Ruby - Feature #4211

Converting the Ruby and C API documentation to YARD syntax

12/27/2010 06:00 AM - lsegal (Loren Segal)

Status:	Rejected	
Priority:	Normal	
Assignee:	drbrain (Eric Hodel)	
Target version:	2.6	

Description

=begin
The Ruby (high level core/stdlib) documentation and its C API (low level) counterparts currently use two different formats (and tools) to write and generate the final docs. This creates a problem for committers and users alike, where:

- Documentation is hard to write, because there is no single documentation style to follow (it depends on the API), and the two existing syntaxes are very different.
- Documentation is harder to read because the style and formatting differ due to the lack of consistent enforcement of a single style.
- Documentation for the C API (specifically) is harder to find

Currently, Doxygen @tag style syntax is slowly being introduced to improve the documentation of Ruby's C API, but this does not solve the three issues noted above. I propose to unify the documentation style used in the codebase to a single format (originally on ruby-core:33883[1]) by using YARD[2] syntax, which is very much like the Doxygen @tag style syntax being introduced anyway. Switching to YARD introduces a number of benefits, namely:

- There would be a single syntax to learn for committers wishing to document code, making it easier to write,
- The documentation would be formatted and styled consistently across both APIs for users to read,
- Documentation would be generated by a single tool for both APIs, meaning a simpler workflow for documenters and users wishing to generate the docs themselves.

I pointed out in the original mailing list that much of the documentation problems come from a lack of unified styling, causing parts of documentation to be (or become) inaccurate due to a variety of "human-error" type issues, and because there are no tools to check the correctness. I believe switching to a unified style and making sure it is used consistently will solve many of those issues even without tooling, because it is easier to manually check for errors with a consistent formatting. Furthermore, using a consistent style allows us to take advantage of our tooling to check basic correctness (or "lint") the docs for simple errors. YARD already has tools to do this kind of thing (and they are easily improved), but they depend on that consistent syntax.

As far as the C API goes, there is little difference in the existing doxygen syntax (except that I'd suggest the '@tag' instead of Doxygen's alternative '\tag' prefixes, for compatibility). As I wrote in the above ruby-core thread, YARD can already handle most of the written doxygen documentation. Granted, a lot of the support for actually *generating* documentation for a straight "C" style API is missing in YARD, but as I mentioned, I would be willing to improve this support if there is a willingness by the ruby-core developers to create a unified documentation style.

h3. Steps forward:

We should first discuss whether the Ruby core developers are in favor of such a change. In the event that they are, we would have to look at a few things:

- Maintaining compatibility with RDoc (or adding YARD's @tag style support to RDoc) for the high level Ruby API docs while converting the syntax. I have a few ideas on how this can be done.
- Improving YARD's ability to generate HTML for straight "C" codebases (which I can implement, if we get this far)
- Any other issues / reservations raised by the Ruby core team

This is certainly a large proposal, and has some compatibility snags (with RDoc, for instance). Regardless, I think these issues can be worked around or dealt with for the most part, and the benefits of much improved documentation, which Ruby really needs, are certainly worth the effort.

[1]: ruby-core:33883: http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/33884
[2]: YARD: http://yardoc.org
=end

History

#1 - 12/27/2010 12:00 PM - naruse (Yui NARUSE)

=begin
Your proposal is interesting, but as a lazy comitter, I want to know what we must do for this change.
For example:

- Committers must learn YARD syntax (what we should read?)
- convert existing documents (who will do?)
- fix some tools
- fix build process

I also want each vision for current use cases, like bin/rdoc and bin/ri.

After all work is calculated and it is reasonable, I won't object the migration.
=end

#2 - 12/28/2010 05:20 AM - tenderlovemaking (Aaron Patterson)

=begin
On Mon, Dec 27, 2010 at 12:01:00PM +0900, Yui NARUSE wrote:

> Issue #4211 has been updated by Yui NARUSE.
>
> Your proposal is interesting, but as a lazy comitter, I want to know what we must do for this change.
> For example:
>
> - Committers must learn YARD syntax (what we should read?)

http://rubydoc.info/docs/yard/file/docs/GettingStarted.md

> - convert existing documents (who will do?)
> - fix some tools
> - fix build process

I think if we do this correctly, conversion may be done slowly without
modifications of our tools.

> I also want each vision for current use cases, like bin/rdoc and bin/ri.
>
> After all work is calculated and it is reasonable, I won't object the migration.

I spoke with Loren about this issue on IRC. I think we can accomplish
his goals by adding @tag support to RDoc.

Adding @tag support to RDoc would allow us to unify Doxygen and RDoc
documentation formats. But this means we must define the meaning of
"@tag support".

Currently, RDoc will ignore @tags in your documentation. For example,
this code:

```
class Foo
  # Converts the object into textual markup given a specific `format`
  # (defaults to `:html`)
  #
  # == Parameters:
  # format::
  #   A Symbol declaring the format to convert the object to. This
  #   can be `:text` or `:html`.
  #
  # == Returns:
  # A string representing the object in a specified
  # format.
  #
  def to_format(format = :html)
    # format the object
  end
end
```

Will produce an HTML document that looks like this:

http://skitch.com/aaron.patterson/rgsq5/class-foo

RDoc doesn't crash or anything, but the output looks bad.

If we can add this support to RDoc, committers will not need to learn a
new format. But if they choose to use the new syntax, Ruby
documentation that is output via YARD will have more functionality.

--
Aaron Patterson
http://tenderlovemaking.com/

Attachment: (unnamed)
=end

#3 - 12/28/2010 05:52 AM - zenspider (Ryan Davis)

=begin

On Dec 27, 2010, at 12:20 , Aaron Patterson wrote:

> On Mon, Dec 27, 2010 at 12:01:00PM +0900, Yui NARUSE wrote:
>
> > Issue #4211 has been updated by Yui NARUSE.
> >
> > Your proposal is interesting, but as a lazy comitter, I want to know what we must do for this change.
> > For example:
> >
> > - Committers must learn YARD syntax (what we should read?)
>
> http://rubydoc.info/docs/yard/file/docs/GettingStarted.md
>
> > - convert existing documents (who will do?)
> > - fix some tools
> > - fix build process
>
> I think if we do this correctly, conversion may be done slowly without
> modifications of our tools.
>
> > I also want each vision for current use cases, like bin/rdoc and bin/ri.
> >
> > After all work is calculated and it is reasonable, I won't object the migration.
>
> I spoke with Loren about this issue on IRC. I think we can accomplish
> his goals by adding @tag support to RDoc.
>
> Adding @tag support to RDoc would allow us to unify Doxygen and RDoc
> documentation formats. But this means we must define the meaning of
> "@tag support".
>
> Currently, RDoc will ignore @tags in your documentation. For example,
> this code:
>
> class Foo
> # Converts the object into textual markup given a specific format
> # (defaults to :html)
> #
> # == Parameters:
> # format::
> #   A Symbol declaring the format to convert the object to. This
> #   can be :text or :html.
> #
> # == Returns:
> # A string representing the object in a specified
> # format.
> #
> def to_format(format = :html)
> # format the object
> end
> end
>
> Will produce an HTML document that looks like this:
>
> http://skitch.com/aaron.patterson/rgsq5/class-foo

RDoc doesn't crash or anything, but the output looks bad.

??? How does the above rdoc output @params ?

=end

#4 - 12/28/2010 07:03 AM - drbrain (Eric Hodel)

=begin
On Dec 26, 2010, at 13:00, Loren Segal wrote:

Feature #4211: Converting the Ruby and C API documentation to YARD syntax
http://redmine.ruby-lang.org/issues/show/4211

Author: Loren Segal
Status: Open, Priority: Normal
Category: DOC

The Ruby (high level core/stdlib) documentation and its C API (low level) counterparts currently use two different formats (and tools) to write and generate the final docs. This creates a problem for committers and users alike, where:

- Documentation is hard to write, because there is no single documentation style to follow (it depends on the API), and the two existing syntaxes are very different.
- Documentation is harder to read because the style and formatting differ due to the lack of consistent enforcement of a single style.
- Documentation for the C API (specifically) is harder to find

Currently, Doxygen @tag style syntax is slowly being introduced to improve the documentation of Ruby's C API, but this does not solve the three issues noted above. I propose to unify the documentation style used in the codebase to a single format (originally on ruby-core:33883[1]) by using YARD[2] syntax, which is very much like the Doxygen @tag style syntax being introduced anyway. Switching to YARD introduces a number of benefits, namely:

- There would be a single syntax to learn for committers wishing to document code, making it easier to write,
- The documentation would be formatted and styled consistently across both APIs for users to read,
- Documentation would be generated by a single tool for both APIs, meaning a simpler workflow for documenters and users wishing to generate the docs themselves.

I pointed out in the original mailing list that much of the documentation problems come from a lack of unified styling, causing parts of documentation to be (or become) inaccurate due to a variety of "human-error" type issues, and because there are no tools to check the correctness. I believe switching to a unified style and making sure it is used consistently will solve many of those issues even without tooling, because it is easier to manually check for errors with a consistent formatting. Furthermore, using a consistent style allows us to take advantage of our tooling to check basic correctness (or "lint") the docs for simple errors. YARD already has tools to do this kind of thing (and they are easily improved), but they depend on that consistent syntax.

As far as the C API goes, there is little difference in the existing doxygen syntax (except that I'd suggest the '@tag' instead of Doxygen's alternative '\tag' prefixes, for compatibility). As I wrote in the above ruby-core thread, YARD can already handle most of the written doxygen documentation. Granted, a lot of the support for actually *generating* documentation for a straight "C" style API is missing in YARD, but as I mentioned, I would be willing to improve this support if there is a willingness by the ruby-core developers to create a unified documentation style.

I highly support having a unified documentation tool and style for Ruby, but I personally dislike an enforced template like doxygen requires.

I find the doxygen-style template leads to stilted wording and low-quality documentation as the author merely has to check all the boxes to consider their work "complete". I've read documentation generated by such tools many times and have always found them difficult to navigate as there is not enough context to understand the interaction between all the components of an API.

If ruby adopts a documentation minimum standard I don't think an enforced template should be our only requirement.

h3. Steps forward:

We should first discuss whether the Ruby core developers are in favor of such a change. In the event that they are, we would have to look at a few things:

- Maintaining compatibility with RDoc (or adding YARD's @tag style support to RDoc) for the high level Ruby API docs while converting the syntax. I have a few ideas on how this can be done.
- Improving YARD's ability to generate HTML for straight "C" codebases (which I can implement, if we get this far)
- Any other issues / reservations raised by the Ruby core team

This is certainly a large proposal, and has some compatibility snags (with RDoc, for instance). Regardless, I think these issues can be worked around or dealt with for the most part, and the benefits of much improved documentation, which Ruby really needs, are certainly worth the effort.

I question merging YARD into Ruby and replacing RDoc.

Why aren't extensions or enhancements to RDoc being made instead?

Based on download numbers from rubygems.org it appears that a minority (about 25%) of rubyists prefer YARD to rdoc. There have been 8300 downloads of RDoc's latest release (1100/day over the last 90 days) compared to 1200 for YARD (285/day). Since March 2010 when the ri data format changed there have been 12000 downloads of rdoc-data which is 20% of YARD's total downloads since March 2007 of 60000.

If every YARD downloader prefers doxygen-style syntax, for me, the numbers imply that extensions should be made to RDoc to support different types of syntax over replacing RDoc.

RDoc has several APIs for adding such extensions, including plugins allowing alternate generators and additional directives:

http://rdoc.rubyforge.org/RDoc/Generator.html
http://rdoc.rubyforge.org/RDoc/RDoc.html
http://rdoc.rubyforge.org/RDoc/Markup/PreProcess.html#method-c-register

(I do not know if the APIs are sufficient as I've never received any communication from anyone attempting to use them for this task. I would love some feedback on this matter.)

Finally, I'm unsure why you've never emailed me or otherwise contacted me to propose adding extensions to RDoc to support YARD-style comments. I am not opposed to an extension that would support it.

If ruby-core preferred such an extension to RDoc over doxygen I would be happy to have it as part of RDoc.

=end

#5 - 12/28/2010 08:10 AM - tenderlovemaking (Aaron Patterson)

=begin
On Tue, Dec 28, 2010 at 05:52:13AM +0900, Ryan Davis wrote:

> On Dec 27, 2010, at 12:20 , Aaron Patterson wrote:
>
> > On Mon, Dec 27, 2010 at 12:01:00PM +0900, Yui NARUSE wrote:
> >
> > > Issue #4211 has been updated by Yui NARUSE.
> > >
> > > Your proposal is interesting, but as a lazy comitter, I want to know what we must do for this change.
> > > For example:
> > >
> > > - Committers must learn YARD syntax (what we should read?)
> >
> > http://rubydoc.info/docs/yard/file/docs/GettingStarted.md
> >
> > > - convert existing documents (who will do?)
> > > - fix some tools
> > > - fix build process
> >
> > I think if we do this correctly, conversion may be done slowly without
> > modifications of our tools.
> >
> > > I also want each vision for current use cases, like bin/rdoc and bin/ri.
> > >
> > > After all work is calculated and it is reasonable, I won't object the migration.
> >
> > I spoke with Loren about this issue on IRC. I think we can accomplish
> > his goals by adding @tag support to RDoc.
> >
> > Adding @tag support to RDoc would allow us to unify Doxygen and RDoc
> > documentation formats. But this means we must define the meaning of
> > "@tag support".
> >
> > Currently, RDoc will ignore @tags in your documentation. For example,
> > this code:
> >
> > class Foo
> > # Converts the object into textual markup given a specific format
> > # (defaults to :html)
> > #
> > # == Parameters:
> > # format::
> > # A Symbol declaring the format to convert the object to. This
> > # can be :text or :html.
> > #
> > # == Returns:

```
# A string representing the object in a specified
# format.
#
def to_format(format = :html)
# format the object
end
end
```

Will produce an HTML document that looks like this:

http://skitch.com/aaron.patterson/rgsq5/class-foo

RDoc doesn't crash or anything, but the output looks bad.

??? How does the above rdoc output @params ?

Sorry, I totally suck. The example code should be this:

```
class Foo
  # Converts the object into textual markup given a specific format.
  #
  # @param [Symbol] format the format type, `:text` or `:html`
  # @return [String] the object converted into the expected format.
  def to_format(format = :html)
    # format the object
  end
end
```

I stole the example from the YARD docs:

http://rubydoc.info/docs/yard/file/docs/GettingStarted.md

--
Aaron Patterson
http://tenderlovemaking.com/

Attachment: (unnamed)
=end

#6 - 12/28/2010 09:54 AM - lsegal (Loren Segal)

=begin

On 12/26/2010 10:01 PM, Yui NARUSE wrote:

Issue #4211 has been updated by Yui NARUSE.

Your proposal is interesting, but as a lazy comitter, I want to know what we must do for this change.
For example:

- Committers must learn YARD syntax (what we should read?)

http://yardoc.org has guides among the one linked by Aaron. The "Getting Started" guide is a good place to look at, though.

- convert existing documents (who will do?)

I'd certainly be interested in helping convert the existing documentation over to the new format. Some (not exactly sure how much) conversion can be done automatically, getting us part of the way there. A few manual conversions can be done afterwards to improve the overall quality. After an initial conversion, the tooling YARD has built-in can certainly help to narrow down what documentation needs work.

- fix some tools
- fix build process

YARD (the tool) could help here to add some sanity checks to the build process and ensure that new additions are well documented. Of course, the process would mostly be derived around a formulated documentation style, which I think should be the first step before starting to move things around. Discussing things like how to format return information, what style of language should be used, minimum requirements about documenting parameters, type information, etc.. Once these guidelines are set, we have something concrete to work with, and it should be fairly easy to apply these rules to start improving docs. I have suggestions about optimal ways to do things here too.

I also want each vision for current use cases, like bin/rdoc and bin/ri.

Ultimately, it would be great for RDoc to begin to support the @tag syntax used to document the Ruby end of the documentation. If this is done, we can have the same RDoc support in Ruby and have it continue to document the Ruby side of things and stay in core, while YARD can be used to generate docs for both APIs for those interested in such generation and tooling support.

- Loren

On 12/26/2010 10:01 PM, Yui NARUSE wrote:

```
Issue #4211 has been updated by Yui NARUSE.


Your proposal is interesting, but as a lazy comitter, I want to know what we must do for this change.
For example:
```

- Committers must learn YARD syntax (what we should read?)

http://yardoc.org has guides among the one linked by Aaron. The "Getting Started" guide is a good place to look at, though.

```
* convert existing documents (who will do?)
```

I'd certainly be interested in helping convert the existing documentation over to the new format. Some (not exactly sure how much) conversion can be done automatically, getting us part of the way there. A few manual conversions can be done afterwards to improve the overall quality. After an initial conversion, the tooling YARD has built-in can certainly help to narrow down what documentation needs work.

```
* fix some tools
 * fix build process
```

YARD (the tool) could help here to add some sanity checks to the build process and ensure that new additions are well documented. Of course, the process would mostly be derived around a formulated documentation style, which I think should be the first step before starting to move things around. Discussing things like how to format return information, what style of language should be used, minimum requirements about documenting parameters, type information, etc.. Once these guidelines are set, we have something concrete to work with, and it should be fairly easy to apply these rules to start improving docs. I have suggestions about optimal ways to do things here too.

```
I also want each vision for current use cases, like bin/rdoc and bin/ri.
```

Ultimately, it would be great for RDoc to begin to support the @tag syntax used to document the Ruby end of the documentation. if this is done, we can have the same RDoc support in Ruby and have it continue to document the Ruby side of things and stay in core, while YARD can be used to generate docs for both APIs for those interested in such generation and tooling support.

- Loren

=end

#7 - 12/28/2010 12:06 PM - lsegal (Loren Segal)

=begin

On 12/27/2010 5:03 PM, Eric Hodel wrote:

> I highly support having a unified documentation tool and style for Ruby, but I personally dislike an enforced template like doxygen requires.

What do you mean by an "enforced template"? Doxygen (or an alternative @tag based syntax) doesn't "require" anything from you, you're free to add the information you choose, just as with any documentation. @tag syntaxes are merely an *equivalent* to the existing documentation. The requirements come from a style guide, and that's part of documentation process, not some tool. However, since the Ruby C API is already starting to

make use of @tag syntax with Doxygen, there is little reason not to leverage this syntax as part of the unified style. Furthermore, YARD already supports this syntax. I'd have no problems with RDoc adding support, too. I don't know of any tools that can document the C end of things, unless RDoc plans on adding support for the C APIs (which I guess I would not object to).

> I find the doxygen-style template leads to stilted wording and low-quality documentation as the author merely has to check all the boxes to consider their work "complete".

I'm not sure what documentation you're basing this on, so I'll simply say this: the quality of any documentation ultimately depends on the effort that is put into writing it. People who "check all the boxes" are going to write poor documentation in any form, be it tag based or any other form. I've seen plenty of *good* documentation using tag based syntax, and I'm sure we can share plenty more anecdotes. One thing I know, though, of the *well documented* code I know, almost all of them use tag based syntax-- this includes Java's stdlib, Python's stdlib, Cocoa's APIS, many well documented JS libs like jQuery, and more. You can interpret this as you wish, but I see this as a positive trend.

> I've read documentation generated by such tools many times and have always found them difficult to navigate as there is not enough context to understand the interaction between all the components of an API.

What does RDoc do better, here? Again, this is not the fault of @tag based syntaxes-- it is the fault of the tools. For instance, doxygen has some rough edges in displaying documentation, but this is the fault of the tool, not the use of @tags. Keep in mind that I'm not proposing a tool, just a *unified documentation style*. RDoc could certainly implement this if you think it does the best job at displaying information to a user.

> If ruby adopts a documentation minimum standard I don't think an enforced template should be our only requirement.

I agree. As mentioned, the goal is to supply a unified *style*, not just a "template". @tag based syntax is merely a part of this style, since it's already being used in the C API, and there is no alternative to properly documenting the C end of things than @tag based tools. It therefore makes sense to continue in this vein and unify the documentation in this way.

However, it's important to realize that supplying a "bare minimum" guideline to a documenter does a lot to improve the quality of documentation. Obviously, manual checks would have to be done to make sure people are not just "checking off boxes", but having those tags there lets us know what to look for when reviewing documentation, and lets *them* know what they should be writing. This is an improvement over the current adhoc "just-write-something" methodology. Tags also give us the ability to implement tooling later, because the docs are much more easily parse-able.

> I question merging YARD into Ruby and replacing RDoc.
>
> Why aren't extensions or enhancements to RDoc being made instead?

YARD has fairly significant architectural differences that don't lend well to simply writing extensions. YARD is actually a superset of RDoc, and not the other way around, so a merge of functionality wouldn't really work. For instance, YARD is not only a "parse and generate" tool. It is also often used as a 2 phase: "parse and store", "generate from stored data" tool (as exemplified by the 'yard server' tool). RDoc does not have this phase, so much of the functionality YARD has could not be implemented easily. YARD's parsing architecture also allows much more leeway to customizing the parsing phase than RDoc's limited markup preprocessing APIs. And yes, these APIs are used in at least a few plugins I know of.

> Based on download numbers from rubygems.org it appears that a minority (about 25%) of rubyists prefer YARD to rdoc.

I'm not sure it's exactly fair to correlate popularity with preference here. PHP is probably downloaded more than Ruby, but I really don't believe programmers prefer PHP to Ruby.

Certainly RDoc has more users, as more people know about RDoc's existence than YARD, since it comes packaged with Ruby. In addition, most users simply don't have the need for YARD's extra features, so RDoc is "good enough". I highly doubt any of this correlates with "preference", though. The http://rubydoc.info project Nick Plante and I have been running for the last little while has been greatly successful and has been receiving quite a lot of traffic and good feedback from users. Given more even exposure to both tools, I'd bet the real numbers would be fairly different.

> RDoc has several APIs for adding such extensions, including plugins allowing alternate generators and additional directives:
>
> http://rdoc.rubyforge.org/RDoc/Generator.html
> http://rdoc.rubyforge.org/RDoc/RDoc.html
> http://rdoc.rubyforge.org/RDoc/Markup/PreProcess.html#method-c-register
>
> (I do not know if the APIs are sufficient as I've never received any communication from anyone attempting to use them for this task. I would love some feedback on this matter.)
> It's great that RDoc has these APIs. I've never used any of them myself, so I can't comment on their usefulness (at first glance they look good). However, a few questions/notes:

- How is support for new constructs (DSLs) added in Ruby code? This one isn't very specific to this discussion, since Ruby core code doesn't use any non-standard DSLs, but I've found that this is one of the most powerful YARD APIs.
- Are there any ways to hook into the information generated by directives *after* they've been declared? For instance, can you check for the existence of a specific directive inside of a custom Generator? Again, this kind of functionality is used extremely often in YARD, and is extremely important to outputting documentation for arbitrary @tag-style metadata.
- Can the preprocessing API handle blocks of text (ones including newlines)? Certain tags require this (@example, for instance).

> Finally, I'm unsure why you've never emailed me or otherwise contacted me to propose adding extensions to RDoc to support YARD-style comments. I am not opposed to an extension that would support it.

This is the first time I've proposed creating a unified style. This feature came at the request of Yugui to have a formalized discussion of the proposal I made in the ruby-core list linked above. It originally had nothing to do with RDoc specifically (and still doesn't). I have no problem adding this support to RDoc (in fact I suggested it in the previous email). I don't care which tool is ultimately used for generation, be it RDoc or YARD. However, would RDoc be generating documentation for the C API?

- Loren

<title></title>

On 12/27/2010 5:03 PM, Eric Hodel wrote:

> I highly support having a unified documentation tool and style for Ruby, but I personally dislike an enforced template like doxygen requires.

What do you mean by an "enforced template"? Doxygen (or an alternative @tag based syntax) doesn't "require" anything from you, you're free to add the information you choose, just as with any documentation. @tag syntaxes are merely an *equivalent* to the existing documentation. The requirements come from a style guide, and that's part of documentation process, not some tool. However, since the Ruby C API is already starting to make use of @tag syntax with Doxygen, there is little reason not to leverage this syntax as part of the unified style. Furthermore, YARD already supports this syntax. I'd have no problems with RDoc adding support, too. I don't know of any tools that can document the C end of things, unless RDoc plans on adding support for the C APIs (which I guess I would not object to).

> I find the doxygen-style template leads to stilted wording and low-quality documentation as the author merely has to check all the boxes to consider their work "complete".

I'm not sure what documentation you're basing this on, so I'll simply say this: the quality of any documentation ultimately depends on the effort that is put into writing it. People who "check all the boxes" are going to write poor documentation in any form, be it tag based or any other form. I've seen plenty of *good* documentation using tag based syntax, and I'm sure we can share plenty more anecdotes. One thing I know, though, of the *well documented* code I know, almost all of them use tag based syntax-- this includes Java's stdlib, Python's stdlib, Cocoa's APIS, many well documented JS libs like jQuery, and more. You can interpret this as you wish, but I see this as a positive trend.

> I've read documentation generated by such tools many times and have always found them difficult to navigate as there is not enough context to understand the interaction between all the components of an API.

What does RDoc do better, here? Again, this is not the fault of @tag based syntaxes-- it is the fault of the tools. For instance, doxygen has some rough edges in displaying documentation, but this is the fault of the tool, not the use of @tags. Keep in mind that I'm not proposing a tool, just a *unified documentation style*. RDoc could certainly implement this if you think it does the best job at displaying information to a user.

> If ruby adopts a documentation minimum standard I don't think an enforced template should be our only requirement.

I agree. As mentioned, the goal is to supply a unified *style*, not just a "template". @tag based syntax is merely a part of this style, since it's already being used in the C API, and there is no alternative to properly documenting the C end of things than @tag based tools. It therefore makes sense to continue in this vein and unify the documentation in this way.

However, it's important to realize that supplying a "bare minimum" guideline to a documenter does a lot to improve the quality of documentation. Obviously, manual checks would have to be done to make sure people are not just "checking off boxes", but having those tags there lets us know what to look for when reviewing documentation, and lets *them* know what they should be writing. This is an improvement over the current adhoc "just-write-something" methodology. Tags also give us the ability to implement tooling later, because the docs are much more easily parse-able.

> I question merging YARD into Ruby and replacing RDoc.

```
Why aren't extensions or enhancements to RDoc being made instead?
```

YARD has fairly significant architectural differences that don't lend well to simply writing extensions. YARD is actually a superset of RDoc, and not the other way around, so a merge of functionality wouldn't really work. For instance, YARD is not only a "parse and generate" tool. It is also often used as a 2 phase: "parse and store", "generate from stored data" tool (as exemplified by the 'yard server' tool). RDoc does not have this phase, so much of the functionality YARD has could not be implemented easily. YARD's parsing architecture also allows much more leeway to customizing the parsing phase than RDoc's limited markup preprocessing APIs. And yes, these APIs are used in at least a few plugins I know of.

```
Based on download numbers from rubygems.org
 it appears that a minority (about 25%) of rubyists prefer YARD to rdoc.
```

I'm not sure it's exactly fair to correlate popularity with preference here. PHP is probably downloaded more than Ruby, but I really don't believe programmers prefer PHP to Ruby.

Certainly RDoc has more users, as more people know about RDoc's existence than YARD, since it comes packaged with Ruby. In addition, most users simply don't have the need for YARD's extra features, so RDoc is "good enough". I highly doubt any of this correlates with "preference", though. The http://rubydoc.info project Nick Plante and I have been running for the last little while has been greatly successful and has been receiving quite a lot of traffic and good feedback from users. Given more even exposure to both tools, I'd bet the real numbers would be fairly different.

```
RDoc has several APIs for adding such extensions, including plugins allowing alternate generators and addi
tional directives:

http://rdoc.rubyforge.org/RDoc/Generator.html
http://rdoc.rubyforge.org/RDoc/RDoc.html
http://rdoc.rubyforge.org/RDoc/Markup/PreProcess.html#method-c-register

(I do not know if the APIs are sufficient as I've never received any communication from anyone attempting
to use them for this task.  I would love some feedback on this matter.)
```

It's great that RDoc has these APIs. I've never used any of them myself, so I can't comment on their usefulness (at first glance they look good). However, a few questions/notes:

- How is support for new constructs (DSLs) added in Ruby code? This one isn't very specific to this discussion, since Ruby core code doesn't use any non-standard DSLs, but I've found that this is one of the most powerful YARD APIs.

- Are there any ways to hook into the information generated by directives *after* they've been declared? For instance, can you check for the existence of a specific directive inside of a custom Generator? Again, this kind of functionality is used extremely often in YARD, and is extremely important to outputting documentation for arbitrary @tag-style metadata.

- Can the preprocessing API handle blocks of text (ones including newlines)? Certain tags require this (@example, for instance).

```
Finally, I'm unsure why you've never emailed me or otherwise contacted me to propose adding extensions to
RDoc to support YARD-style comments.  I am not opposed to an extension that would support it.
```

This is the first time I've proposed creating a unified style. This feature came at the request of Yugui to have a formalized discussion of the proposal I made in the ruby-core list linked above. It originally had nothing to do with RDoc specifically (and still doesn't). I have no problem adding this support

to RDoc (in fact I suggested it in the previous email). I don't care which tool is ultimately used for generation, be it RDoc or YARD. However, would RDoc be generating documentation for the C API?

- Loren

=end

#8 - 12/29/2010 10:18 AM - drbrain (Eric Hodel)

=begin
On Dec 27, 2010, at 19:03, Loren Segal wrote:

> On 12/27/2010 5:03 PM, Eric Hodel wrote:
>
> > I highly support having a unified documentation tool and style for Ruby, but I personally dislike an enforced template like doxygen requires.
>
> What do you mean by an "enforced template"?

If we're building a documentation standard for ruby we'll need an example of good documentation, a "template" which we will enforce when documentation is submitted.

> Doxygen (or an alternative @tag based syntax) doesn't "require" anything from you, you're free to add the information you choose, just as with any documentation. @tag syntaxes are merely an *equivalent* to the existing documentation. The requirements come from a style guide, and that's part of documentation process, not some tool. However, since the Ruby C API is already starting to make use of @tag syntax with Doxygen, there is little reason not to leverage this syntax as part of the unified style. Furthermore, YARD already supports this syntax. I'd have no problems with RDoc adding support, too. I don't know of any tools that can document the C end of things, unless RDoc plans on adding support for the C APIs (which I guess I would not object to).

It would not be difficult to add to RDoc.

> > I've read documentation generated by such tools many times and have always found them difficult to navigate as there is not enough context to understand the interaction between all the components of an API.
>
> What does RDoc do better, here? Again, this is not the fault of @tag based syntaxes-- it is the fault of the tools. For instance, doxygen has some rough edges in displaying documentation, but this is the fault of the tool, not the use of @tags. Keep in mind that I'm not proposing a tool, just a *unified documentation style*. RDoc could certainly implement this if you think it does the best job at displaying information to a user.

Good documentation is independent of the tools used to process it or the syntax used to write it. I don't think a different tool can make better documentation without human intervention. Some characteristics of good documentation are consistent wording, approachable writing, comprehensive coverage and useable examples. I'm unsure of how a tool can help with anything other than comprehensive coverage.

> > If ruby adopts a documentation minimum standard I don't think an enforced template should be our only requirement.
>
> I agree. As mentioned, the goal is to supply a unified *style*, not just a "template". @tag based syntax is merely a part of this style,

We are in agreement here.

> since it's already being used in the C API, and there is no alternative to properly documenting the C end of things than @tag based tools. It therefore makes sense to continue in this vein and unify the documentation in this way.

Granted.

> However, it's important to realize that supplying a "bare minimum" guideline to a documenter does a lot to improve the quality of documentation. Obviously, manual checks would have to be done to make sure people are not just "checking off boxes", but having those tags there lets us know what to look for when reviewing documentation, and lets *them* know what they should be writing. This is an improvement over the current adhoc "just-write-something" methodology. Tags also give us the ability to implement tooling later, because the docs are much more easily parse-able.

A common feature of documentation in ruby core is to "tag" method parameters in ++, or . Matching the "tagged" items in a comment to the method's arguments is trivial way to implement an enhanced coverage report without requiring @tags.

One of the greatest joys I've found in programming is the ability to have the computer do work for me. I would prefer to have a tool that could figure out what I haven't documented, or documented well enough, with the fewest possible concessions.

> I question merging YARD into Ruby and replacing RDoc.
>
> Why aren't extensions or enhancements to RDoc being made instead?

YARD has fairly significant architectural differences that don't lend well to simply writing extensions. YARD is actually a superset of RDoc, and not the other way around, so a merge of functionality wouldn't really work. For instance, YARD is not only a "parse and generate" tool. It is also often used as a 2 phase: "parse and store", "generate from stored data" tool (as exemplified by the 'yard server' tool).

Sounds like ri -f html. ri uses RDoc::RI::Store to load data from a stored documentation tree.

> RDoc does not have this phase, so much of the functionality YARD has could not be implemented easily.

Loading an RDoc::RI::Store and running it back through an HTML generator could be implemented very easily. The HTML generator may need to be adapted to handle pre-parsed comments which would involve adding two lines.

> YARD's parsing architecture also allows much more leeway to customizing the parsing phase than RDoc's limited markup preprocessing APIs. And yes, these APIs are used in at least a few plugins I know of.
>
> > RDoc has several APIs for adding such extensions, including plugins allowing alternate generators and additional directives:
> >
> > http://rdoc.rubyforge.org/RDoc/Generator.html
> > http://rdoc.rubyforge.org/RDoc/RDoc.html
> > http://rdoc.rubyforge.org/RDoc/Markup/PreProcess.html#method-c-register
> >
> > (I do not know if the APIs are sufficient as I've never received any communication from anyone attempting to use them for this task. I would love some feedback on this matter.)
>
> It's great that RDoc has these APIs. I've never used any of them myself, so I can't comment on their usefulness (at first glance they look good). However, a few questions/notes:
>
> - How is support for new constructs (DSLs) added in Ruby code? This one isn't very specific to this discussion, since Ruby core code doesn't use any non-standard DSLs, but I've found that this is one of the most powerful YARD APIs.

I built a proof-of-concept parser for Rakefiles:

https://github.com/rdoc/rdoc-rake

Since I am still supporting Ruby 1.8.7 I haven't been able to drop the irb-based pure-ruby parser.

Additionally there's Hugh Sasse's perl/pod parser:

https://github.com/rdoc/rdoc-perl_pod

And a fortran parser.

All these parsers build an RDoc::CodeObject tree which the generator handles transparently. rdoc-rake subclasses RDoc::Include, RDoc::NormalModule and RDoc::AnyMethod to have a better-looking implementation.

> - Are there any ways to hook into the information generated by directives *after* they've been declared? For instance, can you check for the existence of a specific directive inside of a custom Generator? Again, this kind of functionality is used extremely often in YARD, and is extremely important to outputting documentation for arbitrary @tag-style metadata.

There is no restriction. If a directive is unhandled it's stored in the code object's metadata hash.

> - Can the preprocessing API handle blocks of text (ones including newlines)? Certain tags require this (@example, for instance).

Currently, no. Calling a registered handler with a third parameter (the comment's text) could add this functionality.

> Finally, I'm unsure why you've never emailed me or otherwise contacted me to propose adding extensions to RDoc to support YARD-style comments. I am not opposed to an extension that would support it.

This is the first time I've proposed creating a unified style. This feature came at the request of Yugui to have a formalized discussion of the proposal I made in the ruby-core list linked above. It originally had nothing to do with RDoc specifically (and still doesn't). I have no problem adding this support to RDoc (in fact I suggested it in the previous email). I don't care which tool is ultimately used for generation, be it RDoc or

> YARD. However, would RDoc be generating documentation for the C API?

It wouldn't be much work to add, perhaps a week to fully polish. My boss says I can work on this in place of company business, so I'll give it a shot.

=end

#9 - 12/30/2010 01:33 PM - lsegal (Loren Segal)

=begin

On 12/28/2010 8:18 PM, Eric Hodel wrote:

> If we're building a documentation standard for ruby we'll need an example of good documentation, a "template" which we will enforce when documentation is submitted.

For the most part I agree. An example will help validate a proposed "template". jQuery and Cocoa's APIs are good, but I also know of a few Ruby libraries using (YARD's) @tag syntax if we're looking for something closer to home. Any suggestions from your end?

> > I've read documentation generated by such tools many times and have always found them difficult to navigate as there is not enough context to understand the interaction between all the components of an API.
> > What does RDoc do better, here? Again, this is not the fault of @tag based syntaxes-- it is the fault of the tools. For instance, doxygen has some rough edges in displaying documentation, but this is the fault of the tool, not the use of @tags. Keep in mind that I'm not proposing a tool, just a *unified documentation style*. RDoc could certainly implement this if you think it does the best job at displaying information to a user.
> > Good documentation is independent of the tools used to process it or the syntax used to write it. I don't think a different tool can make better documentation without human intervention. Some characteristics of good documentation are consistent wording, approachable writing, comprehensive coverage and useable examples. I'm unsure of how a tool can help with anything other than comprehensive coverage.

You're right about human intervention. The tools are basically there to help you sift through the data, bringing to light problem areas based on heuristics. I don't think this functionality should be underestimated, though. With so much documentation to sift through, it's useful to have some way to do some quick sanity checks-- ultimately this will come down to good manual checking of the raised issues.

Good tools can be used to, for instance, test that the examples provided in the docs actually work. This is kind of hard with high level APIs that depend on some initial state, but with Ruby's core classes where you can write isolated examples, this would be extremely useful and effective. Both tools and syntax can also be used to ensure you're specifying the right type information in your documentation. Syntax also especially helps to ensure consistent wording across documentation, because you end up with less of it, which means less to manually review.

A few examples I commonly come across:

- The common phrasing of "This method returns ...." becomes obsoleted by a single "@return ...". The wording of how "this method returns" is phrased, and *where* it can be found in the docstring is simplified so that it becomes really hard to screw up.

- Having the tag there means you can also check for the existence of that information much more easily. In docs without such a consistent syntax, you couldn't just check for the existence "Returns", as it's not guaranteed to exist. See Array#transpose for the first example I came across. Thankfully in this case there is at least a simple working example that summarizes the information. That is not always the case.

- Again, @return gives documentation a consistent form, and also gives users a consistent placement of that information. Misunderstanding Array#reject vs #reject! is a common "newbie mistake". Although #reject!'s documentation does say it returns nil, that information is often lost in the wording ("... but returns nil if..." is not exactly put front and center as it should be). A @tag based syntax would allow you to rewrite the #reject! method as:

Deletes all elements from *self* for which the given block

evaluates to +true+.
@yield [item] each item in the array
@return [Array] if at least one change was made
@return [nil] if nothing was changed

Having tags allows the tool to (more easily) display this information consistently and organize information for the reader by return type, rather than having the documentor arbitrarily decide the importance of a return type or general return information-- or having reviewers manually ensure that "return information is displayed at the ____ of a docstring". The organization of the data should be provided by the tool, which makes up a big part of the overall documentation quality. Using a tool that provides poor organizational flow can be just as bad as having poorly written documentation-- you pointed this out above, too.

- I'll throw this in there: YARD uses a convention return type "void" (@return [void]), which denotes that although Ruby always returns "a value", the value being returned from the method is effectively "meaningless" and equivalent to void. There are a small batch of methods in the Ruby docs that have this kind of behaviour, but this happens a lot in higher level APIs. Usually people ignore making any mention of such a "meaningless" return value, but this causes ambiguity because now the reader doesn't know when this information was omitted due to "meaninglessness", or carelessness by the documentor. Checking that a @return tag exists make sure that there is never such an ambiguity. And this is definitely something that syntax can give you.

Approachable writing, of course, can't be corrected by a tool. That one needs proper manual review. I would certainly not mind volunteering some of my time for such reviews.

> However, it's important to realize that supplying a "bare minimum" guideline to a documenter does a lot to improve the quality of documentation. Obviously, manual checks would have to be done to make sure people are not just "checking off boxes", but having those tags there lets us know what to look for when reviewing documentation, and lets *them* know what they should be writing. This is an improvement over the current adhoc "just-write-something" methodology. Tags also give us the ability to implement tooling later, because the docs are much more easily parse-able.
> A common feature of documentation in ruby core is to "tag" method parameters in ++, or. Matching the "tagged" items in a comment to the method's arguments is trivial way to implement an enhanced coverage report without requiring @tags.

I'm not sure I really see how this could be applied. The existence of a +parameter_name+ does not always imply that the quality is there. This is *also* true for a @param tag. On the flipside, the *lack* of a +parameter_name+ in the docs does *not* mean the argument was not documented. For instance, people often say "takes a string" instead of "takes string +str+", when there is only one argument (I think this is acceptable). However, if you enforce documentation for a parameter to be located inside a @param tag, you *can* guarantee that a missing @param tag means the parameter was not documented. You still end up with a few false negatives, where the parameter really needs no documentation, but these false negatives are a relative few to the non-tag alternative, IMO.

> One of the greatest joys I've found in programming is the ability to have the computer do work for me. I would prefer to have a tool that could figure out what I haven't documented, or documented well enough, with the fewest possible concessions.

Certainly. I don't believe @tags are concessions, though. If anything, there is less superfluous writing when using tags, because you can rely on the tool to expand language for you in a consistent fashion. YARD actually takes advantage of this rule: when given a @return tag and no extra docstring ('docstring' meaning plain "English" text, not tag metadata), YARD expands this out to a full English sentence in the method summary for you. You can rewrite the awkward:

```
# Returns a new string that is encoded using the ROT13 cipher
# @return [String] the rot13 version of the string
def rot13; ... end
```

Into a much more compact:

```
# @return [String] a new string that is encoded using the ROT13 cipher
def rot13; ... end
```

YARD handles the "Returns " for you in the summary, that is you still get the same plain English text as above, but also still organizes the return type information alongside the metadata below. I think there are a lot of places where tags can be leveraged in this way to make your life *easier*, not harder. This was certainly one of YARD's original design goals from the get-go. If I truly believed @tags were more cumbersome and less advantageous, I would not have used them.

> I question merging YARD into Ruby and replacing RDoc.
>
> Why aren't extensions or enhancements to RDoc being made instead?
> YARD has fairly significant architectural differences that don't lend well to simply writing extensions. YARD is actually a superset of RDoc, and not the other way around, so a merge of functionality wouldn't really work. For instance, YARD is not only a "parse and generate" tool. It is also often used as a 2 phase: "parse and store", "generate from stored data" tool (as exemplified by the 'yard server' tool).
> Sounds like ri -f html. ri uses RDoc::RI::Store to load data from a stored documentation tree.

Useful. It would be much better if the gem server leveraged this instead of forcing the static documentation to be generated on a gem install automatically. That is a painfully slow process. YARD only does a parse-thru on install (unless you tell it to generate static docs), and then dynamically generates the HTML when using yard server --gems. This shaves minutes off of some larger gem installs (like Rails), and is a much better alternative than adding --no-rdoc to your (which a lot of people are doing AFAIK). It also saves plenty of disk space, since few users actually access the static gem docs, and those that do can generate them on demand with gem doc anyway. BTW the YARD server actually can do the parsing dynamically too if it wasn't previously done, so you don't even need to do any parsing on install, which means even *less* time is spent installing. We still do the parse-run for our RI implementation though, because there's no way around pre-parsing for that.

> RDoc does not have this phase, so much of the functionality YARD has could not be implemented easily.
> Loading an RDoc::RI::Store and running it back through an HTML generator could be implemented very easily. The HTML generator may need to be adapted to handle pre-parsed comments which would involve adding two lines.

I think this would be a great move for the reasons given above.

> YARD's parsing architecture also allows much more leeway to customizing the parsing phase than RDoc's limited markup preprocessing APIs. And yes, these APIs are used in at least a few plugins I know of.

> I built a proof-of-concept parser for Rakefiles:
>
> https://github.com/rdoc/rdoc-rake

Seeing yours, I built a small proof of concept Rakefile parser in YARD this afternoon to exemplify the API differences:
http://gist.github.com/759428

It integrates into the existing template using the same method you are using (namespaces as Module, tasks as Method) and also lists dependencies of tasks as metadata tags. I fleshed out a more complete version that performs better customization to the template and lists the Rake tasks in the index page along with the readme files and objects: https://github.com/lsegal/yard-rake -- I just might introduce this into a future release if it gets a little more work. :-)

- Are there any ways to hook into the information generated by directives *after* they've been declared? For instance, can you check for the existence of a specific directive inside of a custom Generator? Again, this kind of functionality is used extremely often in YARD, and is extremely important to outputting documentation for arbitrary @tag-style metadata.
  There is no restriction. If a directive is unhandled it's stored in the code object's metadata hash.

That's good to know, and certainly useful.

> However, would RDoc be generating documentation for the C API?
> It wouldn't be much work to add, perhaps a week to fully polish. My boss says I can work on this in place of company business, so I'll give it a shot.

Great. If RDoc was generating the C API and handling the @tags, that
would help move things along.

- Loren

=end

#10 - 03/18/2012 06:49 PM - nahi (Hiroshi Nakamura)

- Description updated

- Assignee set to drbrain (Eric Hodel)

#11 - 03/31/2012 11:09 AM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

#12 - 11/20/2012 10:22 PM - mame (Yusuke Endoh)

- Target version set to 2.6

#13 - 04/28/2013 07:07 AM - zzak (zzak _)

- Status changed from Assigned to Rejected

I'm going to reject this, if you have a specific proposal how to improve RDoc please open a feature ticket.