

# An Introduction to Desktop Apps with Ruby

**It's easy to create nifty little cross-platform system tray/menu bar application using JRuby.**

by RubyLearning  
<http://rubylearning.com/blog/>

# IMPORTANT

**You Do NOT Have Rights to Edit, Resell or Claim Ownership to this Document!**

**However...You DO Have the Right to Pass this Document Along to Others Who Might Benefit from it!**

**Feel free to share it with your blog readers and give it away as a freebie.**

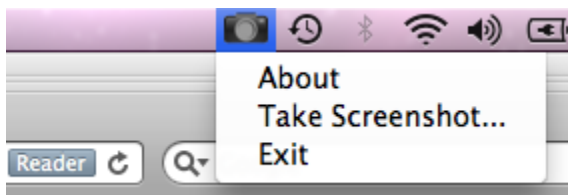
**ALL RIGHTS RESERVED:** You do not have any rights to sell or profit from this document. All content is to remain unedited and all links must stay in tact as they are. You can not claim any type of ownership without express written permission from the creator and only the creator, Martin Sadler. All rights to this report belong to the author only.

**DISCLAIMER:** All information contained within this document is strictly the views represented by Martin Sadler, hereafter referred to as the author or author, at time of publication. The author reserves all rights. Reasonable attempt has been made to accurately substantiate all information in this document. However, the author, does not, and cannot, take responsibility for any errors or exclusions that may be contained in within. No suitability of use is meant or implied in regard to any system or program, use at your own risk. It is recommended the reader contact the appropriate qualified professional for advice in these and any other areas should it be needed.

# An Introduction to Desktop Apps with Ruby

## *Introduction*

**I**n this article I'm going to show you how easy it is to create a nifty little cross-platform system tray/menu bar application using [JRuby](#). The application enables a user to take a screenshot of their desktop and then do something interesting with it!



## *What's covered in this article*

Some of the key areas covered include:

- Using Java Classes within Ruby
- Refactoring
- Blocks

**Note:** This tutorial assumes no prior Java knowledge and covers basic Ruby concepts only.

## *What is JRuby?*

[JRuby](#) is a version of Ruby that runs on top of the [JVM](#). It runs just the same as regular [MRI Ruby](#) except it's about twice as fast and you get access to existing Java libraries.

## ***Getting started with JRuby***

It's never been easier to try JRuby out. RVM is your friend here. Assuming you have RVM already installed it's simply a case of doing the following inside your terminal session:

```
rvm install jruby
```

and then (where x.y.z is the version of JRuby)

```
rvm use jruby-x.y.z
```

From this point all your normal ruby commands will be using JRuby in the current terminal. Sweet.

## **Part 1: Taking a screenshot**



Robot does for the Desktop what Selenium and other automation tools do for the browser. The main difference is that you have far more access to a user's machine. You can move the mouse around, control keystrokes, manipulate other programs, and most usefully take a screenshot.

What is Robot? It's a Java class and we going to make good use of it in our first script.

## Our first JRuby script

```
#-----  
# Summary: Takes a screenshot of the desktop and saves to disk  
#-----  
  
# this gives us direct access to Java classes in Ruby  
# example1.rb  
include Java  
  
# here we are importing Java classes, just like you might require 'yaml' or  
'date'  
import java.awt.Robot  
import java.awt.Toolkit  
import java.awt.Rectangle  
import javax.imageio.ImageIO  
  
# Taking the screenshot  
# 1) Create a new instance of the Robot class  
# 2) Use the Toolkit class to get the size of the screen  
# 3) and pass those dimensions to the Robot instance for capture  
robot      = Robot.new  
toolkit     = Toolkit.get_default_toolkit  
dim         = toolkit.get_screen_size  
rectangle   = Rectangle.new(0, 0, dim.get_width, dim.get_height)  
image       = robot.create_screen_capture(rectangle)  
  
# Save the file to disk  
file = java::io::File.new('test.png')  
ImageIO.write(image, "png", file)
```

Running the above script is a simple case of:

```
ruby example1.rb
```

The resulting screenshot gets saved to the same directory you are in as 'test.png'. Result.

**Tip:** One question you might be asking by this point is how do you know what classes are available when you include Java? [Java API docs](#) have the answer.

Now, although helpful, it's still pure Java so you'll need to do a bit of translating to the equivalent JRuby calls. The tree view makes it easier to see how all the classes fit together.

So we've got a script to take a screenshot, it does the job, but we can make it better.

## Refactoring into a re-usable Ruby Class

```
# screenshot.rb
class Screenshot
  include Java

  import java.awt.Robot
  import java.awt.Toolkit
  import java.awt.Rectangle
  import javax.imageio.ImageIO

  def self.capture(filename = 'screenshot.png')
    robot      = Robot.new
    toolkit    = Toolkit.get_default_toolkit
    dim        = toolkit.get_screen_size
    rectangle  = Rectangle.new(0, 0, dim.get_width,
dim.get_height)
    image      = robot.create_screen_capture(rectangle)
    file       = java::io::File.new(filename)
    ImageIO::write(image, "png", file)
  end
end
```

Now fire up a copy of irb and enter the following:

```
require 'screenshot'  
Screenshot.capture('screenshot1.png')
```

By moving the code into a class method it's much tidier and easier to use for the next part of the application.

**Gotcha:** File naming is really important here. The Ruby class defined in the file must be the CamelCase version of the filename for the require to work otherwise you'll get errors like :  
LoadError: use 'java\_import' to load normal Java classes  
Note, this is only the case in JRuby not regular MRI Ruby

## Part 2: Creating the system tray application

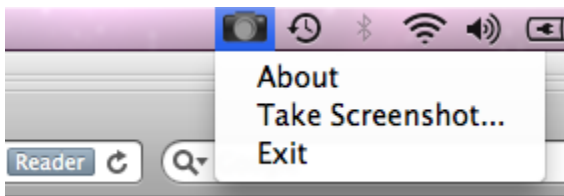
Now we have a class for taking a screenshot lets create the system tray application:

```
#-----  
-  
# Summary: Runs a system tray application with ability to take a screenshot  
#-----  
-  
# example2.rb  
require 'screenshot'  
  
# Import the Java class libraries we wish to use  
include Java  
import java.awt.TrayIcon  
import java.awt.Toolkit  
  
# Setup our menu items  
exititem      = java.awt.MenuItem.new("Exit")  
screenshotitem = java.awt.MenuItem.new("Take Screenshot...")  
aboutitem     = java.awt.MenuItem.new("About")  
  
# Event handling  
exititem.add_action_listener {java.lang.System::exit(0)}  
screenshotitem.add_action_listener {Screenshot.capture}  
  
# Add the items to the popup menu itself  
popup = java.awt.PopupMenu.new  
popup.add(aboutitem)  
popup.add(screenshotitem)  
popup.add(exititem)  
  
# Give the tray an icon and attach the popup menu to it  
image = java.awt.Toolkit::default_toolkit.get_image("screenshot.gif")  
tray_icon = TrayIcon.new(image, "Screenshot!", popup)  
tray_icon.image_auto_size = true  
  
# Finally add the tray icon to the tray  
tray = java.awt.SystemTray::system_tray  
tray.add(tray_icon)
```



Before you run this script make sure you have an icon named screenshot.gif in the same directory. Short of creating one you can use [Google to find an icon](#).

On running the application you'll see the menu appear in your system tray. Click the screenshot menu item and the screenshot gets saved to disk!



## Refactor

The application is neat but it feels quite bulky, lets condense it down in to a much tighter API:

```
# example3.rb
require 'tray_application'; require 'screenshot';
app = TrayApplication.new("Deskshot")
app.icon_filename = 'screenshot.png'
app.item('Take Screenshot') {Screenshot.capture}
app.item('Exit')           {java.lang.System::exit(0)}
app.run
```

Not bad for an application of 6 lines! Where did all that code go?

```

# tray_application.rb
class TrayApplication

  include Java
  import java.awt.TrayIcon
  import java.awt.Toolkit

  attr_accessor :icon_filename
  attr_accessor :menu_items

  def initialize(name = 'Tray Application')
    @menu_items = []
    @name = name
  end

  def item(label, &block)
    item = java.awt.MenuItem.new(label)
    item.add_action_listener(block)
    @menu_items << item
  end

  def run
    popup = java.awt.PopupMenu.new
    @menu_items.each{|i| popup.add(i)}

    # Give the tray an icon and attach the popup menu to it
    image =
java.awt.Toolkit::default_toolkit.get_image(@icon_filename)
    tray_icon = TrayIcon.new(image, @name, popup)
    tray_icon.image_auto_size = true

    # Finally add the tray icon to the tray
    tray = java.awt.SystemTray::system_tray
    tray.add(tray_icon)
  end

end

```

A similar pattern of refactoring was carried out here as per the Screenshot class.

Probably the most interesting point is the way you can use Ruby to capture a block of code. In the case of the Screenshot application there are certain actions we only want to be executed when the user clicks a menu item. You can see these as code surrounded by curly braces. Alternatively they could have been written as follows:

```
app.item('Take Screenshot') do
  Screenshot.capture
End
```


Which does more or less the same job. Usually the do/end format is preferred for multi-line code blocks.

What is handy for us is that we can capture these code blocks using the &block parameter in the method definition. This makes the block available as a local variable inside the method and it can be simply pushed onto an array (@menu\_items) for later recall.

Curious as to how a block is executed once stored? Here is a simple example:

```
def hello(&block)
  block.call # execute the contents of the hello block
using call
end

hello do
  p "hello there"
end
```



**Tip:** Always try to keep your code tidy as you go along. Look for patterns and bake them into classes if need be. Think about how the end product will look like and work out how to get there. See also: [Readme Driven Development](#).

### ***Part 3: One further improvement***

So now we have a tray application that takes a screenshot and saves to disk. How about we get it to open in an image browser too? No problem.

Lets make a small addition by making use of the [Desktop Java Class](#).

```
# screenshot.rb (revisited)
class Screenshot
  include Java

  import java.awt.Desktop # added Desktop to import
  import java.awt.Robot
  import java.awt.Toolkit
  import java.awt.Rectangle
  import javax.imageio.ImageIO

  def self.capture(filename = 'screenshot.png')
    robot = Robot.new
    toolkit = Toolkit.get_default_toolkit
    dim = toolkit.get_screen_size
    rectangle = Rectangle.new(0, 0, dim.get_width, dim.get_height)
    image = robot.create_screen_capture(rectangle)

    file = java::io::File.new(filename)
    ImageIO::write(image, "png", file)

    # Open the file in the users default application for the given file type
    desktop = Desktop.get_desktop
    desktop.open(file)
  end
end

end
```

So now when you click ‘Take Screenshot..’ up pops Preview (on Mac) with your Desktop image.

## ***Beyond the screenshot***

So we have put together the foundations of a screenshot tray application and in the process built a skeleton GUI [JRuby](#) framework. So where could we go from here?

- Send the screenshot to a web service

This would make for a handy utility for a [support web app](#). Take the file and post it via http to your app for processing.

- Add an About Box

Sadly the amount of code to show this here would have detracted from the main article. However it is still pretty straight forward. JFrame is your friend.

- HotKey for taking the screenshot

- Package! [Rawr](#). Distribute

So currently you'll need JRuby in order to run this application. One of the main benefits of JRuby is it compiles down to [Java bytecode](#) and will run on any Java enabled platform. Mac, Windows, and Unix support in one go!

- Variations on the theme

Why stop at screenshots? Here are few more ideas:

- Drag and drop copy and upload of files to a web service
- A notification service e.g. new support messages, tweets
- System analytics tool
- Copy and Paste share bin

## **Reference**

- [JRuby Homepage](#)
- [Java API AWT docs](#)
- [JRuby GUI Frameworks](#)

## **Attribution**

- Robot

Image: <http://www.flickr.com/photos/loresjoberg/2212063279>

## **In Summary**

*I hope you found this article valuable and that it gives you an insight into the world of possibilities with JRuby and desktop applications. Feel free to ask questions and give feedback in the comments section of [this blog post](#). Thanks!*