# Introduction to Low-Level Scripting


## By Michael Kelly



**First published on the Rational Developer Network**
http://www.rational.net


Deep in the dark corners of the IBM Rational® Robot tool, there lurks a little-talked-about, often-misunderstood feature known as low-level recording. With this feature, you can record mouse movements and keyboard actions by screen coordinates and exact timing. While impractical for most uses, this feature nonetheless has its place in the fight against software bugs. This article will show you why this functionality can open doors that are closed to object-oriented recording and will tutor you in how and when to use it.

## What Is Low-Level Recording?

Robot has two recording modes: object-oriented and low-level. Object-oriented recording is what you get by default when you record a new script in Robot. It captures objects in the application-under-test at the window layer and then uses internal object names (instead of mouse movements or absolute screen coordinates) to identify them. If objects in your application's graphical user interface change locations, or your screen resolution changes, your tests still execute because the scripts aren't location dependent. I always advocate this as the best model for script development and suggest building frameworks to further modularize your scripts and make them even less dependent on the application-under-test.

Low-level recording, by contrast, captures the exact mouse movements and keyboard actions performed by the user. These actions are stored in an ASCII file, called a journal file, that's attached to your script just like a verification point. Graphically, you see such files listed in the asset pane on the left side of the script window, and in the repository they're stored in the `<datastore>\TMS_Scripts\lls` directory. When you delete a script, all of its low-level files are deleted. You can copy a low-level file from one script to another. In all of these ways, a low-level script behaves just like a verification point. Simply think of it as another test asset that you can associate with a script.

## A Look at the Low-Level Script File

Let's look at a journal file containing a low-level script. If you were to open one up you would see something similar to Figure 1.

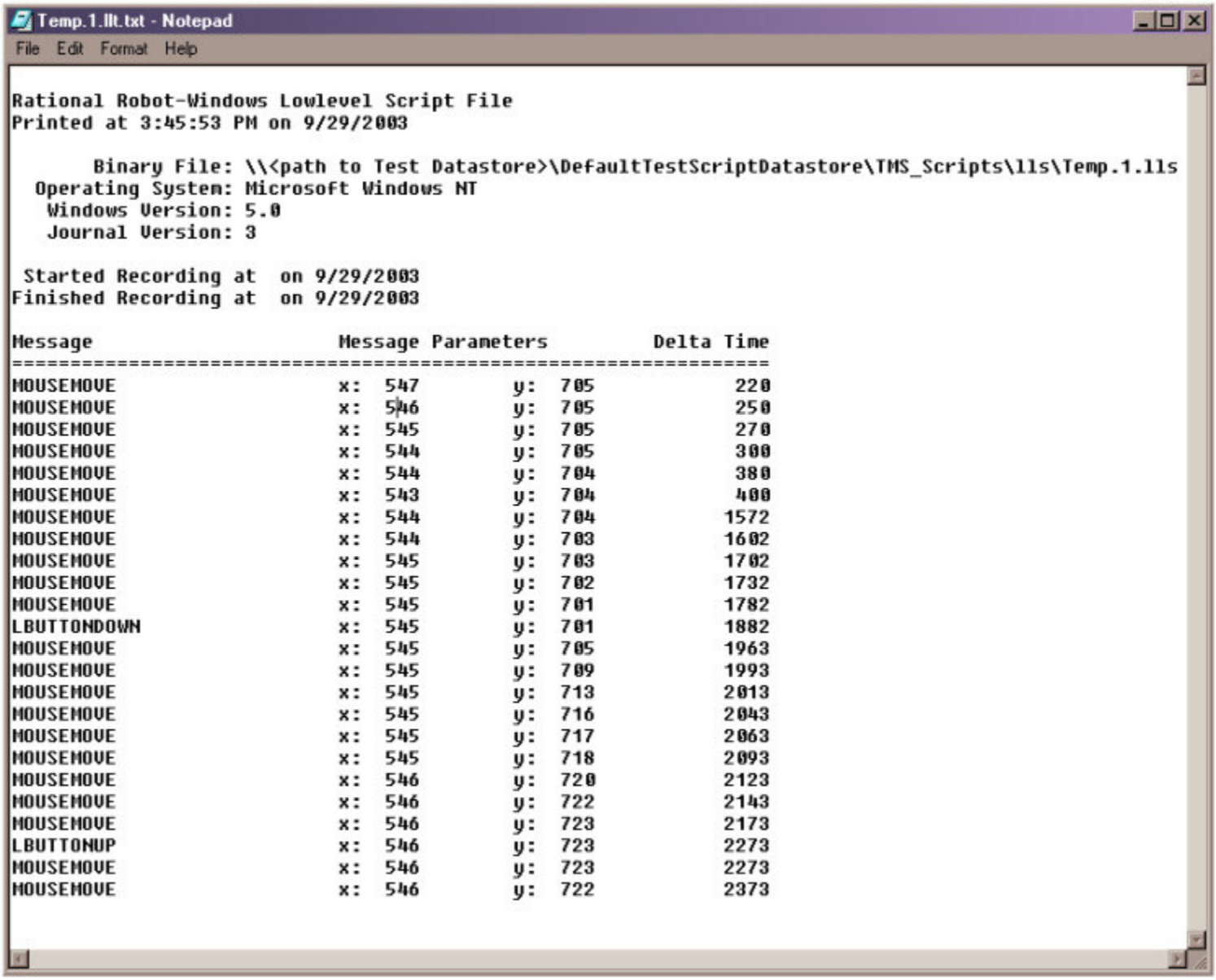



***Figure 1:*** *A typical low-level script file*

This low-level script draws the number "1" using the mouse. The only change from the original that I've made is to shorten the path to the binary file due to length constraints. As you can see, the file is very simple. It contains a header with metadata for the file, and three columns:

- The Message column is a keyword description of the type of action that took place at each point in the low-level script.
- The Message Parameters column contains either the horizontal (x) and vertical (y) axis screen-pixel coordinates of the location where the low-level action occurred, or a Vkey value (not shown), which is a low-level representation of the keyboard action performed.
- The Delta Time column shows the cumulative elapsed time (in milliseconds) required to complete the low-level actions.

Now let's go through the script and look at the mouse movements and keyboard actions it captured. First I moved the mouse to get the mouse pointer where I wanted it:

```
MOUSEMOVE       x:  547     y:  705      220
MOUSEMOVE       x:  546     y:  705      250
MOUSEMOVE       x:  545     y:  705      270
MOUSEMOVE       x:  544     y:  705      300
MOUSEMOVE       x:  544     y:  704      380
MOUSEMOVE       x:  543     y:  704      400
MOUSEMOVE       x:  544     y:  704     1572
MOUSEMOVE       x:  544     y:  703     1602
MOUSEMOVE       x:  545     y:  703     1702
MOUSEMOVE       x:  545     y:  702     1732
MOUSEMOVE       x:  545     y:  701     1782
```

Then I pressed the left mouse button and held it down:

```
LBUTTONDOWN     x:  545     y:  701     1882
```

I drew a nearly straight line, as evidenced by the one-pixel change in the x coordinate:

```
MOUSEMOVE       x:  545     y:  705     1963
MOUSEMOVE       x:  545     y:  709     1993
MOUSEMOVE       x:  545     y:  713     2013
MOUSEMOVE       x:  545     y:  716     2043
MOUSEMOVE       x:  545     y:  717     2063
MOUSEMOVE       x:  545     y:  718     2093
MOUSEMOVE       x:  546     y:  720     2123
MOUSEMOVE       x:  546     y:  722     2143
MOUSEMOVE       x:  546     y:  723     2173
```

Finally, I released the left mouse button, completing the number:

```
LBUTTONUP       x:  546     y:  723     2273
```

When you understand the format of the file, you can read it to figure out what the script is doing. Various actions are read as follows:

- As seen above, a mouse drag is a `BUTTONDOWN` followed by `MOUSEMOVE` commands and finished with a `BUTTONUP`.
- A click is a `BUTTONDOWN` followed immediately by a `BUTTONUP`:

```
LBUTTONDOWN     x:  781     y:  481     1071
LBUTTONUP       x:  781     y:  481     1172
```

- A double click would be the same as above, just repeated for each click.
- For scrolling, if you click the scroll box and then move the mouse to navigate, it will look like the following:

```
MBUTTONDOWN     x:  773     y:  386     7341
MOUSEMOVE       x:  773     y:  386     7351
MBUTTONUP       x:  773     y:  386     7501
```

- If you scroll by using the scroll wheel on the mouse (up and down), it will look like this:

```
SYSKEYUP        0x318       0x1E2       9644
SYSKEYUP        0x318       0x1E2       9654
SYSKEYUP        0x318       0x1E2       9684
SYSKEYUP        0x318       0x1E2       9914
SYSKEYUP        0x318       0x1E4       9934
```

- Pressing a key on the keyboard will look like this:

```
KEYDOWN         VKey: 0x51  Scan: 0x10  12177
KEYUP           VKey: 0x51  Scan: 0x10  12318
```

- Pressing a keyboard combination, such as Ctrl + Shift, will be represented by a double `KEYDOWN` followed by a double `KEYUP`, as follows:

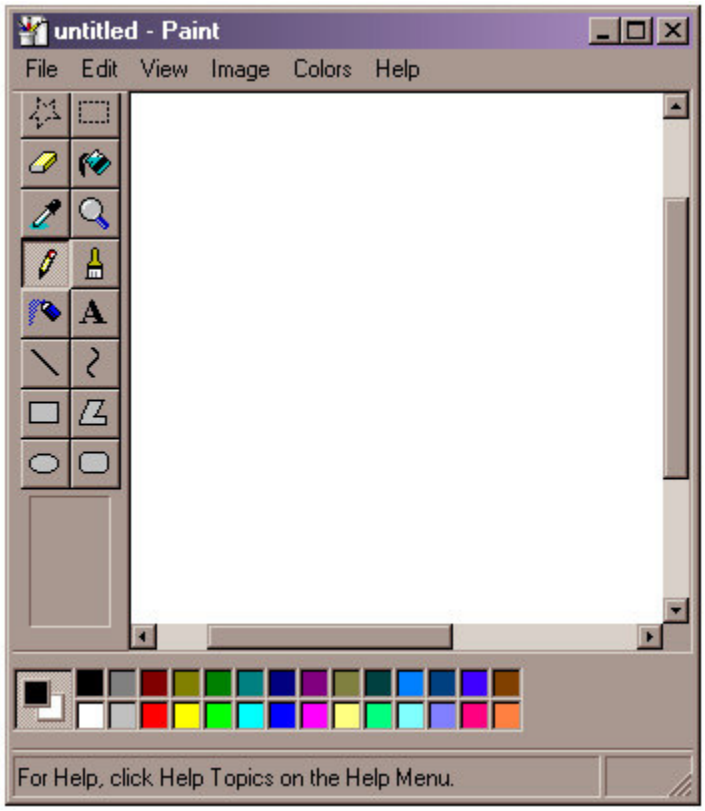
```
KEYDOWN          VKey: 0x11     Scan: 0x1D          13900
KEYDOWN          VKey: 0x5A     Scan: 0x2C          14260
KEYUP            VKey: 0x5A     Scan: 0x2C          14401
KEYUP            VKey: 0x11     Scan: 0x1D          14441
```

There are other commands and combinations you can run into, but those are the basics.

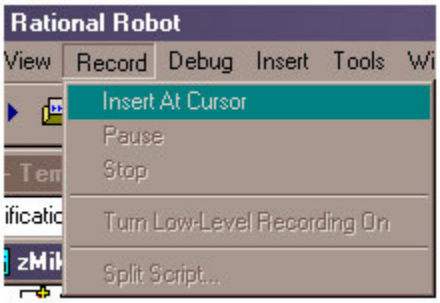
## Low-Level Recording in Action

Let's record a couple of scripts to demonstrate the difference between object-oriented recording and low-level recording. Perform the following steps:

1. Create a new GUI script.
2. Open Microsoft Paint and select the Pencil tool:




3. In your new script, choose Record > Insert At Cursor:



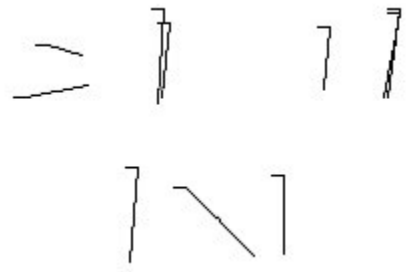

4. Record yourself writing "ABCD 123" in the Paint application:

5. Stop recording.

The script you just recorded was object-oriented, the default setting for Robot. Your script should look similar to the following:

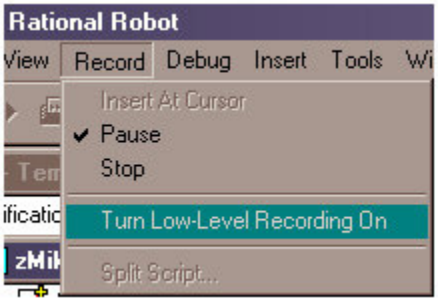
```
Sub Main

    'ABCD 123 - Record and Playback
    Window SetContext, "Caption=untitled - Paint", ""
    GenericObject Left_Drag, "Class=Afx:;ClassIndex=1", "Coords=42,110,79,104"
    GenericObject Left_Drag, "Class=Afx:;ClassIndex=1", "Coords=53,84,76,89"
    GenericObject Left_Drag, "Class=Afx:;ClassIndex=1", "Coords=111,66,114,113"
    GenericObject Left_Drag, "Class=Afx:;ClassIndex=1", "Coords=114,73,116,110"
    GenericObject Left_Drag, "Class=Afx:;ClassIndex=1", "Coords=193,75,196,106"
    GenericObject Left_Drag, "Class=Afx:;ClassIndex=1", "Coords=228,66,228,110"
    GenericObject Left_Drag, "Class=Afx:;ClassIndex=1", "Coords=228,68,226,110"
    GenericObject Left_Drag, "Class=Afx:;ClassIndex=1", "Coords=98,145,100,192"
    GenericObject Left_Drag, "Class=Afx:;ClassIndex=1", "Coords=122,155,161,189"
    GenericObject Left_Drag, "Class=Afx:;ClassIndex=1", "Coords=170,149,176,188"

End Sub
```

6. Switch to the Paint application, open a new picture, and make sure the Pencil tool is still selected.
7. Play back the script just recorded. You should get a result similar to the following:



Not quite what you recorded, eh? Now we'll try the same thing using low-level recording.

1. Again, switch to the Paint application, open a new picture, and make sure the Pencil tool is still selected.
2. In your script, delete the object-oriented code just recorded and again choose Record > Insert At Cursor.
3. Select the title bar on the Paint application to set the window context for testing.
4. Turn low-level recording on in one of two ways:
    - Maximize Robot and from the Record menu choose Turn Low-Level Recording On:




- Use the hotkeys combination Ctrl + Shift + R. (I use the hotkeys so there's less cleanup in the script.)

5. Record yourself writing "ABCD 123" in the Paint application:

6. You can then simply stop the recording, or you can turn low-level recording off first. For this test it won't make a difference, but you should get in the habit of disabling low-level recording when you've finished using it.

The script you just recorded should contain a couple of lines of object-oriented code (where you set the window context) and then a call to a journal file. It should look similar to the following:

```
Sub Main

    'ABCD 123 - Low-level Record
    Window SetContext, "Caption=untitled - Paint", ""
    PlayJrnl ("001")

End Sub
```

7. Switch to the Paint application, open a new picture, and make sure the Pencil tool is still selected.
8. Play back the script you just recorded.

You should get a result similar to the following:

If everything worked right, it should look exactly like your drawing. Keep this in mind when we discuss practical applications of low-level recording below.

Something else worth noting before we end this example is that the journal (the low-level ASCII file) that you just recorded is displayed on the left, below where you would normally see verification points:


You can double-click the journal file to see the source for the low-level recording.

## Practical Applications of Low-Level Recording

The demonstration I've just guided you through should make it clear that low-level recording is the method to use if you're interested in accurately capturing handwriting and drawing. The main application for this is in automating basic tasks when testing CAD/CAM programs, graphics software packages, commercial 3D games, and programs for PDAs as well as other handwriting recognition devices for which there are emerging markets, such as tablet PCs, security devices, and cell phones. Without a doubt, more and more software will be developed for such devices in the future, and all of this software will need to be tested.

Take software for PDAs alone. According to IDC, an information technology market intelligence firm, there will be 1.54 billion users of PDAs and other smart devices by 2004. This means that if you're in software development, it's virtually inevitable that you've already looked at, or shortly will be looking at, testing your software's integration with some sort of PDA version of the software or some other software yours will integrate with. My last job was with a small nonprofit community mental health center, and even there we were looking ahead to integration with PDAs and eventually creating custom software for various PDA platforms.

As you experienced above, if you were to record a script for handwriting recognition using object-oriented recording, you would quickly become discouraged and be forced to live a boring life of manual testing. But using our newly found low-level scripting capability, we can create a library of low-level scripts that capture handwriting characters and commands for popular PDA fonts like Graffiti and Jot. We can then import these to any test that needs to use handwriting recognition.

Figure 2 is an example of using such a library, this one of script files representing the characters we used above. I recorded each character individually as I wrote it on the VTech Helio PDA. (You can download the Helio Emulator and IDE from VTech's Helio PDA Web site.)

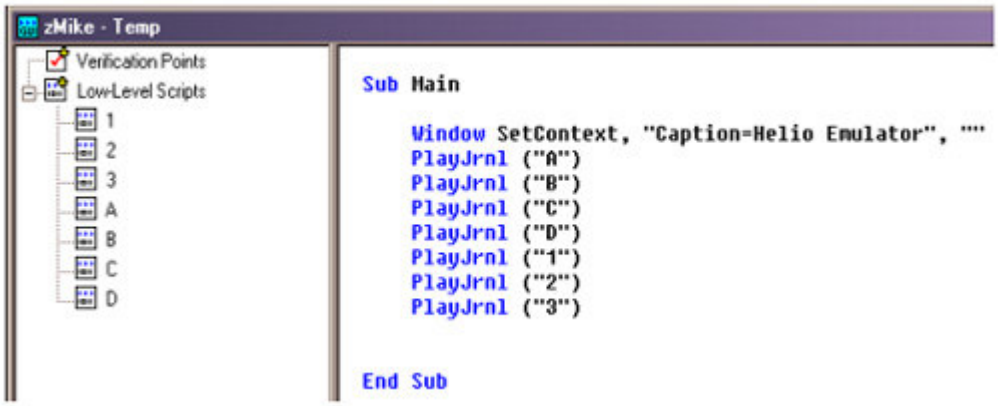



***Figure 2:*** *A script that uses a library of handwriting characters diagram.*

If I play the script back, I get the results shown in Figure 3. This shows that the application recognizes the graffiti I drew by hand.

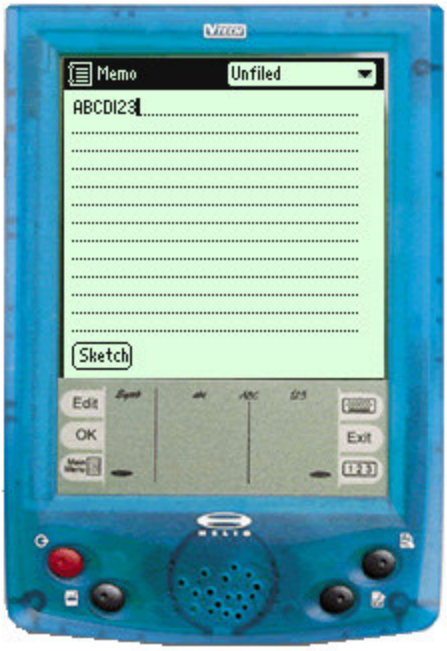



***Figure 3:*** *Results of playing back the sample script diagram.*

Even if you never use low-level scripting for handwriting recognition or graphic design, you may find yourself using it to develop a workaround for an object Robot just won't recognize. I've had to do this for some nonvisual objects that have been used by developers as well as for common objects that have been extended — a JavaTable with custom features, for example.

## Limitations of Low-Level Recording

Now that I've built up low-level recording and shown you how to use it, I should also tell you about its limits and pitfalls. Low-level recording isn't the silver bullet of software automation. It should be used somewhere between never and as sparingly as possible for most GUI applications where object-oriented recording works just fine.

The dependency of low-level recording on screen resolution and the position of the window for the application-under-test opens a huge opportunity for error. In the PDA example above, had I moved my emulator window two inches to the right or executed the test on a computer with a different resolution, nothing would have been written to the emulator. This seemingly small hurdle shouldn't be underestimated, since window positions will inevitably change, and as your number of test machines grows and your hardware changes, your resolutions will unavoidably change as well.

Another point against low-level scripting is the level of rework required when graphical changes are made to the application-under-test. With the Helio PDA, when a letter is written on the centerline in the writing area (circled in red in Figure 4), the letter is capitalized. If the letter is written to the left of that line, it's lowercase. If requirements change and now lowercase is on the line and uppercase is to the left of the line, all of the journal files in every regression script will need to be changed. For a project of any size, this news would probably not be warmly received by the project manager.


***Figure 4:*** *The centerline in the Helio PDA writing area diagram.*

And finally, low-level scripting isn't the most efficient of scripting methods. Timing and data are both hard-coded into your script. If you want any level of reuse, like that offered by data-driven techniques such as datapools or spreadsheets, you'll need to parse the ASCII file at run time and update it real-time. Although I've never tried this, I'm sure it's both possible and insanely not worth the effort. Further,

it can be problematic to reuse a journal file. A technique suggested by Brian Bryson is to put one journal file in a script and then call the script whenever you need to execute that low-level script.

## Roundup

That's low-level scripting at a glance. I encourage you to spend a few more minutes familiarizing yourself with it. Create some scripts and compare them to object-oriented scripts. Then consider low-level scripting to be like a pipe wrench in your toolbox. If you're a plumber (in this case one of the few and the proud who need low-level scripting), you'll use it all the time. If you're not, you'll just take it out every now and then and use it in some odd way it was never intended for but that still achieves the desired result.

## About the Author

Mike Kelly is currently a programmer analyst for the Northeastern Center, Inc. He's had experience managing a software automation testing team and has been working with the Rational tools since 1999. His primary areas of interest are software development lifecycles, project management, and exploring new methods of software development. Mike can be reached by e-mail.