

# RubyMine 1.1.1 internal ruby API tips

---



[Roman.Chernyatchik](#) 1,452 posts since

Sep 18, 2007

Ok, let's start =)

Our ruby API isn't ready yet and planned on RubyMine 1.5 but some features are available right now. I think that API creation is iterative process so this post will be our first step =)  
We will appreciate if somebody try to play with it and provide us any feedback. Also this post isn't supposed to be considered as official documentation of RubyMine API, just several tips for enthusiasts.

RubyMine allows to write extensions using jruby. Our java API isn't documented yet and I will not consider it in this post.

You can find our ruby part of RubyMine at **\$RubyMine\_installation\_dir\$/rb**. E.g. by default

\* On MacOS: /Applications/RubyMine 1.1.1.app/rb

\* On Windows: C:\Program files\RubyMine 1.1.1\rb

**rb** folder contains three important subfolders:

\* api

\* scripts

\* paramdefs

On startup RubyMine will load all files from **rb/scripts** directory. So :

Rule 1. if you want to write/fix any RubyMine extension than at first look at rb/scripts folder.

Rule 2. Your changes in this folder will be lost if you delete/uninstall RubyMine, or probably update RubyMine 1.1.1 using auto update.

As I mentioned this isn't even beta quality API. Obviously in next versions we will allow users to store their scripts with ruby extensions outside RubyMine installation folder.

## Code insight for values of rails calls (e.g. render, url\_for, validate, ...)

I suppose this is the most completed part of API. Almost all code insight of values for rails call's parameters is written using this API.

**Example:** Let's imagine that you implemented your own "render" method with name "ActionController::Base.my\_render" and you want enable the same support as RubyMine has for original ActionController::Base.render method. So you should

1. open: **rb/paramdefs/rails/paramdefs\_loader.rb.**
2. find code

```
paramdef 'ActionController::Base', ['render', 'render_to_string'], either( render_paramdef_hash(), RenderRefP
```

3. tell RubyMine that *my\_render* call should have the same semantics as *render*.

```
paramdef 'ActionController::Base', ['render', 'render_to_string', 'my_render'], ...
```

Let's take a look on registration code. Registration is done using **paramdef** call.

- \* First argument is full name of method's containing class.
- \* Second argument is

- method name (String)
- or array of method names (Array of String)

\* Third etc. parameters describes semantics for each call argument. We use Tree like format of descriptions with leaf and composite nodes.

E.g *ActionController::Base.render* accepts:

1. method or view name
2. :update value
3. hash with options.

So we use

\* **either(...)** description to describe three alternatives of arguments semantics

\* **one\_of(:update)** - this describes that argument value is one of elements from list (strings or symbols)

\* **render\_paramdef\_hash()**. This describe hash keys and semantics for it's values, see method declaration at the end of file

```
def render_paramdef_hash() { :action => either(action_ref, view_ref), :text => nil, :template => view_ref, :partial =
```

\* **nil** - shows that we have no special description for leaf declaration, so smart autocompletion/inspections/resolve wont be activated. E.g.

```
:text => nil
```

will tell RubyMine that special semantic for value of *render :text => <caret>* isn't specified.  
And

```
:template => view_ref
```

will tell that value expected to be path to view file. Also you can specify fake key

```
:enable_optional_keys => true
```

which will allow RubyMine to not highlight all unmentioned keys as "unexpected hash key :my\_dynamically\_generated\_key". So as **Example 2** you can disable warning on unexpected hash keys for render method.

\* RenderRefParam.new - has semantics which is described in /rb/paramdefs/rails/render\_ref\_param.rb. Actually this description allows to resolve value as action method name or view file relative path and provides corresponding autocompletion.

---

Next extensions are not ready to be used outside our team but I've decided to mention them if somebody decided to take a look.

## Intention actions

You can find bundled examples at rb/scripts folder: braces\_to\_do.rb, modifier\_to\_statement.rb, string\_to\_symbol.rb

Such actions are displayed in **File | Settings | Intentions**. To register custom intention action you should use following template

```
include Java
# requires API which allow to register intention actions
require File.expand_path(File.dirname(__FILE__) + '/../api/intention_action_helper')
# registers action and provides it's implementation register_intention_action "My custom intention action name",

# here should be code which will take current PSI element from context and replace it with some other
# element. E.g. you can create string with source code and ask RubyMine to create PSI element from this
```

```
# string (e.g. context.create_element(":{s.content}") as in string_to_symbol.rb)
end
```

## Editor actions

These actions allow to change text in editor, move caret on shortcut. For example take a look on "rhtml\_braces.rb" which inserts <%= %> in rhtml(erb) files by Ctrl+Shift+.

### Template:

```
include Java
# requires api for editor action require File.expand_path(File.dirname(__FILE__) + '/../api/editor_action_helper.rb')

# action_name : String - name of action
# attrs : Hash
# :shortcut - string value, you can use use "control", "shift", "alt" modifiers, letters, etc. E.g.: :shortcut => "control shift PERIOD"
# :file_type - "RHTML", "Ruby", "HAML", "YAML", "CSS", "HTML", "XML", "Cucumber", "Rjs", "RXML". E.g: :file_type => "RHTML"
# :keymap - String, keymap name. Attribute is optional, if missed shortcut will be registered in all keymaps.
register_editor_action action_name, attrs do |editor, file| # editor - see EditorWrapper object in rb/utis/editor_wrapper.rb
  # file - com.intellij.psi.PsiFile, PSI element for file opened in editor
end
```

## Typed Action

This action will be invoked on each typed character in ruby code. E.g if you select text and type '#' then selected text will be converted to "#{SELECTED\_TEXT}". See example at rb/scripts/surround\_selection.rb

### Template:

```
Include Java
# attaches api for typed handler require File.expand_path(File.dirname(__FILE__) + '/../api/typed_action_helper.rb')

# registers action
# character - typed character
# project - current opened RubyMine com.intellij.openapi.project.Project instance. Is necessary for some other methods.
# editor - see EditorWrapper object in rb/utis/editor_wrapper.rb
# file - PsiFile
# returns - false if RubyMine should invoke try to process using other registered actions, true if should stop or
           String value if should stop processing and replace select text with value of this string.
```

## RubyMine 1.1.1 internal ruby API tips

```
register_typed_action lambda { |character, project, editor, file| # implementation
}
```

Tags: api



[linux\\_china](#) 263 posts since

Sep 2, 2002 1. **Re: RubyMine 1.1.1 internal ruby API tips** Jul 10, 2009 1:37 PM

Great new! I will take a try. There are a lot of things in Ruby, so extension is very important for developer. 😊



[Roman.Chernyatchik](#) 1,452 posts since

Sep 18, 2007 2. **Re: RubyMine 1.1.1 internal ruby API tips** Jul 10, 2009 1:53 PM

In coming 1.5 first EAP will be also available some additional API. I'll write about it later.



[Mark Ericksen](#) 11 posts since

Dec 12, 2009 3. **Re: RubyMine 1.1.1 internal ruby API tips** Dec 12, 2009 7:11 PM

Can you tell me if the Editor Actions are supported yet in RM 2.0?

That's really the thing I care most about at this point. Coming from Netbeans, I'm accustomed to a few editor shortcuts that I can't get working in RM. I've tried creating the extensions in RM 2.0 but I couldn't get the keyboard bindings to register they were there and work.

What I want:

- Change the <%= %> keybindgs. (Want it to be "re+TAB")
- Add <% %> entry bound to "r+TAB"
- Change the behavior of the " => " entry and keybinding. The current script includes a leading space before the equal sign. I don't want that. I want to space, hit "l+TAB" and have it add it.

I'm sure there are other bindings and shortcuts that I will want to make as well.

RubyMine 1.1.1 internal ruby API tips

Any ETA, workarounds, or tips on this? Otherwise I really enjoy RM. However these bits annoy me everytime I encounter them (which can be often).

Thanks,

-Mark E.



[Dmitry Jemerov](#) 11,708 posts since

Aug 19, 2002 4. **Re: RubyMine 1.1.1 internal ruby API tips** Dec 14, 2009 3:01 PM

Hello Mark,

For all of the things you've listed, you don't need to use the extensibility APIs at all. Simply go to Settings | Live Templates, define the snippets you need, specify the abbreviations and set Tab to use as the key to expand them.

Can you tell me if the Editor Actions are supported yet in RM 2.0?

That's really the thing I care most about at this point. Coming from Netbeans, I'm accustomed to a few editor shortcuts that I can't get working in RM. I've tried creating the extensions in RM 2.0 but I couldn't get the keyboard bindings to register they were there and work.

What I want:

- Change the <%= %> keybindgs. (Want it to be "re+TAB")
- Add <% %> entry bound to "r+TAB"
- Change the behavior of the " => " entry and keybinding. The current script includes a leading space before the equal sign. I don't want

that. I want to space, hit "l+TAB" and have it add it.

I'm sure there are other bindings and shortcuts that I will want to make as well.

Any ETA, workarounds, or tips on this? Otherwise I really enjoy RM.

However these bits annoy me everytime I encounter them (which can be often).

Thanks,

-Mark E.

---

Original message URL:

<http://www.jetbrains.net/devnet/message/5252511#5252511>

--

Dmitry Jemerov

Development Lead

JetBrains, Inc.

<http://www.jetbrains.com/>

"Develop with Pleasure!"

RubyMine 1.1.1 internal ruby API tips



[Mark Ericksen](#) *11 posts since*

*Dec 12, 2009* **5. Re: RubyMine 1.1.1 internal ruby API tips** *Dec 14, 2009 5:07 PM*

Perfect! I've played with it and, yes, it does everything I want.

Thanks for answering my question and sharing the method!

-Mark E.