# Keyboard Emulation

## GUI automation using a Robot

Wednesday, September 15th, 2010

I would like to welcome guest writer [*Takeshi (Kesh) Ikuma*](http://www.mathworks.com/matlabcentral/fileexchange/authors/36665). Kesh has posted several interesting utilities on the Matlab File Exchange, including the award-winning [*Enhanced Input Dialog Box*](http://www.mathworks.com/matlabcentral/fileexchange/25862-enhanced-input-dialog-box). Today, Kesh will describe how we can automate GUI actions programmatically.

http://images.undocumentedmatlab.com/robot.gif

Automating GUI actions, including controlling a mouse and keyboard programmatically, is often useful. This can be used, for example, to demonstrate GUI usage or to perform a recorded macro.

Matlab’s Handle Graphics interface provides a simple way to manipulate the mouse cursor position, but not to emulate mouse or keyboard clicks. However, we can utilize Java’s [java.awt.Robot](http://download.oracle.com/javase/6/docs/api/java/awt/Robot.html) class.

This article provides an overview of the Robot class and how it can be used to program mouse movement, button clicks and keyboard strikes.

### java.awt.Robot class

The online Java documentation describes the purpose of the Robot class as follows:  
This class is used to generate native system input events for the purposes of test automation, self-running demos, and other applications where control of the mouse and keyboard is needed.

This class has three main functionalities: mouse control, keyboard control, and screen capture. Here are some of the important member functions:

#### Mouse control functions

*void* [*mouseMove*](http://download.oracle.com/javase/6/docs/api/java/awt/Robot.html#mouseMove%28int,%20int%29)*(int x, int y)*

This function moves the cursor to the coordinate (x, y) which is defined with respect to the top-left screen corner (in contrast to Matlab’s coordinate origin at the bottom-left corner).

*void* [*mousePress*](http://download.oracle.com/javase/6/docs/api/java/awt/Robot.html#mousePress%28int%29)*(int buttons)  
void* [*mouseRelease*](http://download.oracle.com/javase/6/docs/api/java/awt/Robot.html#mouseRelease%28int%29)*(int buttons)*

This pair of functions performs the button click. Their input argument is an OR’ed combination of [java.awt.event.InputEvent](http://download.oracle.com/javase/6/docs/api/java/awt/event/InputEvent.html)s:

java.awt.event.InputEvent.BUTTON1\_MASK *// left mouse button*

java.awt.event.InputEvent.BUTTON2\_MASK *// middle mouse button*

java.awt.event.InputEvent.BUTTON3\_MASK *// right mouse button*

#### Keyboard control functions

Keyboard action is emulated by the following pair of functions. Their keycodes are defined in [java.awt.event.KeyEvent](http://download.oracle.com/javase/6/docs/api/java/awt/event/KeyEvent.html):

*void* [*keyPress*](http://download.oracle.com/javase/6/docs/api/java/awt/Robot.html#keyPress%28int%29)*(int keycode)  
void* [*keyRelease*](http://download.oracle.com/javase/6/docs/api/java/awt/Robot.html#keyRelease%28int%29)*(int keycode)*

NOTE: Although java.awt.event.KeyEvent constants defines most of the US QWERTY keys, not all can actually be used with java.awt.Robot. Specifically, it appears that only the KeyEvent constants for unmodified keys can be used. See the following section for an example.

#### Utility functions

The robot can be put to sleep for a desired duration (in milliseconds). Also, the calling routine can be blocked until the robot exhausts its command queue.

*void* [*delay*](http://download.oracle.com/javase/6/docs/api/java/awt/Robot.html#delay%28int%29)*(int ms)  
void* [*waitForIdle*](http://download.oracle.com/javase/6/docs/api/java/awt/Robot.html#waitForIdle%28%29)*()*

Readers interested in Robot’s screen-capture capability are invited to take a look at Yair’s [ScreenCapture utility](http://www.mathworks.com/matlabcentral/fileexchange/24323-screencapture-get-a-screen-capture-of-a-figure-frame-or-component) on the File Exchange.

### Using java.awt.Robot in Matlab

To create the Robot object in Matlab, simply run

robot = java.awt.Robot;

#### Moving the mouse cursor

With Matlab’s root (0) handle, the mouse cursor can be repositioned by

set(0,'PointerLocation',[x y]);

The mouse cursor is placed on the x-th pixel from left edge and the y-th pixel from the bottom edge of the screen.

Alternately, we could use the semi-documented [***moveptr*** function](http://undocumentedmatlab.com/blog/undocumented-mouse-pointer-functions/#moveptr), which was described here last year.

The same operation can also be done using Robot as follows:

scrSize = get(0,'ScreenSize');

robot.mouseMove(x,scrSize(2)-y);

The extra step must be taken to convert from Matlab to Java screen coordinates.

Depending on the specific case (for example, whether or not we know the absolute or only relative screen coordinates), we may prefer using any of these equivalent mouse-movement alternatives.

#### Clicking mouse buttons

Unfortunately, we have few alternatives for automating mouse-clicks – Robot is basically our only option. Matlab recognizes 4 different mouse click types as specified in the Figure’s **SelectionType** property:

* *Normal*: Click left mouse button
* *Extend*: SHIFT-click left mouse button
* *Alternate*: CTRL-click left mouse button
* *Open*: Double click left mouse button

To observe the mouse click, open a figure and set its **WindowButtonDownFcn** callback:

myCallback = @(hObj,event) disp(get(hObj,'SelectionType'));

set(gcf,'WindowButtonDownFcn', myCallback);

To trigger the callback, the figure must have the focus and the mouse cursor must be on the figure. To ensure the callback invocation, “figure(gcf); drawnow;” commands are included in the code examples below. Make sure to place the mouse cursor on the figure before running these codes.

Here’s how the Robot can be used to perform each type of clicking. First, normal click:

figure(gcf); drawnow;

robot.mousePress (java.awt.event.InputEvent.BUTTON1\_MASK);

robot.mouseRelease(java.awt.event.InputEvent.BUTTON1\_MASK);

For “open” or double click, repeat it twice:

figure(gcf); drawnow;

robot.mousePress (java.awt.event.InputEvent.BUTTON1\_MASK);

robot.mouseRelease(java.awt.event.InputEvent.BUTTON1\_MASK);

robot.mousePress (java.awt.event.InputEvent.BUTTON1\_MASK);

robot.mouseRelease(java.awt.event.InputEvent.BUTTON1\_MASK);

For the other two click types, we need to use the keyboard functions. First, “extend” or SHIFT-click:

figure(gcf); drawnow;

robot.keyPress (java.awt.event.KeyEvent.VK\_SHIFT);

robot.mousePress (java.awt.event.InputEvent.BUTTON1\_MASK);

robot.mouseRelease(java.awt.event.InputEvent.BUTTON1\_MASK);

robot.keyRelease (java.awt.event.KeyEvent.VK\_SHIFT);

Lastly, “alternate” or CTRL-click:

figure(gcf); drawnow;

robot.keyPress (java.awt.event.KeyEvent.VK\_CONTROL);

robot.mousePress (java.awt.event.InputEvent.BUTTON1\_MASK);

robot.mouseRelease(java.awt.event.InputEvent.BUTTON1\_MASK);

robot.keyRelease (java.awt.event.KeyEvent.VK\_CONTROL);

#### Clicking keyboard keys

Just as we have shown above how to press a modifier key (Ctrl or Alt key) to set the figure’s **SelectionType** property, keyPress and keyRelease functions can be used with character keys to type text. For example, with the focus on Matlab’s Command Prompt, running the following code executes the ***ver*** command:

robot.keyPress (java.awt.event.KeyEvent.VK\_V);

robot.keyRelease (java.awt.event.KeyEvent.VK\_V);

robot.keyPress (java.awt.event.KeyEvent.VK\_E);

robot.keyRelease (java.awt.event.KeyEvent.VK\_E);

robot.keyPress (java.awt.event.KeyEvent.VK\_R);

robot.keyRelease (java.awt.event.KeyEvent.VK\_R);

robot.keyPress (java.awt.event.KeyEvent.VK\_ENTER);

robot.keyRelease (java.awt.event.KeyEvent.VK\_ENTER);

In the previous section, I mentioned that not all KeyEvent constants are supported by java.awt.Robot, and keys must be defined with their unmodified characters. For example, a semicolon (‘;’) can be typed by:

robot.keyPress (java.awt.event.KeyEvent.VK\_SEMICOLON);

robot.keyRelease (java.awt.event.KeyEvent.VK\_SEMICOLON);

However, doing the same for a colon (‘:’ or SHIFT-’;') causes an error:

robot.keyPress (java.awt.event.KeyEvent.VK\_COLON);

robot.keyRelease (java.awt.event.KeyEvent.VK\_COLON);

??? Java exception occurred:

java.lang.IllegalArgumentException: Invalid key code

at sun.awt.windows.WRobotPeer.keyPress(Native Method)

at java.awt.Robot.keyPress(Unknown Source)

Instead of using the VK\_COLON constant, VK\_SEMICOLON constant must be used while SHIFT modifier key is pressed:

robot.keyPress (java.awt.event.KeyEvent.VK\_SHIFT);

robot.keyPress (java.awt.event.KeyEvent.VK\_SEMICOLON);

robot.keyRelease (java.awt.event.KeyEvent.VK\_SEMICOLON);

robot.keyRelease (java.awt.event.KeyEvent.VK\_SHIFT);

Although unconfirmed, I suspect VK\_SEMICOLON constant would not work with keyboards in which semicolon is mapped to SHIFT-’,’ (e.g., Spanish or Italian keyboard layouts), while the VK\_COLON constant would work with a French keyboard.

### Putting it all together: jMouseEmu and inputEmu

To simplify the execution of the mouse and keyboard actions outlined above, I (Kesh Ikuma) have created the [jMouseEmu](http://www.mathworks.com/matlabcentral/fileexchange/28357-jMouseEmu) and [inputEmu](http://www.mathworks.com/matlabcentral/fileexchange/28603-inputEmu) utilities, which are available on the Matlab File Exchange. These interface/wrapper functions will be described in next week’s article.

Do you have any favorite use for Java Robots in your workflow? If so, please tell us all about it in a comment [below](http://undocumentedmatlab.com/blog/gui-automation-robot/#respond).

Related posts:

1. [GUI automation utilities](http://undocumentedmatlab.com/blog/gui-automation-utilities) This article explains a couple of Matlab utilities that use Java's Robot class to programmatically control mouse and keyboard actions...
2. [ScreenCapture utility](http://undocumentedmatlab.com/blog/screencapture-utility) The ScreenCapture utility uses purely-documented Matlab for capturing a screen region as an image from within Matlab. ...
3. [Uicontrol callbacks](http://undocumentedmatlab.com/blog/uicontrol-callbacks) This post details undocumented callbacks exposed by the underlying Java object of Matlab uicontrols, that can be used to modify the control's behavior in a multitude of different events...
4. [Detecting window focus events](http://undocumentedmatlab.com/blog/detecting-window-focus-events) Matlab does not have any documented method to detect window focus events (gain/loss). This article describes an undocumented way to detect such events....
5. [Matlab-Java interface using a static control](http://undocumentedmatlab.com/blog/matlab-java-interface-using-static-control) The switchyard function design pattern can be very useful when setting Matlab callbacks to Java GUI controls. This article explains why and how....
6. [Inactive Control Tooltips & Event Chaining](http://undocumentedmatlab.com/blog/inactive-control-tooltips-event-chaining) Inactive Matlab uicontrols cannot normally display their tooltips. This article shows how to do this with a combination of undocumented Matlab and Java hacks..