Ruby - Bug #17144

Tempfile.open { ... } does not unlink the file

09/03/2020 09:21 AM - Eregon (Benoit Daloze)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:		**Backport:**	2.5: UNKNOWN, 2.6: UNKNOWN, 2.7: UNKNOWN

Description

```
ruby -rtempfile -e 'Tempfile.open("txt") { |f| $path = f.path }; p File.exist?($path)'
true
```

but it should be false.

This means even after the block finishes to execute the file still exists on a disk
And this might or not be addressed by finalization much later on.

This is inconsistent with the resource block pattern and basically all usages of SomeClass.open { ... }.
There are more than 10 SomeClass.open in core+stdlib, and AFAIK all these methods release all resources at the end of the block, except Tempfile.open.

Since the block creates the file, it should also delete it, so there are no leftovers after the block.

The (English) docs don't mention the file is kept on disk after the block:
https://docs.ruby-lang.org/en/2.7.0/Tempfile.html#method-c-open

I made a PR to do unlink in https://github.com/ruby/tempfile/pull/3 and some commits in ruby/ruby (notably https://github.com/ruby/ruby/commit/fa21985a7a2f8f52a8bd82bd12a724e9dca74934).

However it can cause some incompatibility, if an existing usage relied on the block only closing the file descriptor but not unlink the path.
See https://github.com/ruby/tempfile/issues/2#issuecomment-686323507

When integrating this change in ruby/ruby, I found that many usages expected that the file be unlinked automatically, but had to add an extra ensure tempfile.close! on top of the block:
https://github.com/ruby/ruby/commit/e8c3872555fc85640505974e6b1c39d315572689 (later partially reverted because such libraries probably want to keep compat with older Rubies)

In all of ruby/ruby I found only 2 usages depending on the file to still be on disk after the block.

@shugo (Shugo Maeda) brought to my attention that Tempfile.create { ... } unlinks the file (Tempfile.create seems to exist since 2.1 yet very few seem to know about it).
I think the semantics of Tempfile.create { ... } is what the vast majority of usages want for Tempfile.open { ... }.
open is the name everyone expects, so I think it's best if Tempfile.open { ... } also links.
create doesn't sound like it will cleanup to me.

As an optimization we can use Tempfile.create(&block) to implement Tempfile.open { ... } to avoid creating needlessly a finalizer.

Found by https://github.com/ruby/spec/commit/d347e89ef6c817e469a1c25985dbd729c52b80fd and the leak detector.

From https://github.com/ruby/tempfile/issues/2

So, OK to keep this change and make Tempfile.open { ... } do what most usages expect,
even if we have to update a few usages that relied on the file to still exist after the block?

Related issues:

Related to Ruby - Feature #7148: Improved Tempfile w/o DelegateClass	**Assigned**

History

#1 - 09/03/2020 09:32 AM - Eregon (Benoit Daloze)

Out of these 2 usages that relied on the file to still exist, 1 was incorrect:
https://github.com/ruby/ruby/blob/e8c3872555fc85640505974e6b1c39d315572689/lib/reline/line_editor.rb#L2085-L2087
The GC can trigger anywhere between these 3 lines and it would break.

The other usage seemed correct, but was IMHO quite ugly (a resource block + an ensure, needed to workaround this issue):
https://github.com/ruby/ruby/blob/e8c3872555fc85640505974e6b1c39d315572689/test/openssl/test_x509store.rb#L36-L49

#2 - 09/03/2020 12:58 PM - akr (Akira Tanaka)

I sympathize with this issue.
I wish Tempfile.open had worked like Tempfile.create from the beginning.

But changing it is incompatible.
It is not our (current) practice that introducing an incompatible change without prior notice.

It would be possible to change it with some migration period (if matz approves).
But I doubt that the benefits are not greater than the pain of its incompatibility because desired functionality, "block exit removes the temporary file", is already provided in Tempfile.create.

Anyway, it would be good to improve the document of Tempfile.open to recommend Tempfile.create.

#3 - 09/03/2020 01:15 PM - akr (Akira Tanaka)

One idea: creating tmpfile library and
Tmpfile.open calls Tempfile.create.

This also resolves an inconsistency between tempfile.rb and tmpdir.rb.
For this consistency, File.tmpfile may be better than Tmpfile.open
because it is similar to Dir.tmpdir.

#4 - 09/03/2020 01:44 PM - Eregon (Benoit Daloze)

It seems hard to deprecate here without changing behavior. Any idea?
OTOH, when an usage relied on the file to still exist it should be quite clear what happens (the file will not be there, so an exception).
It took me seconds to find out when looking at the failing tests.

The change is already part of NEWS, I could mark it as experimental if desired.

I think very few cases will break, so it seems reasonable to change to me.
I'd think most cases which want the file to still be on disk either have all the logic inside the block, or don't use a block (so those are unaffected).
Ruby 3.0 has other incompatible changes of course, IMHO we need to evolve the old APIs, not be stuck forever with strange and non-intuitive APIs.

Many gems test against ruby-head, maybe let's simply wait a bit and see if they report some breakage?
From experience, that seems a practical way to estimate incompatibility.

#5 - 09/03/2020 03:05 PM - Glass_saga (Masaki Matsushita)

- Related to Feature #7148: Improved Tempfile w/o DelegateClass added

#6 - 09/03/2020 03:14 PM - Dan0042 (Daniel DeLorme)

-1 for breaking compatibility with no deprecation, just for the sake of perceived consistency.

But then again it's important to note that Tempfile.open{ } returns nil, so the only way to cause an incompatibility is if the blocks "leaks" the Tempfile reference, like Tempfile.open{ @f = _1 } or Tempfile.open{ return _1 }

#7 - 09/03/2020 03:15 PM - Glass_saga (Masaki Matsushita)

If we could allow incompatible changes of Tempfile in 3.0 or later, it is a good chance to reimplement it as a subclass of File without delegate.rb.
#7148

#8 - 09/03/2020 03:46 PM - Dan0042 (Daniel DeLorme)

> Tempfile.open{ } returns nil

I apologize for this brain fart. Tempfile.open{ } returns the result of the block. So it's entirely likely that someone would use
tmp = Tempfile.open{ |f| f.write(data); f }
instead of
tmp = Tempfile.open; tmp.write(data); tmp.close

#9 - 09/03/2020 06:56 PM - Eregon (Benoit Daloze)

Dan0042 (Daniel DeLorme) wrote in #note-6:

> -1 for breaking compatibility with no deprecation, just for the sake of perceived consistency.

Not "the sake of perceived consistency".
It's leaking a resource (a file on the disk) outside of a resource block.
Seems a severe issue to me.

And people must be annoyed all the time by this bug/surprising behavior.
I think we annoy less people by fixing this behavior (very few places need to adapt) than keeping it (all future usages have to use Tempfile.open { ... }; ensure or discover the non-standard-named method Tempfile.create { ... }).

Regarding the pattern, Tempfile.open requires this complete anti-pattern to correctly release all resources:
(from https://github.com/ruby/ruby/commit/e8c3872555fc85640505974e6b1c39d315572689)

```
begin
  tf = Tempfile.open 'test' do |io| # yet another variable name for the same thing
    io.write "..."
    io # leaking the argument of the resource block, that feels very hacky
  end
  # use tf.path
ensure
  tf.close! if tf # Am I using Ruby or Go? The block should deal with this.
end
```

Do we want to perpetuate this anti-pattern?

With this change, it's:

```
Tempfile.open 'test' do |io|
  io.write "..."
  io.close # or io.flush
  # use tf.path
end
```

If we want to deprecate then we'd deprecate Tempfile.open with a block (and use Tempfile.create instead).
That's for sure causing more pain than the change I did.

#10 - 09/04/2020 01:18 AM - akr (Akira Tanaka)

Eregon (Benoit Daloze) wrote in #note-9:

> Dan0042 (Daniel DeLorme) wrote in #note-6:
>
> > -1 for breaking compatibility with no deprecation, just for the sake of perceived consistency.
>
> Not "the sake of perceived consistency".
> It's leaking a resource (a file on the disk) outside of a resource block.
> Seems a severe issue to me.

It is compared to wrong use of Tempfile.open

I understand "perceived consistency" is compared to Tempfile.create.

> And people must be annoyed all the time by this bug/surprising behavior.
> I think we annoy less people by fixing this behavior (very few places need to adapt) than keeping it (all future usages have to use Tempfile.open { ... }; ensure or discover the non-standard-named method Tempfile.create { ... }).

I feel that "create" is the second standard choice of factory method.

> Regarding the pattern, Tempfile.open requires this complete anti-pattern to correctly release all resources:
> (from https://github.com/ruby/ruby/commit/e8c3872555fc85640505974e6b1c39d315572689)
>
> ```
> begin
>   tf = Tempfile.open 'test' do |io| # yet another variable name for the same thing
>     io.write "..."
>     io # leaking the argument of the resource block, that feels very hacky
>   end
>   # use tf.path
> ```

> ```
> ensure
>   tf.close! if tf # Am I using Ruby or Go? The block should deal with this.
> end
> ```
>
> Do we want to perpetuate this anti-pattern?
>
> With this change, it's:
>
> ```
> Tempfile.open 'test' do |io|
>   io.write "..."
>   io.close # or io.flush
>   # use tf.path
> end
> ```
>
> If we want to deprecate then we'd deprecate Tempfile.open with a block (and use Tempfile.create instead).
> That's for sure causing more pain than the change I did.

We can use Tempfile.create.
There is no incompatibility pain if we don't change Tempfile.open behavior.

Also, Tempfile.create has advantages to the proposed Tempfile.open change:

- It is available for all maintained Ruby versions. (It is available since Ruby 2.1. Zero-argument call is permitted since Ruby 2.4.) Tempfile.create is usable without worrying Ruby version dependencies.
- The proposed change doesn't eliminate all curiousness of Tempfile.open. It still use Tempfile class for delegation which is eliminated in Tempfile.create.

As far as I understand the advantage of the proposed Tempfile.open change over Tempfile.create is just a method name which is consistent with other classes.

#11 - 09/06/2020 12:55 AM - Dan0042 (Daniel DeLorme)

Austin Ziegler wrote on mailing list:

> If we don’t change the behaviour, could we at least modify the documentation for Tempfile.open to recommend most people use Tempfile.create, since I don’t think that I’ve ever used it and reach for Tempfile.open most of the time, because of its similarity to File.open (and I’ve been using Ruby since 2002).

That would be good, and in fact I believe improving the documentation is the best way to address this issue.

#12 - 09/25/2020 04:45 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Closed

To keep compatibility, we are not going to change the behavior of Tempfile.open. But to reduce the confusion, documentation was updated.
In the future, maybe something like RuboCop warning will be convienient.

Matz.

#13 - 09/25/2020 09:30 AM - Eregon (Benoit Daloze)

OK, https://github.com/ruby/tempfile/pull/4 is the PR that reverts it and already applied to ruby master.

I will try to improve the documentation further, something like
"Tempfile.open is basically deprecated but kept for compatibility, please use Tempfile.create instead whenever possible".

#14 - 09/25/2020 04:24 PM - akr (Akira Tanaka)

Tempfile.open would be useful when an user want to remove the file by GC.
Tempfile.create doesn't provide the feature, intentionally.

So, as far as an user needs temporary file removal by GC, we should not deprecate Tempfile.open
(unless we provide an alternative).