Ruby class_eval, __FILE__ and __LINE__ arguments

Here is an example of `class_eval` method call that we will discuss:

```
def my_attr_reader(sym)
  class_eval <<-READER, __FILE__ , __LINE__ + 1
    def #{sym}
      @#{sym}
    end
  READER
end
```

What `class_eval` method does and what is it used for? Why do we send file name and line number values to it? These are the questions that we will answer here.

`class_eval` evaluates a string or block in the context of the receiver **self**which can be a class or module. In the following example we are using`class_eval` to define an instance method in the class identified by **self**(`User`):

```
module MyAttrReader
  def my_attr_reader(sym)
    class_eval <<-READER, __FILE__ , __LINE__ + 1
      def #{sym}
        @#{sym}
      end
    READER
  end
end

class User
  extend MyAttrReader

  def initialize(name)
    @name = name
  end

  my_attr_reader :name
end

u = User.new('dalibor')
p u.name # calls the method 'name' defined with class_eval
```

Let's now explain the pseudo variables that are used:

```
__FILE__ # the current source file name.
__LINE__ # the current line number in the source file.
```

We can simplify the `class_eval` call by removing the here document:

```
def my_attr_reader(sym)
  class_eval("def #{sym}; undefined_method; @#{sym}; end;", __FILE__ , __LINE__ )
end
```

Now when the syntax is clear, if we check the documentation forclass_eval we can see that second and third arguments are used to set the text for error message which will help us locate where the error has occurred.

Look at the errors produced by the following 2 code snippets. Notice that we added `undefined_method` in the string evaluated by `class_eval` so that it produces an error and we removed the file and line pseudo variables from the first snippet:

```
module MyAttrReader
  def my_attr_reader(sym)
    class_eval "def #{sym}; undefined_method; @#{sym}; end;"
  end
end

class User
  extend MyAttrReader

  def initialize(name)
    @name = name
  end

  my_attr_reader :name
end

u = User.new('dalibor')
p u.name

# (eval):1:in `name': undefined local variable or method `undefined_method' for #
(NameError)


module MyAttrReader
  def my_attr_reader(sym)
    class_eval "def #{sym}; undefined_method; @#{sym}; end;", __FILE__ , __LINE__
  end
end

class User
```

```
  extend MyAttrReader

  def initialize(name)
    @name = name
  end

  my_attr_reader :name
end

u = User.new('dalibor')
p u.name

# t.rb:3:in `name': undefined local variable or method `undefined_method' for # (NameError)
```

Error messages are displayed at the bottom of the code snippets.

You can notice that the second error message is clear, telling us that the error is on file line 3, but the first error message doesn't tell us the file line where the error has occurred.