Improved YARD CHEATSHEET

Forks

- Originally forked from here last update around 2012.
- Forked from here

Web Resources

- Official Getting Started Guide
- Official Tags documentation
- Type naming examples

Templates to remind you of the options and formatting for the different types of objects you might want to document using YARD.

Types

Type	Description
`[Foo, Bar]`	Foo or Bar
`Array<String>`	Array of string
`Array(String, Integer)`	Array of length 2: String followed by Integer
`Hash{KeyType => Book,Movie,Series<Thriller>}`	A Hash with keys of type `KeyType` and values of eithe
`#foo`	an object that responds to `foo`

Grammar

Method Parameters

```
@param OPTIONAL_NAME [TYPE] DESCRIPTION TEXT HERE
```

Options hashes

```
@option OPTIONS_HASH_NAME [TYPE] KEY_SYMBOL (DEFAULT) DESCRIPTION TEXT HERE
```

References

```
# Inline reference link
{SomeClass#method}

# Attribute-based reference
@see SomeClass#method

# Inline typewriter param reference
+my_parameter+
```

API Modifiers

```
@private

@abstract

@deprecated Use {#my_new_method} instead
  continued...

@since VERSION
```

Blocks

```
@yield [VAR_FOO, VAR_BAR, VAR_C] Description of block

@yieldparam argname [TYPE, TYPE, ...] description

@yieldreturn [TYPE, TYPE, ...] description
```

Meta Programming

```
@!attribute [r | w | rw] attribute_name
```

Methods

```
# An alias to {Parser::SourceParser}'s parsing method
#
# @author Donovan Bray
#
# @see http://example.com Description of URL
# @see SomeOtherClass#method
#
# @deprecated Use {#my_new_method} instead of this method because
#   it uses a library that is no longer supported in Ruby 1.9.
#   The new method accepts the same parameters.
#
# @abstract
# @private
```

Method parameters

```
# @param [Hash] opts the options to create a message with.
# @option opts [String] :subject The subject
# @option opts [String] :from ('nobody') From address
# @option opts [String] :to Recipient email
# @option opts [String] :body ('') The email's body
#
```

```
# @param (see User#initialize)
# @param [OptionParser] opts the option parser object
# @param [Array<String>] args the arguments passed from input. This
#   array will be modified.
# @param [Array<String, Symbol>] list the list of strings and symbols.
# @param [Hash<Symbol, String>] a hash with symbol keys and string values
#
# The options parsed out of the commandline.
# Default options are:
#   :format => :dot
```

Method Keyword paramters

From the documentation:

> For keyword parameters, use @param, not @option.

```
# @param name [String] The name of the person to sing for
def sing_for(name:)
  # ...
end
```

Variable number of Method Parameters

```
# As these are really accessed via an array in the method, use the same
# syntax as for arrays
#
# @param obj [Object] The object for which interface must be checked
# @param method_symbols [Array<Symbol>] Variable number of method names that must exist
#        for the object to pass the interface.
# def check_interface!(obj, *method_symbols)
# ...
# end
```

Multiple types

Multiple types are comma-seperated

```
# @return [Movie, Book]
def movies_and_books
  [Movie.new("300", "Zack Snyder"), Book.new("The Andromeda Strain", "Michael Crichton")]
end
```

Multiple Methods in Duck-Type

Specifying single duck-types is simple (see below) but specifying multiple methods is not yet idiomatically possible in YARD, but can be implemented according to

the author. Instead it's recommended that you specify a new type containing all your methods, even if you do not use it in the code.

```
# Parse the document from a string or an object that responds to +read+
# @param stream_or_string [String, #read]
def parse(stream_or_string)
...
end
```

Examples

```
# @example Reverse a string
#   "mystring".reverse #=> "gnirtsym"
#
# @example Parse a glob of files
#   YARD.parse('lib/**/*.rb')
```

Modules

```
# Namespace for classes and modules that handle serving documentation over HTTP
# @since 0.6.0
```

Classes

```
# Abstract base class for CLI utilities. Provides some helper methods for
# the option parser
#
# @author Full Name
# @abstract
# @since 0.6.0
# @deprecated Describe the reason or provide alt. references here
#
#
# If you generate attributes via meta programming, use
# @!attribute [r | w | rw] attribute_name
```

See https://www.rubydoc.info/gems/yard/file/docs/Tags.md#attribute for more information on documenting attributes.

Attributes

```
# Attributes can be documented directly like this
# @return [String]
attr_reader :hello
# **NOTE** the reader attribute should always carry the doc for its writer as well,
# for this we must use the overload notation
```

```
# @overload foo
#   Returns the value @foo
#   @return Foo
# @overload foo=(value)
#   @param value [Foo]
#   @note Something interesting about the behaviour of setting foo
attr_reader :foo

# ignored by yard
attr_writer :foo
```

Exceptions

```
# @raise [ExceptionClass] description
```

Return values

```
# @return [optional, types, ...] description
# @return [true] always returns true
# @return [void]
# @return [String, nil] the contents of our object or nil
#   if the object has not been filled with data.
#
# We don't care about the "type" here:
# @return the object
#
# @return [String, #read] a string or object that responds to #read
# @return description here with no types
```

Anywhere

```
# @todo Add support for Jabberwocky service
#   There is an open source Jabberwocky library available
#   at http://somesite.com that can be integrated easily
#   into the project.
```

Blocks

```
# for block {|a, b, c| ... }
# @yield [a, b, c] Description of block
#
# @yieldparam [optional, types, ...] argname description
# @yieldreturn [optional, types, ...] description
```

Miscellaneous

Linking to Objects

To link to another "object" (class, method, module, etc.), use the format:

```
# It's worth looking at the {Parser#parse parse method} as well as the constant
# {Parser::TOKEN_SPACE} to understand how things are split up. In this class, you
# may also find {#explain} interesting, as it explains the generated AST.
# You may also like to (see Interpreter) to understand how this language works.
```

Rendering Objects

This is more useful in an index page or tutorial than it is elsewhere

```
# The Movie class uses a simple decoder as can be seen below.
# {render:Movie#decode}
#
# The encoder is also pretty neat
# {render:Movie#encoder}
```