconsistent with the language's elegant OO design - and I don't think that anyone will counter this.

And at this point, the "dispute" is about the method name.

Which method name would you choose, if you had two choices (.main? | .executed?) ?

#9 - 06/06/2011 03:23 PM - cjheath (Clifford Heath)

On 06/06/2011, at 3:51 PM, Lazaridis Ilias wrote:

> So maybe all rarely used methods should be written in long descriptive statements, to overcome non-semantic-hiding and become more productive?

I thought you prided yourself on your ability to use reason? Because the above is really an irrational statement, being both a logical non-sequitur and unrelated to my argument.

> There's one simple fact:
> "if __FILE__ == $PROGRAM_NAME" is inconsistent with the language's elegant OO design

I think that's entirely false.

If I was explaining the intent of my code to you, I'd say "If we're running this file", or "if this is the file we're running". These have the same semantic structure as the Ruby code above... completely *unlike* the suggestion you make.

> And at this point, the "dispute" is about the method name.

No, it isn't. It's about how to make Ruby easier to read.

> Which method name would you choose, if you had two choices (.main? | .executed?) ?

Neither. The meaning that is intended cannot be expressed using the natural meaning of any single word. If the method was implemented, I'd still write it out long-hand.

I want to minimise the *cognitive load* on people who read my code.

Clifford Heath.

#10 - 06/06/2011 08:50 PM - lazaridis.com (Lazaridis Ilias)

Clifford Heath wrote:
[...] - omitting, to focus on essence

> > And at this point, the "dispute" is about the method name.
>
> No, it isn't. It's about how to make Ruby easier to read.

The status of this issue (#4824) is "agreed providing a method" (= provide an OO construct) and more actual "We need a better name." (Source: See the first comment).

So, it's about to make the language easier to read, in context of an OO construct.

I've added some elaborations, subjecting the (OO method) name. Ideally, it would be one word, but it's possible to use something like this:

```
if this_file_is_equal_with_the_program_name?
```

Although I dislike a two-word choice, the most logical seems to be this one:

```
if is_executed?
# refers to the actual main object. the user has *anyway* to learn: there is a "self" behind, an object behind

if self.is_executed? # can be written with "self", to make it more clear

main() if self.is_program?

main() if is_program?
```

So, the point is, to find a concise method name (ideally one word) which reads nice and fits in the overall existent naming scheme of the language.

(note that I "back-off" from this issue for now)

#11 - 06/06/2011 09:09 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Ruby allows a question mark in method names exactly for avoiding writing methods beggining with "is_" like in Java. I don't mind two word methods, but I don't want the first word to be a "is" when it is finished with a question mark. That is redundant.

By the way, among suggestions, I prefer either main? or main_script?. Maybe aliases :)

#12 - 06/08/2011 04:53 PM - headius (Charles Nutter)

On Mon, Jun 6, 2011 at 7:09 AM, Rodrigo Rosenfeld Rosas
rr.rosas@gmail.com wrote:

> Issue #4824 has been updated by Rodrigo Rosenfeld Rosas.
>
> Ruby allows a question mark in method names exactly for avoiding writing methods beggining with "is_" like in Java. I don't mind two word methods, but I don't want the first word to be a "is" when it is finished with a question mark. That is redundant.
>
> By the way, among suggestions, I prefer either main? or main_script?. Maybe aliases :)

I like main? as well. But I have a concern: the method would have to be able to see the caller's context, along the lines of eval. I *hate* the idea of adding more methods that can do that.

I might be more inclined to a keyword or pseudo constant along the lines of __FILE__, like __MAIN__ that produces true iff __FILE__ == $0. Something we can statically determine before runtime without digging around in the caller's frame.

- Charlie

#13 - 06/08/2011 06:23 PM - zenspider (Ryan Davis)

On Jun 8, 2011, at 00:44 , Charles Oliver Nutter wrote:

> I might be more inclined to a keyword or pseudo constant along the lines of __FILE__, like __MAIN__ that produces true iff __FILE__ $0. Something we can statically determine before runtime without digging around in the caller's frame.

While I think the feature request is a bit inane to begin with, I think __MAIN__ is a beautifully pragmatic compromise. Easy to implement on all impls, non-hacky, and yet makes its intention very clear.

#14 - 06/08/2011 08:29 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Em 08-06-2011 06:13, Ryan Davis escreveu:

> On Jun 8, 2011, at 00:44 , Charles Oliver Nutter wrote:
>
> > I might be more inclined to a keyword or pseudo constant along the lines of __FILE__, like __MAIN__ that produces true iff __FILE__ == $0. Something we can statically determine before runtime without digging around in the caller's frame.
> > While I think the feature request is a bit inane to begin with, I think __MAIN__ is a beautifully pragmatic compromise. Easy to implement on all impls, non-hacky, and yet makes its intention very clear.

I like the idea too. I just think that Ruby is very simple to start with because it is very consistent and with a few rules. Someone would expect __MAIN__ to be a constant, which is not. Maybe something like __MAIN? would show the contrast to something that seems to be like a constant (MAIN) but doesn't seem at the same time (?). I know this is still confusing, but I prefer something like this instead of a pure __MAIN__...

#15 - 06/08/2011 08:31 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Actually, maybe something like the snippet below would be ideal:

```
if defined?(__MAIN__) ...
```

This means __MAIN__ is still a constant, but one defined by the interpreter in some conditional way that is injected only in a single file (the main one).

#16 - 06/08/2011 08:53 PM - spatulasnout (B Kelly)

Hi,

Rodrigo Rosenfeld Rosas wrote:

> I like the idea too. I just think that Ruby is very simple to start with because it is very consistent and with a few rules. Someone would expect __MAIN__ to be a constant, which is not.

Why would one expect __MAIN__ to be any more or less of a constant than __FILE__ or __LINE__ or __method__ ?

Seems to me its behavior would be consistent with the others.

Regards,

Bill

#17 - 06/08/2011 09:23 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Em 08-06-2011 08:50, Bill Kelly escreveu:

> Why would one expect __MAIN__ to be any more or less of a constant than __FILE__ or __LINE__ or __method__ ?
>
> Seems to me its behavior would be consistent with the others.

Hi Bill, yes, you're right. I agree with you. I didn't think about this before :)

#18 - 06/08/2011 11:29 PM - nobu (Nobuyoshi Nakada)

Hi,

At Wed, 8 Jun 2011 16:44:55 +0900,
Charles Oliver Nutter wrote in [ruby-core:36833]:

> I might be more inclined to a keyword or pseudo constant along the lines of __FILE__, like __MAIN__ that produces true iff __FILE__ == $0. Something we can statically determine before runtime without digging around in the caller's frame.

__FILE__.main?

--
Nobu Nakada

#19 - 06/08/2011 11:53 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Em 08-06-2011 11:28, Nobuyoshi Nakada escreveu:

> Hi,
>
> At Wed, 8 Jun 2011 16:44:55 +0900,
> Charles Oliver Nutter wrote in [ruby-core:36833]:
>
> > I might be more inclined to a keyword or pseudo constant along the lines of __FILE__, like __MAIN__ that produces true iff __FILE__ == $0. Something we can statically determine before runtime without digging around in the caller's frame.
>
> __FILE__.main?

I loved this one!

#20 - 06/09/2011 01:13 AM - lazaridis.com (Lazaridis Ilias)

Nobuyoshi Nakada wrote:

> Hi,
>
> At Wed, 8 Jun 2011 16:44:55 +0900,
> Charles Oliver Nutter wrote in [ruby-core:36833]:
>
> > I might be more inclined to a keyword or pseudo constant along the lines of __FILE__, like __MAIN__ that produces true iff __FILE__ == $0. Something we can statically determine before runtime without digging around in the caller's frame.

@C.O. Nutter
I you dislike "digging around in the caller's frame", then you can possibly implement it in a different way.

> __FILE__.main?

This is not an OO approach, even I would prefer to use "if __FILE__ == $0" instead.

I would expect to see __FILE__.main? in python, not in ruby.

There is already an object available, accessible via "self".

```
main? # read: is main?
 # do main stuff
end
```

or

```
self.main? # read: self is main?
 # do main stuff
```

```
end
```

in some way, self refers to the file or code module/object.

#21 - 06/09/2011 01:23 AM - mfn (Markus Fischer)

Hi,

I take the courtesy to hijack this because ...

On 08.06.2011 13:31, Rodrigo Rosenfeld Rosas wrote:

> Issue #4824 has been updated by Rodrigo Rosenfeld Rosas.
>
> Actually, maybe something like the snippet below would be ideal:
>
> ```
> if defined?(__MAIN__) ...
> ```
>
> This means __MAIN__ is still a constant, but one defined by the interpreter in some conditional way that is injected only in a single file (the main one).

This "injected only in a single file" makes me wonder about one thing:
does Ruby in some way provide a per file context?

I guess those familiar with Python will recognize this immediately, as a file is treated as a (python) module which has it's on scoping and provides the ability for others to "import" only certain features of a module/file you want.

I though it would give a nice addition to Ruby but as I understand it, it would be quite radical as Ruby has the module keyword already and things work (quite?) differently.

- Markus

#22 - 06/09/2011 02:05 AM - lazaridis.com (Lazaridis Ilias)

Markus Fischer wrote:

> Hi,
>
> I take the courtesy to hijack this because ...
> [...]

Hijacking issues on an issue-tracking-system is really not the way to go.

Better open a new issue (or a new discussion topic) and place a link to it, if it's somehow related.

#23 - 06/10/2011 02:30 AM - lazaridis.com (Lazaridis Ilias)

Lazaridis Ilias wrote:

> Nobuyoshi Nakada wrote:
> [...]
>
> > __FILE__.main?
>
> This is not an OO approach, even I would prefer to use "if __FILE__ == $0" instead.
>
> I would expect to see __FILE__.main? in python, not in ruby.
>
> There is already an object available, accessible via "self".
> [...]
>
> ```
> self.main? # read: self is main?
>  # do main stuff
> end
> ```
>
> in some way, self refers to the file or code module/object.

Correcting myself:

Taking in account that "self" refers to the "main" object (the global object) and not to the file object (as it should, from my point of view), possibly this one could do it:

```
FILE.executed?
```

(this could get a related method: FILE.imported?)

Is there anything that disallows usage of "FILE" instead of "__FILE__"? I personally cannot look at those __x__ things when writing OO (one reason I dropped python).

A convention like: CAPITALS for immutable constant objects (without those __ __)?

#24 - 06/10/2011 04:49 AM - Cezary (Cezary Baginski)

This may seem like heresy, but isn't really:

```
__FILE__ == $0
```

just a hack for letting a file be *both* a script and a "library" at the same time? With the only sane use (I can think of) being adding unit tests?

This was probably useful in the early years of Ruby, but now with the internet, social coding, methodologies, TDD, BDD, packaging (gems), etc. - doesn't it make more sense to have tests and scripts in separate files?

Why add a construct for handling a block of code that cannot be called in any other way, than running the script directly, creating dead code that isn't included in coverage?

#25 - 06/10/2011 07:20 AM - rocky (Rocky Bernstein)

Cezary Baginski wrote:

> This may seem like heresy, but isn't really:
>
> `__FILE__ == $0`
>
> just a hack for letting a file be *both* a script and a "library" at the same time? With the only sane use (I can think of) being adding unit tests?

I wrote my thoughts regarding this to ruby-core on June 5-6. See http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/36772. But I realize that doesn't get reflected here in redmine.

It has one answer to your question which, in sum, is "demo code". Demonstration code is not the same as a unit test.

But there is another use. One can write a program that has a command-line interface, but folks can use this as a library instead. For example see, https://github.com/rocky/ps-watcher

> This was probably useful in the early years of Ruby, but now with the internet, social coding, methodologies, TDD, BDD, packaging (gems), etc. - doesn't it make more sense to have tests and scripts in separate files?
>
> Why add a construct for handling a block of code that cannot be called in any other way, than running the script directly, creating dead code that isn't included in coverage?

#26 - 06/10/2011 09:08 AM - rbjl (Jan Lelis)

I'd still prefer a Kernel method - It's about better readability, isn't it? My favourites:

```
directly_executed?
standalone?
```

If it should be some kind of keyword, I can't see any serious issue against the __MAIN__ solution. It's true in the "main" file, false in all others and

```
if __MAIN__
```

looks OK.

#27 - 06/10/2011 09:53 PM - austin (Austin Ziegler)

On Thu, Jun 9, 2011 at 3:49 PM, Cezary Baginski
cezary.baginski@gmail.com wrote:

> Issue #4824 has been updated by Cezary Baginski.
> This may seem like heresy, but isn't really:
>
> `__FILE__`

#28 - 06/10/2011 11:23 PM - Cezary (Cezary Baginski)

On Fri, Jun 10, 2011 at 07:20:32AM +0900, Rocky Bernstein wrote:

> Issue #4824 has been updated by Rocky Bernstein.
>
> Cezary Baginski wrote:
>
> > This may seem like heresy, but isn't really:
> >
> > ```
> > __FILE__ == $0
> > ```
> >
> > just a hack for letting a file be *both* a script and a "library"
> > at the same time? With the only sane use (I can think of) being
> > adding unit tests?
>
> I wrote my thoughts regarding this to ruby-core on June 5-6. See
> http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/36772.

Yes, I read your comments (I read the list). It was what got me
thinking. I didn't like checking using file path strings either.

> It has one answer to your question which, in sum, is "demo code".
> Demonstration code is not the same as a unit test.

Yes, but shouldn't that be in a README or example.rb instead?

> But there is another use. One can write a program that has a
> command-line interface, but folks can use this as a library instead.
> For example see, https://github.com/rocky/ps-watcher

Shouldn't there be an executable script installed in bin/ loading a
lib/ps-watcher/cli.rb file ? It would be accessible from $PATH.
And unit/integration tests can be simpler and more robust.

I understand the use case(s), but I don't see why '__FILE__ == $0' is
really that useful and good practice enough to be explicitly supported
by the language.

As for the name - anything containing 'main' assumes familiarity with
C-type languages, which may or may not be the case for novices.

--
Cezary Baginski

#29 - 06/10/2011 11:23 PM - Cezary (Cezary Baginski)

On Fri, Jun 10, 2011 at 09:35:24PM +0900, Austin Ziegler wrote:

> On Thu, Jun 9, 2011 at 3:49 PM, Cezary Baginski
> cezary.baginski@gmail.com wrote:
>
> > Issue #4824 has been updated by Cezary Baginski.
> >
> > ```
> > __FILE__ == $0
> > ```
> >
> > just a hack for letting a file be *both* a script and a "library"
> > at the same time? With the only sane use (I can think of) being
> > adding unit tests?
>
> Tests are most *certainly* not the only good reason for this feature,
> and I would regret its departure.

By "sane" use I didn't mean "good" use. Especially with a team working
together and source control. I have nothing against the current use of
checking for main script, but I don't think promoting its use through
additional language support is beneficial.

--
Cezary Baginski

#30 - 06/13/2011 10:29 PM - Cezary (Cezary Baginski)

On Sat, Jun 11, 2011 at 11:20:31AM +0900, Rocky Bernstein wrote:

> On Fri, Jun 10, 2011 at 10:03 AM, Cezary cezary.baginski@gmail.com wrote:
>
> > On Fri, Jun 10, 2011 at 07:20:32AM +0900, Rocky Bernstein wrote:
> >
> > > It has one answer to your question which, in sum, is "demo code".
> > > Demonstration code is not the same as a unit test.
> >
> > Yes, but shouldn't that be in a README or example.rb instead?
>
> I prefer demo code to be runable in the same way that tests are runable. And
> for simple things, one file is better, feels more lightweight, and is less
> clumsy (...)

So why not instead check for specific script arguments, like '--demo', '--test', '--help', etc.? If checking for '__FILE__ == $0' is really the best solution in a given case, my question is: why provide a special method for the same thing?

> I think we can tolerate different styles, no?

Absolutely! So why not just use the following in such programs:

```
def main?; __FILE__ == $0; end      # for 'if main?'

class String; def main?; self == $0; end; end  #for '__FILE__.main?'
```

That way, anyone can use their own style for the idiom with little coding overhead. And it will work with any Ruby version out there.

> > It would be accessible from $PATH.
> > And unit/integration tests can be simpler and more robust.
>
> If you looked at the project, you'll see that there are tests. I don't see
> how unit/integration tests would be made simpler by putting the trivial
> command-line interface, or main() routine, in another file.

Regarding the ps-watcher project specifically, it requires other project files anyway, so you cannot use the file standalone. So, in that case, you can assume most of the code (demo?) can be moved to other files, leaving only the "main" code, after which the '__FILE__' check around it can be removed altogether.

There is no README file, so it is a little hard to figure out how to use the application, without reading through everything. Replacing the "main checking" with a new Ruby 1.9.3 method (that would do the same) is something I find not worth the time to even discuss.

As for tests - there already is a Rakefile which runs them. Why not add a 'demo' task there as well?

I'm not trying to criticize the project here - I just don't see it as a convincing example showing how a new method for the idiom is valuable.

> Could it just be sufficient to accept that others find it useful
> even if it is not useful to you?

Sure, the idiom is useful, even if I have a hard time thinking of a case where its use is elegant, the best practice and recommended - enough to explain the absolute need for anything more that the way it is done currently (with __FILE__ == $0).

> This is now feels like discussions in Perl when folks would ask if "until" was really important enough when there was "if" and "not" And if the form with "statement if expression" was really warranted since you had "if expression ...".

I get what you are trying to say, but the example isn't a good one IMHO - keywords are much harder to emulate without language support, because they affect syntax. Things like Ruby's "unless" are valuable because of that. __FILE__.main? OTOH is an easy one-liner.

Let me try one: some people mix paint with screwdrivers, because they have them handy (after opening the paint box). Why would we want to add "paint mixer" features to *every* screwdriver to support this?

Having a special method for a special case is like having 7.is_five? or "y".is_yes?. Extreme examples, I know, but why is #main? an exception here?

> If you feel strongly about such matters, I suppose you could write a new programming language or perhaps carve out a subset of Ruby that does not have such things that you don't feel are useful or are good practice.

To me this argument is along the lines of: "If you don't like adding very useful is_mp3? and is_html? methods to String, why don't you design your own programming language?".

IMHO, the idiom is not generic enough to deserve its own method.

Actually, it reminds me of discussions about Rails's HashWithIndifferentAccess. That class handles a specialized case which is very hard to do otherwise. I understand why it isn't necessary to support it in Ruby - and I think many will agree, but why in that case do we need a specialized method to replace code that already does a job just fine?

I would really like some clear guidelines for proposing and accepting similar methods in the future.

I don't really care if a method is added or not, I am really interested in: on what basis was it decided that it is worth considering?

Ideally, a reason that would be more definite than just a matter of taste or popularity. And without scrutiny, we may be missing possibly more sensible options. For example:

```
require 'main'

Main.run do
  puts "Running a demo"
end
```

That way we can even call the main manually, handle exit code, override, etc. Just an idea though.

--
Cezary Baginski

#31 - 06/14/2011 08:23 PM - Cezary (Cezary Baginski)

On Tue, Jun 14, 2011 at 03:23:27PM +0900, Rocky Bernstein wrote:

> On Mon, Jun 13, 2011 at 9:25 AM, Cezary cezary.baginski@gmail.com wrote:
>
> > So why not just use the following in such programs:
> >
> > ```
> > def main?; __FILE__ == $0; end      # for 'if main?'
> > ```
>
> Simplicity and unreliability.

My first reaction is usually: is it valuable? Simple != valuable.

I think it is reliable enough for the cases mentioned. If reliability is an important issue here, then the implementation is more important than the name anyway. Unless the name is just a starting point for considering the issue at all.

Why not start with a gem first? Like Object#me (which became #tap) or #andand which IMHO is much more valuable but would greatly benefit from parser support in Ruby.

If simplicity is the main criteria, we could end up with Ruby's namespace exploding with "simplifying" methods that almost no one will use for various reasons. Why not create a 'main' gem and work on getting #andand (#._?) support in Ruby's parser instead?

#tap turned out awesome IMHO. I don't see #main? as revolutionary.

> Everything you suggest, adds more code. I want less boilerplate code in fewer files. That is what I meant by "lightweight".

Modularity and more code-sharing friendliness is more important. The ps-watcher project seems to reflect this - it contains more than one file, tests separate, Rakefile, functionality split up into small *.rb files, etc. If you like, I can spend some time to see what ps-watcher would like without the 'main' check. Not as a criticism, just as a way to support my point regarding design.

I think it would be great to first have a 'main' gem until the implementation matures. And it could be used in older Ruby applications immediately. Like #tap, #andand, etc.

> Early on in the thread, Matz had said he was amenable to the idea of adding a method. But he was unsure about the name. If he had indicated he wasn't interested, I would have dropped the topic and never have posted a response initially.

Exactly! But I'm still not convinced about the value of such a method. I know I'm dumb, but I don't think I'm dumb enough to not understand a simple, valuable use case where a method is really an improvement worth adding and backporting. Or why wasn't this ticket immediately rejected.

Initially, I assumed I'm an idiot and believed the experts on this list saw the value, which I couldn't. Being interested in improving my skills, I got curious to learn what I am not seeing.

There is so much functionality that doesn't belong in Ruby core that is way more valuable. How did this get anything else than "rejected"? My only guess is unfortunate popularity of a use case that in itself suggests bad design - which is why an alarm in my head went off.

> In my first post which you said I read, I also discussed why the idiom is unreliable.

Yes, and I can imagine it being a problem fixable with a well thought out implementation. It is good you brought it up.

I just don't see why improving the reliability of such a minor issue (IMHO) is really productive and worth any other reaction than rejecting or at least suggesting a new gem first.

Sorry for prolonging this discussion - I believe it may be worth learning to deal with this issue (or even just people like me) and preventing a whole class of similar cases (discussions?) in the future.

If my issues are pointless - let me know and I'll put in more trust in faith in the experts reading on ruby-core and give up on trying to change the way I think.

Thanks!

--

Cezary Baginski

#32 - 06/16/2011 06:29 AM - sandal (Gregory Brown)

On Sat, Jun 4, 2011 at 10:17 AM, Yukihiro Matsumoto matz@ruby-lang.org wrote:

> Issue #4824 has been updated by Yukihiro Matsumoto.
>
> I agree providing a method to tell whether it is loaded as a library or is executed as a command is more elegant than '**FILE**

#33 - 06/19/2011 01:23 AM - headius (Charles Nutter)

On Wed, Jun 15, 2011 at 4:24 PM, Gregory Brown
gregory.t.brown@gmail.com wrote:

> Lastly, I think __MAIN__ is reasonable if a method is not to be used,
> though it feels a bit too magic for me.

I'll repeat what others have repeated...more magic than __FILE__,
__LINE__, or even $0 itself?

- Charlie

#34 - 06/23/2011 03:20 PM - lazaridis.com (Lazaridis Ilias)

The sentence "defines" essentially the terminology:

Yukihiro Matsumoto wrote:

> I agree providing a method to tell whether it is loaded as a library or
> ^^^^^^^^^
> "is loaded"

loaded?

> is executed as a command
> ^^^^^^^^^^
> "is executed"

executed?

> is more elegant than '__FILE__ == $0', but I am still afraid #executed? can mean multiple ways so that the name can cause confusion. We need a better name.

Thus the question to focus on is:

- is "executed?" not clear enough?

If not, possibly just "loaded?" can be introduced.

```
main() unless loaded?
```

#35 - 03/25/2012 04:06 PM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

#36 - 06/08/2012 05:12 AM - fxn (Xavier Noria)

I vote for __MAIN__.

I think "main" will resonate to a lot of people with different backgrounds to indicate the entry point of the program, and choosing a keyword instead of a method seems coherent with other keywords like __FILE__.

#37 - 11/20/2012 10:58 PM - mame (Yusuke Endoh)

- Target version set to 2.6

#38 - 01/25/2015 10:18 PM - sawa (Tsuyoshi Sawada)

Close to Nobu's proposal FILE.main?, but I don't think it makes sense to introduce two new things FILE and main? that can only be used under this combination. I propose defining a method on the existing File class. Something along:

```
File.main?
File.from_here?
```

Edit: Seeing Nobu's comment 39, I realize it was my mistake. Sorry.

#39 - 01/25/2015 11:29 PM - nobu (Nobuyoshi Nakada)

- Description updated

My proposal was __FILE__.main?, not FILE.main?.
Just you're confused by markdown.

#40 - 01/26/2015 06:25 PM - headius (Charles Nutter)

The "main" toplevel object already gets special methods, so why not just define main.main? Avoid polluting Kernel or creating a special class for __FILE__ just to support a "main" feature.

#41 - 12/25/2017 06:15 PM - naruse (Yui NARUSE)

- Target version deleted (2.6)

#42 - 12/26/2017 07:22 AM - mame (Yusuke Endoh)

The names (or APIs) suggested so far:

```
executed?
required?
main?
first_script?
main_script?
run_script?
is_executed?
is_program?
defined?(__MAIN__)
__FILE__.main?
FILE.executed?
directly_executed?
standalone?
loaded?
__MAIN__
File.main?
File.from_here?
main.main?
```

#43 - 01/29/2018 05:30 AM - naofumi-fujii (naofumi fujii)

Hi, i created a patch for this ticket.
please take a look.

https://github.com/ruby/ruby/pull/1802

#44 - 03/04/2018 02:25 PM - shevegen (Robert A. Heiler)

It seems as if this may be decided at the upcoming ruby developer meeting.

Matz voiced concern about the suggested name. Yusuke Endoh provided a list of alternative names (or rather, a summary). I assume that the name for the functionality here is the primary concern; the functionality I assume is fine, so let's have a look at **possible names**.

I myself currently use this:

```
if __FILE__ == $PROGRAM_NAME
```

The reason why I use the longer $PROGRAM_NAME is because it tells me more than $0. $0 really does not tell me anything; the only advantage $0 has is that it is short. But a method call would be even shorter.

The name "executed?" is not good, I think, because it implies a state that has already been passed, in the past (that is my interpretation, as a non-native english speaker).

I like the name **run_script?** - although it may not be a 100% fitting name (what is a "script" for example), it has a meaning that tells me something. Perhaps **run_file?** may be a better name, but I am not absolutely sure either. What I like about this is that it is more explicit and conveys some meaning to me when I look at code.

I don't like the name **main.main?** because of the repetition. It seems superfluous to repeat it; and "main?" also tells me nothing really.

I don't like __ MAIN __ much either because of the leading and trailing __. I understand that ruby uses it already internally and we use it via **FILE** == comparisons but I do not think that __ MAIN __ is any improvement over **FILE** ==

Sorry for splitting the above, **markdown** appears to think that __ means some bold emphasis.

defined?(**MAIN**) is bad because it requires too much typing and is not a substantial improvement over **FILE** ==

I also do not like the calls to "File.method" such as FILE.executed? - calls to Kernel are better, IMO.

Anyway, I don't want to inflate this too much so I will add some names as suggestions - I will focus on something that may have a meaning from the name alone, and assumingly resides on the Kernel namespace; I'll also update on mame's list, at the least those entries that I think make for better names:

(I think I prefer names that refer to "standalone" and "run" or something along those lines.)

```
is_standalone?
run_standalone?
standalone?
is_executed?
is_main?
run_script?
run_file?
shall_run?
is_runnable?
```

I prefer the first two variants:

is_standalone? or **run_standalone?**

They would convey that the file may be run "standalone", that is, "on its own" - similar to the current check that I (and others) do such as via:

```
if __FILE__ == $PROGRAM_NAME

end
```

Anyway, if we ignore any specific name, perhaps if we can not narrow it down to just one name, we (or the ruby core team at a meeting) could pick three variants, and we could use all three as experimental feature, before deciding on the primary way and removing the others (or keeping them as aliases; but I think, although I love aliases, it may be better to use only one way here, for a method, so that people all use the same in their code. I love aliases but in this case, I think it is better to encourage only the best name. When we have e. g. 3 possible names, I am sure it will be much easier to select the best name altogether after a while of ruby hackers using these names; and it could be an experimental feature before the xmas release, upon where it could be finalized and the best one decided. Or matz just decides on a name, that may be the shortest

and simplest perhaps. :D My primary hope here is that
the name is a good name which is used a lot - it would
not be good if people prefer "if **FILE** == $PROGRAM_NAME"
because they dislike the new name)

#45 - 03/04/2018 04:45 PM - dsferreira (Daniel Ferreira)

Why not:

```
Kernel#executable?
# => true if FILE == $0
```

meaning: Is the file good to be executed?

#46 - 03/04/2018 10:39 PM - graywolf (Gray Wolf)

I'm not a native English speaker, so this whole comment might be completely off,
but at least to me the ending ~able? implies that it is possible to use it
that way, not necessarily that is was used that way.

So Kernel#executable? tells me that something is able to be executed, not that
it was in fact executed. is_standalone? imho suffers from the same thing (btw
why not just standalone?).

At least to me Kernel#executed? looks better.

But I like __MAIN__ or Kernel#main? the best. Anyone who ever saw C or Python
would be able to guess what it means even without checking documentation.

#47 - 03/04/2018 11:12 PM - dsferreira (Daniel Ferreira)

graywolf (Gray Wolf) wrote:

> So Kernel#executable? tells me that something is able to be executed, not that
> it was in fact executed.

Correct, Kernel#executable? infers if the code is able to be executed.

Usually when we use

```
if __FILE__ == $0
  # call code to be run
  main(ARGV)
end
```

We are saying that the code should only be executed if the condition applies.

The code will be executed once we call main(ARGV).
The code is able to be executed if __FILE__ == $0.

#48 - 03/04/2018 11:18 PM - graywolf (Gray Wolf)

dsferreira (Daniel Ferreira) wrote:

> [..]
>
> Usually when we use
>
> ```
> if __FILE__ == $0
>   # call code to be run
>   main(ARGV)
> end
> ```
>
> We are saying that the code should only be executed if the condition applies.
>
> [..]

Exactly, and the condition is: *was this file directly* **executed?**.

Or: *is this the* **main?** *script*
Or: *is this the* **main_script?**

I mean, I get what you mean, but still think it's a bit confusing.

EDIT:

Given these two snippets in pseudo-code, which better shows the intent of the if?

```
if this file is the main script
    do main stuff
end
```

or

```
if the code below is able to be executed
    do main stuff
end
```

Sure, both do the same thing but I think the first one shows the intent better.

#49 - 03/04/2018 11:26 PM - phluid61 (Matthew Kerwin)

For what it's worth, I think __main__ is best, or main? if we're moving away from underscores for source-level metacode.

"Executable" is such an overloaded term, and "executed" is weird (isn't all code executed?); the convention of a single "main" entry point has been around for a long time now.

#50 - 03/04/2018 11:26 PM - dsferreira (Daniel Ferreira)

graywolf (Gray Wolf) wrote:

> Exactly, and the condition is: was this file directly executed?.

We have two different contexts:

- File context (FILE#executed?) - Was the file executed?
- Code context (Kernel#executable?) - Is the code executable?

I prefer to work at the code context level since we can then add other conditions to allow a code to be executable.
e.g.

```
def executable?
  __FILE__ == $0 && ARGV[1] == "foo"
end
```

#51 - 03/04/2018 11:33 PM - dsferreira (Daniel Ferreira)

phluid61 (Matthew Kerwin) wrote:

> For what it's worth, I think **main** is best, or main?

I would agree with main in the following way:

```
# Kernel#main method to be overridden and only run if __FILE__ == $0
def main
   Foo.bar
end
```

#52 - 03/04/2018 11:40 PM - phluid61 (Matthew Kerwin)

dsferreira (Daniel Ferreira) wrote:

> phluid61 (Matthew Kerwin) wrote:
>
> > For what it's worth, I think **main** is best, or main?
>
> I would agree with main in the following way:

__main__, not **main**. Returns a boolean value, which can be used as a predicate:

```
if __main__
  foo
end
```

#53 - 03/05/2018 04:57 AM - sawa (Tsuyoshi Sawada)

From the code __FILE__ == $0, we can tell that what matters is that this condition is within the main file that is executed. If we are to write something like if __main__ or if main?, I have a concern that it will become less clear that that condition has to be placed within the main file. Some (careless) users might take it, for example, that such condition becomes false when it is a part of a gem, and becomes true when it is not part of the gem, irrespective of which file that condition is placed.

In order to avoid such problems, I think a method name like main_file? would be better. Also, in order to give flexibility so that the condition can be placed in files other than the main file, it might be good to allow an optional argument as in main_file?(file_name), where file_name defaults to __FILE__.