Using Jau

Andrew Davison

Department of Computer Engineering
Prince of Songkla University
Hat Yai, Songkhla 90110
Thailand

E-mail: ad@fivedots.coe.psu.ac.th

Date: 28th April 2014

Contents

1. Introduction, and a Note on Fragility

Jau is a smallish library which makes it easy for a Java program to invoke a separate Microsoft Windows application, and communicate with it via its GUI interface. Jau code typically creates an instance of the application, and then interacts with the application's *window*. This interaction utilizes the window's GUI controls (e.g. menus, buttons, text fields) and/or shortcut keys. The essential ideas behind Jau are shown in Figure 1.

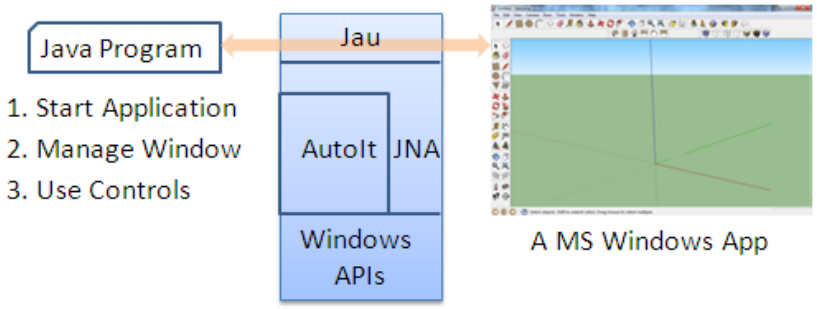

Figure 1. Jau Programming on Windows.

Jau rests on top of two powerful libraries – Java Native Access (JNA) (https://github.com/twall/jna) and the AutoItX DLL that forms part of AutoIt (http://www.autoitscript.com/site/autoit/). I'll explain the details of the connections between Jau, JNA, and AutoIt in the implementation section.

There are many examples of the 'Jau' style of programming in the download (in the "Test" and "Use" files). For example, you can easily create slides with PowerPoint, convert DOCX files to PDF with MS Word, utilize Firefox to retrieve word definitions, convert a JPG image to PNG format with XnView, play a music clip with Windows Media Player, and start building a 3D scene in SketchUp.

The obvious drawback of these astounding abilities is that the Java code is hardwired to a particular application (e.g. Word, XnView) and to a particular OS (varieties of MS Windows).

A less obvious problem is the inherent "fragility" of this kind of window-based communication. There's a likelihood that such code will break at runtime. For example, communication in a Java-PowerPoint program may be progressing wonderfully, with Java sending text and commands to the PowerPoint window to build a slide. Suddenly the user clicks on another window on the Desktop, causing the OS'es window focus to change. Unfortunately, most Jau programs assume that the focus won't change after the application window has become active. This means that the Java program's text and commands will now be delivered to another window without an alarm being raised.

Even without user intervention problems, the behavior of a GUI, especially when an application first starts, can be hard to handle programmatically. For instance, What should a Jau program do when a "Do you want to update this software?" dialog appears, or an information window pops up? The appearance of these windows will change the OS'es focus, and the Jau program will lose its link to the application window.

The problem of windows appearing at random and so changing the focus isn't limited to the application invoked by Jau. Any other program currently running in the OS can trigger such changes.

Consequently, I wouldn't recommend that Jau be used for mission-critical applications, such as nuclear reactor monitoring, death-ray controllers, or deep-space command systems. If there's a Java API available for an application, then it's much better to use that. However, if you don't mind a string, brown paper, and glue approach to programming, then Jau's for you!

2. Jau's Features

If you've read this far, then you're clearly a programmer who enjoys risks. What does Jau offer the programming thrill-seeker?

The 200 odd static methods in the Jau class can be broadly divided into six groups:

- **Running Applications**. There are several varieties of the Jau.run() method. It asks Windows to start a specified program, waits for a window to appear, and returns that window's handle as its result. Subsequent communication with the window and its controls all use this handle. See the TestRun.java program in the download for examples.
- **Application Locating**. Jau.run() often only needs the programmer to supply the name of the application (e.g. "notepad"), and it'll do the rest. However, it's sometimes necessary to use the methods in this group to find the correct name or path for an application. For instance, Jau.run() requires the string "winword" to start MS Word. See TestApps.java for examples of these methods.
- **Window Manipulation**. Once an application window has appeared, the methods in this group become helpful for manipulating the window's state, size, and position. One important method is Jau.send() for sending text and control keys to a window. See TestWin.java and TestDesktop.java.
- **Using Controls**. Just about every application utilizes GUI controls, such as buttons, menus, lists and tree views, and the status bar. The methods in this group allow controls to be manipulated using their control IDs. Unfortunately, there's no simple way to determine an ID at run time, so the programmer must do some detective work before writing his Jau program. An excellent tool for examining an application's controls (and other GUI properties) is WinSpy++ (http://www.catch22.net/software/winspy-17). Controls are utilized in almost all the "Use" examples.
- **The Mouse**. These methods allow Jau to click, press, move, and drag the mouse cursor, and find its current location. If an application doesn't have an interface that uses controls, or shortcut keys, then programmatically positioning and clicking the mouse is worth a try. UsePaint.java illustrates this use of the mouse.
- **Processes**. These functions return information about the processes associated with windows, most notably their process IDs. Most Jau programmers will probably not need these methods, since window handles (for windows) and control IDs (for controls) are normally enough to uniquely identify everything in a GUI.

A list of features is a good overview, but some small Jau examples would help to make things more concrete. But first, how is Jau installed?

3. Installing and Using Jau

Jau depends on Java (of course), and two other libraries: JNA (https://github.com/twall/jna) and AutoIt (http://www.autoitscript.com/site/autoit/)

The JNA download is packaged as two JAR files, which I typically place in d:/jna, while AutoIt has an installer that stores it in a standard location (e.g. C:\Program Files\AutoIt3).

The Jau download includes two batch files for simplifying compilation and running (called compile.bat and run.bat). They assume that the JNA binaries are in d:/jna, while the Jau class assumes that AutoIt is installed in its normal location.

Inside the Jau/ directory, the Jau and AutoItX classes are compiled like so:

```
> compile Jau.java
```

Then try compiling and running "Test" or "Use" example:

```
> compile UseNotepad3.java
> run UseNotepad3
```

4. Brief Examples

The following main() function (taken from UseNotepad3.java) starts Notepad, writes "Hello World" into its text window, waits 5 seconds, and then kills the application:

```
public static void main (String args[])
{
  HWND hWnd = Jau.run("notepad");
  Jau.send("Hello World{ENTER}");
  Jau.sleep(5000);    // milliseconds
  Jau.winKill(hWnd);
}
```

The Notepad window after the send() call looks like Figure 2.

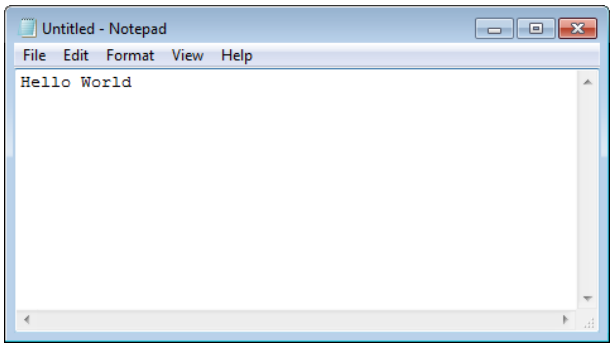

Figure 2. Hello from Notepad.

The hWnd variable holds a window *handle*, a unique identifier used to label the different entities (such as windows and controls) maintained by MS Windows. The Jau window handle type is HWND, borrowed from JNA, which is why almost every Jau program imports the com.sun.jna.platform.win32.WinDef package.

I use Jau.winKill() to force the application to terminate despite the appearance of a "Save" dialog window (like the one in Figure 3).

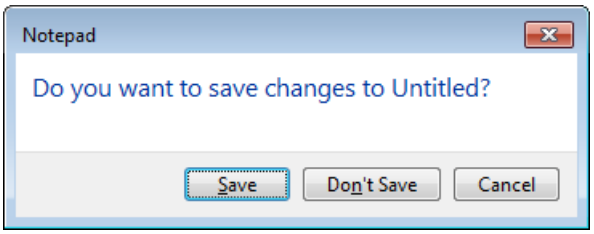

Figure 3. A Notepad Save Dialog.

Instead of using Jau.winKill(), I could employ winClose(), but then I'd have to write some code to deal with the save dialog. It must wait for this window to *possibly* appear, and then 'press' the "Don't Save" button. I say 'possibly' because the dialog will only pop up if the text inside Notepad has changed. A revised version of main() does these extra jobs:

```
public static void main (String args[])
{
  HWND hWnd = Jau.run("notepad");
  Jau.send("Hello World{ENTER}");
  Jau.sleep(5000);    // millisecs

  Jau.winClose(hWnd);
  if (Jau.winWaitActive("Notepad", 4))  // wait for up to 4 secs
    Jau.send("!n");     // select the "Don't Save" button
}
```

The Jau.winWaitActive() call waits for up to 4 seconds for a window to appear whose title begins with "Notepad" (like the one in Figure 3). Then Jau.send("!n") sends the keys <alt>-n to the active window (the "Save" dialog), which is a shortcut for selecting the "Don't Save" button. I know this because of the underlining below the "n" on that button in Figure 3.

4.1 Controls

The following UseCalc2.java example shows how to use the GUI controls inside the Microsoft calculator. The idea is to 'type' "23" into the calculator's text entry field, then a "+", then "10", a "=" to complete the calculation, and finally read the control's text to discover that 23+10 is 33!.

As soon as controls are needed in Jau, the programmer must first perform a bit of 'sleuthing' to identify the control IDs. I use WinSpy++

(http://www.catch22.net/software/winspy-17) for the task, although there are several similar tools around, including AU3Info, which comes as part of the AutoIt download. WinSpy++'s interface is shown in Figure 4.

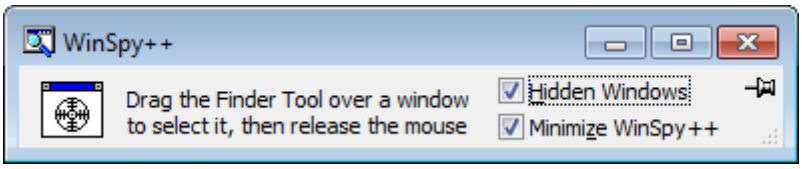

Figure 4. WinSpy++.

As the text in the window states, the user needs to drag the finder tool (the grooved circle) out of the WinSpy++ window and over the control of interest, which is highlighted as it's selected.

On my test XP machine, the calculator's highlighted input textfield looks like Figure 5.

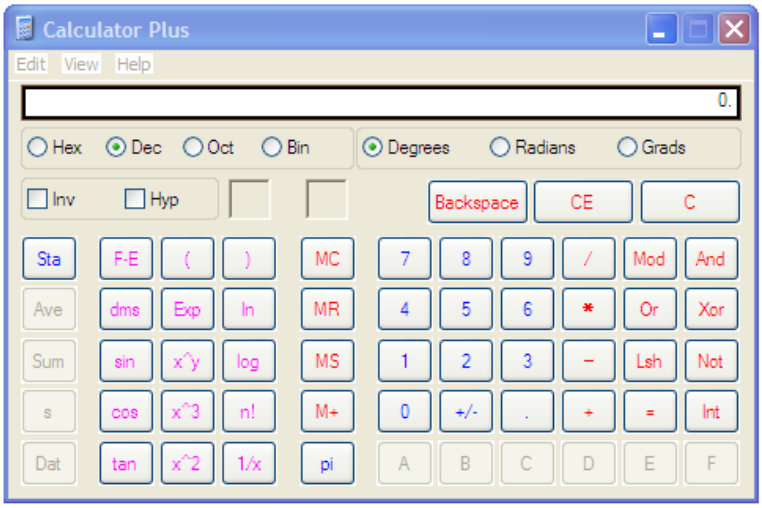

Figure 5. The Highlighted Calculator Input Textfield.

At the same time that the textfield is highlighted, WinSpy++'s "General" tab shows the information in Figure 6.

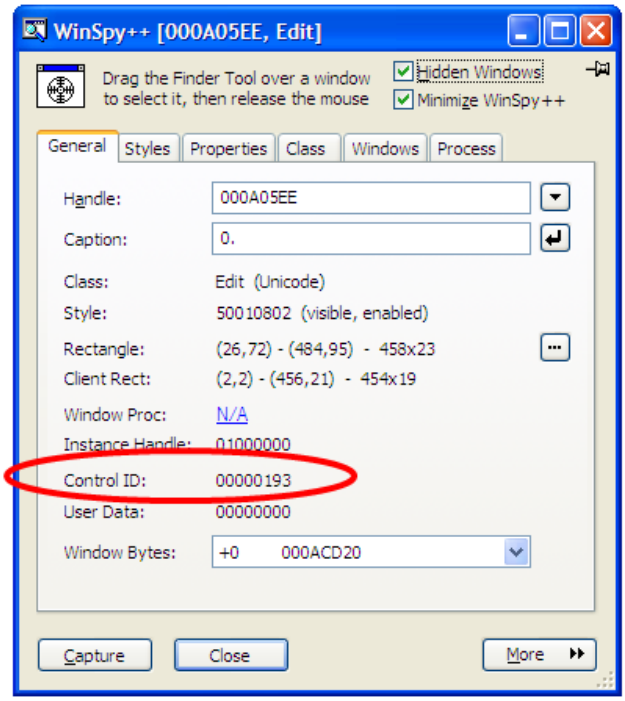

Figure 6. The WinSpy++ Information for the Calculator Text Field Control.

The red oval was added by me to highlight the important information – the control ID for the text field is the hexadecimal 0x0193.

That's all I need to write a calculator addition program, which is shown below:

```
public static void main (String args[])
{
  HWND hWnd = Jau.run("calc");
  int inCtrlID = 0x193;
                // the textfield ID in "Calculator Plus"
                // on my XP test machine

  Jau.controlSend(hWnd, inCtrlID, "23");
  Jau.controlSend(hWnd, inCtrlID, "+", 1);
                // use raw mode to send the "+"
                // without its usual special meaning
  Jau.controlSend(hWnd, inCtrlID, "10");
  Jau.controlSend(hWnd, inCtrlID, "=");

  System.out.println("Result: " +
                  Jau.controlGetText(hWnd, inCtrlID));
  Jau.winClose(hWnd);
}  // end of main()
```

Why "calc" (or, what's your name)?

One mystery about the above code is how I knew that the calculator should be referred to as "calc" in the call to Jau.run(). I used the fact that if a program name is understood by the Start menu's "Run" command, then it will be understood by Jau.run() (because their underlying implementation is the same). Several kind people have collected lists of names understood by "Run", including ones at:

- http://eightfire.com/complete-list-of-run-commands-in-windows-application-names.html
- http://mypchell.com/guides/34-guides/69-156-useful-run-commands

If you type "calc" into the "Run" dialog, then the calculator starts.

One slight exception to this correspondence is if the program is a DOS command or a Microsoft Management Console (MMC) snap-in, in which case it should be invoked with the help of the Windows command processor. For instance, you can start Windows' "Computer Management" application by typing compmgmt.msc into the Run command, but Jau.run() requires the argument "cmd /c compmgmt.msc" (as illustrated in UseComMgmt.java). In general, you should use "cmd /c" to start any ".msc" application. A nice list of snap-ins can be found at http://www.devdashnull.com/?p=192

If Jau.run() can't find an application by itself, then you'll need to point it to the correct location. For example, in UseSketchup.java, the location of SketchUp.exe is hardwired as:

```
String sketchup = Jau.appsDir() +
                    "\\SketchUp\\SketchUp 2013\\SketchUp.exe";
HWND welcomeHandle = Jau.run(sketchup);
```

Jau.appsDir() returns the default program file directory.

Sending and Reading

Another new Jau method in the calculator code is controlSend(), which usually takes three arguments – the window handle, the control ID, and the text to be sent to that ID. At the end of the program, I also employ controlGetText() to retrieve the result, 33.

This program executes so quickly, that it's useful to add a few calls to Jau.sleep() in between the control sends, to give the user a chance to see the computation progressing. For example, just before the call to controlGetText(), the calculator's textfield looks as in Figure 7.

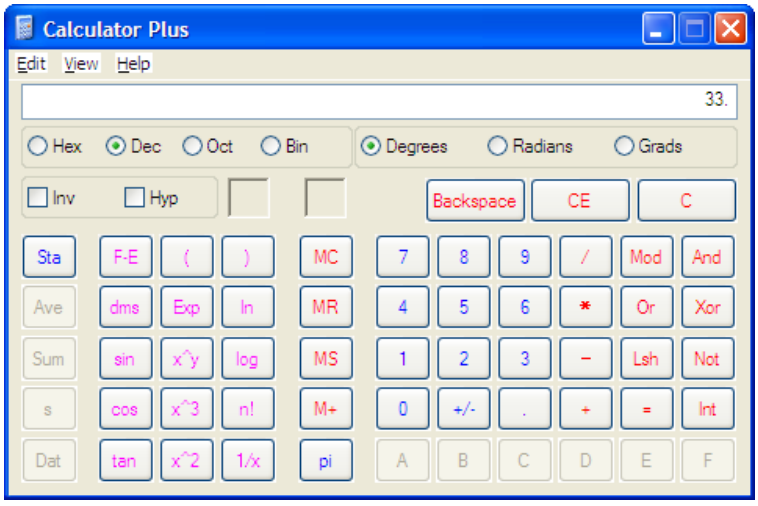

Figure 7. The Calculator showing 33

Not all Calculators are the Same

One *slight* problem with my calculator program is that is doesn't work on other versions of Windows, and probably not even on different installations of Windows XP! By "doesn't work", I don't mean a nice clean compile time error, or even a somewhat informative runtime error. The program compiles and runs – the calculator appears, and then nothing happens for a second or so before the calculator closes, and the result is reported as "null".

The reason is that my test machine is not using the calculator that originally came with XP, but "Calculator Plus", an extended version that can be downloaded for free from Microsoft (http://www.microsoft.com/en-us/download/details.aspx?id=21622).

Even if the machine still had the original version of the calculator, my code would probably still not work on other versions of Windows (such as Windows 7), since the calculator was upgraded between versions. More specifically, the IDs for its controls were changed, but the name of the program (and its path location in Windows) was not. This means that Jau.run("calc") will fire up a calculator on all versions of Windows, just not the same one. The calculator on my Windows 7 machine is shown in Figure 8.

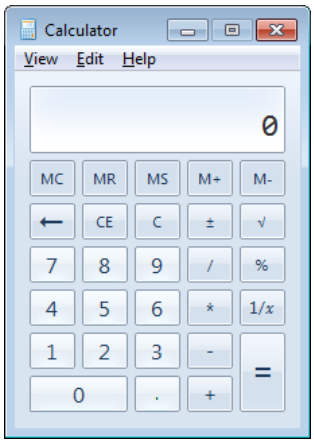

Figure 8. The Windows 7 Calculator.

If the calculator on Windows 7 is examined with WinSpy++, its control IDs are different from the controls in the XP calculator. Figure 9 shows that the input text field on the Windows 7 calculator has the ID 0x0096.

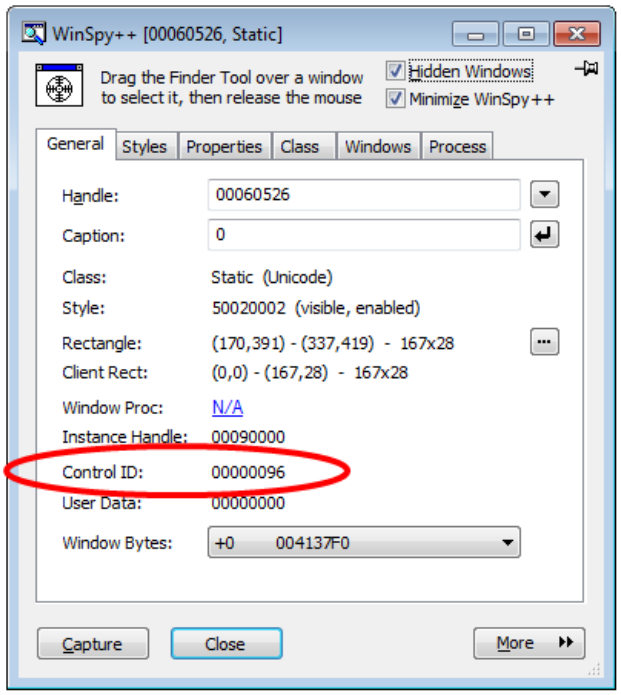

Figure 9. Calculator Input Text Field Control Information on Windows 7.

This illustrates one aspect of what I referred to as Jau's *fragility* back in section 1. Jau code may not be portable across different versions of the same OS, or even different machines with the same OS.

There are a number of possible solutions: one is to modify the code to branch to different control IDs depending on the version of Windows (which can be retrieved with Jau.windowsVersion()). The 'solution' I've actually gone for is based on examining the title of the calculator window. On my test XP machine, the calculator's title bar is "Calculator Plus" (see Figure 7), whereas on Windows 7 it's just "Calculator" (see Figure 8). The new code is:

```
public static void main (String args[])
{
  HWND hWnd = Jau.run("calc");
  String winTitle = Jau.winGetTitle(hWnd);
  System.out.println("Title: \"" + winTitle +"\"");

  int inCtrlID = 0x96;
              // in "Calculator" on my Windows 7 test machine
  if (winTitle.contains("Plus"))
    inCtrlID = 0x193;
              // in "Calculator Plus" on my XP test machine

  Jau.controlSend(hWnd, inCtrlID, "23");
  Jau.controlSend(hWnd, inCtrlID, "+", 1);
  Jau.controlSend(hWnd, inCtrlID, "10");
  Jau.controlSend(hWnd, inCtrlID, "=");

  System.out.println("Result: " +
                Jau.controlGetText(hWnd, inCtrlID));
  Jau.winClose(hWnd);
```



```
}  // end of main()
```

This Band-Aid is far from sturdy, since it doesn't address all the possible variants of Windows and calculators that might occur, but it works across all the Windows machines in my local lab, which is good enough for me ☺

4.2. Adding a Title to Jau.run()

Behind the scenes, the single-argument Jau.run() calls the AutoIt Run() function. AutoIt returns a process ID and run() waits until a window becomes active which has that process ID. The handle of that window is then returned.

Unfortunately, this approach can fail for more fancy GUI programs such as Word and PowerPoint. The problem is that some applications briefly display temporary windows before the 'work' window appears (e.g. the one containing a blank document or slide). A typical example is a splash screen or loading dialog. The temporary window may be assigned the process ID returned by AutoIt Run(), or the 'work' window may have a different ID altogether, and so the single-argument version of Jau.run() will return the wrong handle. We want Jau.run() to wait until the 'work' window appears, and return its handle, not the useless (and transient) splash screen handle.

My solution is a two-argument version of Jau.run(). The programmer now supplies the application name *and* the title of the 'work' window, and Jau waits for that window to become active. For example, Word is invoked in UseWord.java using:

```
HWND hWnd = Jau.run("winword", "Document");
```

The "Document" argument tells Jau to wait for a window whose title starts with "Document", such as "Document 1", or "Document 2". Note that the argument need only be for the start of the title, but case does matter. This matching behavior can be changed using Jau.winTitleMatchMode().

PowerPoint is started in UsePP.java using:

```
HWND hWnd = Jau.run("powerpnt", "Presentation");
```

This causes run() to wait until a window starting with "Presentation" is active on-screen.

In UseSketchUp.java, the call to SketchUp is:

```
String sketchup = Jau.appsDir() +
                    "\\SketchUp\\SketchUp 2013\\SketchUp.exe";
HWND welcomeHandle = Jau.run(sketchup, "Welcome");
```

This means that run() will ignore SketchUp's splash screen, and return the handle to its "Welcome to SketchUp" window.

4.3. Fragile? Oh, yes.

It's worth mentioning the word "fragile" again at this point. The Jau.run() approach to running applications can easily break because of the mix of installed software and/or windows already running on the Desktop. For example, consider the call to Word from above:


```
HWND hWnd = Jau.run("winword", "Document");
```

This works fine *if* there isn't already an active window on the Desktop whose title begins with "Document". If there is such a window, then there's a good chance that run() will select it, and so return a handle to the wrong window.

4.4. Looking at the Examples

Probably the easiest way to get a feel for Jau is to look at the "Test" and "Use" examples that come in the download. The "Use" examples focus on communication with a particular application

- UseCalc.java, UseCalc2.java: the Microsoft (MS) calculator and its controls;
- UseClip.java: the clipboard;
- UseComMgmt.java: the "Computer Management" MMC snap-in, using tree and list view controls;
- UseFirefox.java: utilize Firefox to access the Web;
- UseNotepad.java, UseNotepad2.java, UseNotepad3.java: simple Jau with Notepad;
- UsePaint.java: drawing with the mouse in MS Paint;
- UsePP.java: creating and saving a slide in MS PowerPoint;
- UseSketchup.java: starting SketchUp, and communicating with its Ruby console;
- UseToolTip.java: tooltips using low-level methods from AutoItX3;
- UseWMP.java: playing (or restarting) a music clip with Windows Media Player;
- UseWord.java: opening an existing DOCX file in MS Word, and saving it in PDF format;
- UseXnView.java: image conversion with XnView.

5. Jau's Implementation

Jau rests on top of Java Native Access (JNA) (https://github.com/twall/jna) and the AutoItX3 DLL that forms part of AutoIt (http://www.autoitscript.com/site/autoit/).

JNA provides Java programs with easy access to native shared libraries, such as the AutoIt DLL, and also includes wrapper classes for many Windows APIs, such as User32 and Shell32. AutoIt is a BASIC-like scripting language designed for automating the Windows GUI and general scripting. It uses a combination of simulated keystrokes, mouse movement, and window/control manipulation in order to automate tasks.

Figure 10 shows the relationships between Jau, AutoIt, and JNA.

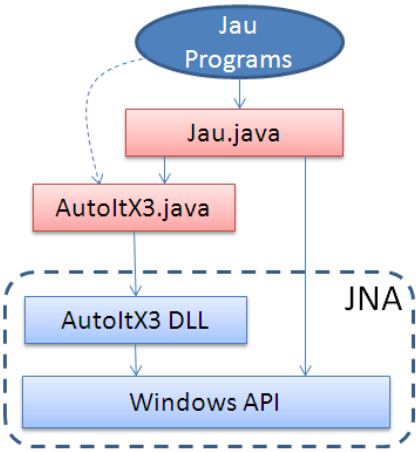

Figure 10. The Relationships between Jau, AutoIt, and JNA.

The AutoItX3 class is a JNA-implemented wrapper around the AutoItX3 DLL. The Jau class builds extra functionality on top of AutoItX3 in two ways – one is to add additional methods by communicating through JNA with other Windows APIs. Its other contribution is the simplification of the signatures of many AutoIt methods, removing (what I believe to be) unnecessary arguments, and replacing a C-like parameter style by something more Java-like. Bearing in mind that I might be wrong in my simplification choices, the original AutoIt functions are still available, via direct calls to the AutoItX3 class. Examples of this approach can be found in TestPIDs.java and UseToolTip.java.

The AutoItX3 class was partially generated with the help of the JNAerator tool (http://jnaerator.googlecode.com/) written by Olivier Chafik. My coding was also helped by examining the JWinAuto source (http://sourceforge.net/projects/jwinauto/) by Alan Richardson, which is a similar wrapper using earlier versions of AutoIt and JNA from 2007.

6. Jau Methods

This section divides the methods in the Jau class into 11 groups:

1. Application Discovery and Directories
2. Running an Application
3. Process IDs
4. Methods Involving all Executing Windows
5. Window Manipulation
6. Sending Text/Messages to a Window
7. Communicating with Controls inside a Window
8. The Clipboard
9. The Status Bar

10. Mouse Manipulation
11. Assorted Others (Probably the most important method in this group is Jau.sleep())

You might start by calling a method from group (1) to get the application's path on the machine or, more likely, just directly call Jau.run() from group (2). For simple GUI applications (such as Notepad), the bulk of the communication will be achieved with Jau.send() calls from group (6), and the application is terminated with Jau.winKill() or Jau.winClose() from group (5). This format was followed by the Notepad examples in section 4.

More complex applications, involving multiple windows and windows with GUI controls, such as buttons, menus, lists, and tree views, will need methods from groups (5) and (7). The control methods were illustrated by the calculator examples in section 4.1.

The "Test" examples in the download concentrate on testing methods from one of the 11 groups:

- TestApps.java: application discovery (group 1)
- TestRun.java: the many ways to start/run an application (group 2)
- TestPIDs.java: process related methods (group 3)
- TestDesktop.java: multiple windows and mouse manipulation (groups 4 and 10)
- TestWin.java: manipulating the application window (group 5)

The rest of this section will briefly describe the methods in each group of the Jau class. Any "Test" or "Use" examples that use a particular method are listed. Methods which are functionally very similar to AutoIt functions include a link to the Web AutoIt documentation for that function. However, bear in mind that the syntax of the Jau method and its 'corresponding' AutoIt function are often different, with the Jau version being simpler.

6.1. Application Discovery and Directories

The methods in this section are for finding an application path, which can then be used to create a window with Jau.run().

```
String appFindPath(String appName);
```

Return a single application path that matches appName, or null if nothing matches. If more than one path contains the string, then a warning is given, and the first match is returned.

File Example: TestApps.java

```
HWND appGetHandle(String appName);
```


Return the window handle for the specified running application, or null if there isn't one.

If appName matches more than one running application, then a warning is printed, and the handle of the first matching application is returned.

```
List<String> appListPaths(String appName)
```

Return all the application paths known to Windows which contain the appName string. The list that is examined comes from Windows' App Paths registry key.

File Examples: TestApps.java

```
List<String> appListPaths()
```

Return all the application paths known to Windows. The list comes from Windows' App Paths registry key.

It's very unlikely that this list will contain every application since it depends on how each one was installed. If a program is missing from the list then you'll need to explicitly include the application's path in the call to Jau.run(), perhaps with the help of Jau.appsDir().

A nice explanation of the App Paths Registry Key can be found at http://www.sepago.de/d/helge/2010/08/26/how-the-app-paths-registry-key-makes-windows-both-faster-and-safer

File Examples: TestApps.java, TestRun.java

```
String appsDir()
```

Return the path name to the default program file directory.

If you append something to this string, it's best to use "\\" as the directory separator.

e.g.
```
String sketchup = Jau.appsDir() +
                  "\\SketchUp\\SketchUp 2013\\SketchUp.exe";
          // the path for SketchUp on my Windows 7 machine
     HWND welcomeHandle = Jau.run(sketchup, "Welcome");
```

File Examples: TestApps.java, UseSketchup.java, UseXnView.java

```
String desktopPath()
```

Return the path name to the default Desktop directory.

If you append something to this string, it's best to use "\\" as the directory separator.

e.g.
```
String testFnm = Jau.desktopPath() + "\\Jau Tests\\library.jpg";
        // a file in the "Jau Tests" directory on the Desktop
```

File Examples: TestApps.java, UsePaint.java, UsePP.java, UseWMP.java, UseWord.java, UseXnView.java

```
String userDir()
```


Return the path name to the user's current working directory.

If you append something to this string, it's best to use "\\" as the directory separator.

e.g.
```
String testFnm = Jau.userDir() + "\\images\\library.jpg";
            // a file in the "images" directory in the working
directory
```

File Example: UseXnView.java

6.2. Running an Application

Jau.run() is built on top of the AutoIt Run() function (http://www.autoitscript.com/autoit3/docs/functions/Run.htm), but has simpler input arguments, and waits for a window to become active before returning its *window handle* rather than a process ID.

Jau.run() is simpler than AutoIt's Run() because it doesn't include parameters to invoke a window in hidden, minimized, or maximized form. There's also no way to redirect stdio from the window.

I've also not bothered supporting AutoIt's RunAs(), RunAsWait(), and RunWait() in Jau, although they are present in AutoItX3.java as AU3_RunWait(), AU3_RunAs(), and AU3_RunAsWait().

Jau.run()'s extended behavior (when compared to the AutoIt Run()) includes its ability to wait for an application's window to become active. It then returns the handle of the window rather than the process ID returned by AutoIt's Run(). One of the design aims of Jau is to refer to windows in only two ways – with their window handle or with their title. Process IDs are not needed in most Jau programs.

AutoIt's Run() is (most probably) implemented using the Win32 API ShellExecute() or ShellExecuteEx() functions (see http://msdn.microsoft.com/en-us/library/windows/desktop/bb776886(v=vs.85).aspx and http://msdn.microsoft.com/en-us/library/ee872121%28VS.85%29.aspx), which looks in several locations for an application, including:

- The current working directory;
- The Windows directory, although no subdirectories are searched;
- The Windows\System32 directory;
- Directories listed in the PATH environment variable;
- The AppPaths registry key, which contains a lengthy list of application path names

Occasionally ShellExecuteEx() fails to find a application listed on the AppPaths key, probably because the supplied application name isn't an exact match, especially since it treats case as being significant. Jau.run() doesn't give up if AutoIt's Run() fails, but tries to select an application from the AppPaths key using a less exact partial match, ignoring case. This second attempt to find an application is hidden inside Jau.run(), but some information about the search is printed to standard output.

```
HWND run(String appName)
```


This call waits until an active window appears which has the process ID of the application, and the handle of that window is returned. The maximum wait time is about 5 seconds, before run() returns a null. The appName is the application name used by Window's Run command, or you can supply a path to the application.

File Examples: TestRun.java, and all the "Use" examples.

```
HWND run(String appName, int timeout)
```

A variant of the one-argument run(), where a timeout is supplied. The value should be in milliseconds.

e.g. `HWND hWnd = Jau.run("impress", 20000);`

This waits for up to 20 seconds for LibreOffice Impress to start, a delay which seems necessary when LibreOffice is *first* called on a machine.

File Example: TestRun.java

```
HWND run(String appName, String title)
```

This call waits until after the application has started **and** an active window with the specified title has appeared. The handle of the titled window is returned. The maximum wait time is about 5 seconds, before run() returns a null. The appName is the application name used by Window's Run command, or you can supply a path to the application.

File Examples: TestRun.java, UsePP.java, UseWord.java

```
HWND run(String appName, String title, int timeout)
```

A variant of the two-argument run() above, where a timeout is supplied. The value should be in milliseconds.

File Example: TestRun.java

6.3. Process IDs

It seems unlikely that a Jau programmer will need the process-related methods listed in this section, because Jau.run() returns window handles, and the other Jau methods for window and control manipulation use handles and control IDs . However, if you directly call AutoIt Run() (which is available via AutoItX3.AU3_Run()), then it does return a process ID. The same is true for the other AutoIt run functions (RunWait(), RunAs(), RunAsWait()). In those cases, the methods of this section would be useful, and a short example of that coding style is included in TestPIDs.java.

```
String processGetApp(int pid)
```

Return the path of the application associated with the supplied process ID , or null if no application cannot be found.

File Example: TestPIDs.java


```
HWND processGetHandle(int pid)
```

Return the handle of the window linked to the supplied process ID , or null if there isn't one. This method gives you a simple way to change the PID returned by AutoIt Run() into a handle useful for Jau methods. TestPIDs.java shows how this is coded, which has a subtle need for a call to Jau.sleep():

```
int pid = AutoItX3.INSTANCE.AU3_Run("calc",
                                    ".", Jau.SW_SHOWNORMAL);
                    // starts the calculator
Jau.sleep(1000);
HWND hWnd = Jau.processGetHandle(pid);
```

If the sleep() is left out then processGetHandle() may return null because the OS isn't given enough time to create a window for the process.

File Example: TestPIDs.java

```
List<Integer> processListPIDs()
```

Return a list of all the application process IDs in the system.

```
List<Integer> processListPIDs(String appName)
```

Return a list of all the process IDs associated with the specified application. You might think that one application equals one process, but that's somewhat unlikely. For example:

```
List<Integer> cpids = Jau.processListPIDs("chrome");
```

returns a list of 17 different process IDs associated with a single Chrome window (which had 7 tabs open when this line was called).

File Example: TestPIDs.java

```
Map<Integer,String> processMapApps()
```

Return a map of all the processes in the system: each key is a process ID, and each value is an application name. One way of using this method is:

```
Map<Integer,String> mapApps = Jau.processMapApps();
// print the map
for (Map.Entry<Integer,String> entry : mapApps.entrySet())
  System.out.println( entry.getKey() + ": " + entry.getValue() );
```

File Example: TestPIDs.java

6.4. Methods Involving all Executing Windows

The methods in this section return information about the windows already executing on the Desktop, or manipulate multiple windows at once. One important use is to check whether an application window is already running before a new window is created (e.g. see UseFirefox.java).

```
int winCascadeAll()
```

Reposition and resize all the windows on the Desktop into cascading order.

```
List<String> winListApps()
```

Return the application paths for every window on the Desktop.

File Example: TestDesktop.java

```
List<String> winListApps(String appName)
```

Return the application paths for every window on the Desktop that contain the appName string.

```
List<HWND> winListHandles()
```

Return the handles for every window on the Desktop.

File Example: TestDesktop.java

```
List<HWND> winListHandles(String titlePart)
```

Return the handles for every window on the Desktop that contain the titlePart string somewhere in their titles.

File Example: TestDesktop.java

```
List<String> winListTitles()
```

Return the titles of every window on the Desktop.

File Example: TestDesktop.java

```
Map<HWND,String> winMapApps()
```

Return a map for all the windows on the Desktop, where each key is a window handle and its value is the window's application path.

```
void winMinimizeAll()
```

Minimize all the windows on the Desktop. Often paired with winMinimizeAllUndo().

File Example: TestDesktop.java

```
void winMinimizeAllUndo()
```

Unminimize all the windows on the Desktop. Often paired with winMinimizeAll().

File Example: TestDesktop.java

```
int winTileAll()
```


Reposition and resize all the windows so they are tiled across the Desktop.

6.5. Window Manipulation

There are so many window-related methods, that their names are summarized in Table 1.

handleString	isWindow	winActivateBottom	winActivate	winActive
winCaretCoord Mode	winClose	winEnable	winEnabled	winExists
winFlash	winGetApp	winGetCaretPos	winGetClientSize	winGetDesktop
winGetForeground	winGetHandle	winGetPos	winGetProcess	winGetState
winGetText	winGetTitle	winHide	winIsApp	winKill
winMaximize	winMaximized	winMenu SelectItem	winMinimize	winMinimized
winMove	winResize	winRestore	winSetOnTop	winSetState
winSetTitle	winSetVisibility	winShow	winTitleMatchMode	winToBack
winToForeground	winVisible	winWait	winWaitActive	winWaitClose
winWaitNotActive				

Table 1. Window Method Names.

Often the same name is used for several different methods which perform variants of the same action. Almost all the window methods come in two versions, one where the window handle is used to identify the window, and one where the window's title is employed. Consider the winActivate() method, which can active a specified window using:

```
Jau.winActivate(hWnd);
```

or

```
Jau.winActivate("Notepad");
```

This is quite a bit simpler than the naming schemes available to the AutoIt window functions, as described in the "Advanced Windows Description" section at http://www.autoitscript.com/autoit3/docs/intro/windowsadvanced.htm. In fact, the AutoIt naming techniques are available to Jau, being accepted by all the methods that normally take a title argument. However, I've not yet encountered a situation where window handles or titles weren't sufficient.

To further reduce the size of Table 1, I've not included methods with slightly different names that carry out similar tasks. For example the name "winMove" stands for winMove(), *and* winMoveToLeftEdge(), winMoveToRightEdge(), winMoveToTopEdge(), winMoveToBottomEdge(), and winMoveToCenter().

Methods which are functionally very similar to their AutoIt counterparts include links to the relevant AutoIt descriptions at http://www.autoitscript.com.

```
String handleString(HWND hWnd)
```

Convert the window handle into a string of hexadecimal characters. This is useful for pretty printing a handle:

```
    System.out.println("Handle: " + Jau.handleString(hWnd));
```

File Examples: TestApps.java, TestDesktop.java, TestPIDs.java, UseSketchup.java

```
boolean isWindow(String title)
boolean isWindow(HWND hWnd)
```

Does the title (or handle) refer to a window?

File Example: TestWin.java

```
int winActivate(String title)
int winActivate(HWND hWnd)
```

Activate (give focus to) a window. Returns 0 if the activation fails.

See http://www.autoitscript.com/autoit3/docs/functions/WinActivate.htm

File Examples: TestDesktop.java, UseFirefox.java, UseWMP.java

```
HWND winActivateBottom()
```

Bring the window at the back of all the windows to the front, and make it active. Return its handle as a result.

```
boolean winActive(String title)
boolean winActive(HWND hWnd)
```

Check to see if the specified window is currently active.

See http://www.autoitscript.com/autoit3/docs/functions/WinActive.htm

File Example: TestWin.java

```
int winCaretCoordMode(int flag)
```

Alter the way that the caret position is calculated when winGetCaretPos() is called. The flag value can be:

- 0 (or CC_WINDOW): the position will be calculated relative to the active window;
- 1 (or CC_SCREEN): the position will be in terms of absolute screen coordinates (this is the default behavior for winGetCaretPos());

- 2 (or CC_CLIENT): the position will be calculated relative to the client area of the active window. This is the area inside the window excluding the title bar, toolbars, status bar, and scroll bars. For example, in Notepad it's the area where the text is displayed.

Example:

```
Jau.winCaretCoordMode(Jau.CC_CLIENT);
System.out.println("Caret Pos: " + Jau.winGetCaretPos());
```

For more information, see the "CaretCoordMode" table entry in the AutoItSetOption documentation at
http://www.autoitscript.com/autoit3/docs/functions/AutoItSetOption.htm and
http://www.autoitscript.com/autoit3/docs/intro/windowsadvanced.htm

File Example: UseNotepad.java

```
boolean winClose(String title)
boolean winClose(HWND hWnd)
```

This method attempts to close the specified window, returning a boolean result.

See http://www.autoitscript.com/autoit3/docs/functions/WinClose.htm

File Examples: almost every example uses winClose() (e.g. UseCalc.java)

```
boolean winEnable(String title, boolean isEnabled)
boolean winEnable(HWND hWnd, boolean isEnabled)
```

Enable or disable mouse and keyboard input to the specified window.

```
boolean winEnabled(String title)
boolean winEnabled(HWND hWnd)
```

Is the specified window enabled (i.e. can it receive mouse and keyboard input)?

```
boolean winExists(String title)
boolean winExists(HWND hWnd)
```

Does the specified window exist?. This does not mean that the window is active (i.e. in focus), for which you should use the winActive() test.

See http://www.autoitscript.com/autoit3/docs/functions/WinExists.htm

```
boolean winFlash(String title)
boolean winFlash(HWND hWnd)
```

Flash a window's taskbar rectangle when it is minimized. Useful for catching the user's eye, or irritating him.

File Example: TestDesktop.java


```
String winGetApp(String title)
String winGetApp(HWND hWnd)
```

Return the path of the application associated with this window.

File Examples: TestDesktop.java, TestWin.java, UseCalc.java

```
Point winGetCaretPos()
```

Return the coordinates of the caret in the foreground window. By default, AutoIt returns the absolute screen coordinates, but this can be changed by calling winCaretCoordMode()

See http://www.autoitscript.com/autoit3/docs/functions/WinGetCaretPos.htm

File Example: TestWin.java

```
Rectangle winGetClientSize(HWND hWnd)
int winGetClientSizeWidth(HWND hWnd)
int winGetClientSizeHeight(HWND hWnd)
```

Return the size of the window's client area. This is the area inside the window excluding the title bar, toolbars, status bar, and scroll bars. For example, in Notepad it's the area where the text is displayed.

See http://www.autoitscript.com/autoit3/docs/functions/WinGetClientSize.htm

File Example: TestWin.java

```
HWND winGetDesktop()
```

Return the handle for the Desktop window. It's equivalent to the call `Jau.winGetHandle("Program Manager")`.

File Example: TestDesktop.java

```
HWND winGetForeground()
```

Return the handle to the window currently in front of all the other windows. This may not be the active window, but usually is.

```
HWND winGetHandle(String title)
```

Return the handle of the window with the specified title. By default, the title string must match at least the beginning of the window's title, with case being relevant. This matching behavior can be adjusted by calling winTitleMatchMode().

See http://www.autoitscript.com/autoit3/docs/functions/WinGetHandle.htm

```
Rectangle winGetPos(String title)
Rectangle winGetPos(HWND hWnd)
```


Retrieve the position **and size** of a given window. The Rectangle object contains the (x, y) coordinate of the window's top-left corner, and its width and height.

See http://www.autoitscript.com/autoit3/docs/functions/WinGetPos.htm

File Example: TestWin.java

```
int winGetProcess(String title)
int winGetProcess(HWND hWnd)
```

Return the process ID associated with the window.

See http://www.autoitscript.com/autoit3/docs/functions/WinGetProcess.htm

```
int winGetState(String title)
int winGetState(HWND hWnd)
```

Retrieve the state of the window. Jau includes integer constants for the standard window states (WIN_EXISTS, WIN_VISIBLE, WIN_ENABLED, WIN_ACTIVE, WIN_MINIMIZED, WIN_MAXIMIZED).

For example, instead of using winActive() to check if a window is active, you can write:

```
if (Jau.winGetState(hWnd) & Jau.WIN_ACTIVE) == Jau.WIN_ACTIVE)...
```

See http://www.autoitscript.com/autoit3/docs/functions/WinGetState.htm

```
String winGetText(String title)
String winGetText(HWND hWnd)
```

Return the text from inside the specified window.

See http://www.autoitscript.com/autoit3/docs/functions/WinGetText.htm

File Examples: TestWin.java, UseNotepad.java

```
String winGetTitle(String title)
String winGetTitle(HWND hWnd)
```

Retrieve the *full* title from the window. The word 'full' explains the purpose of the first version of this method, since the supplied string need only match the start of the title.

See http://www.autoitscript.com/autoit3/docs/functions/WinGetTitle.htm

File Examples: TestDesktop.java, TestWin.java, UseCalc.java, UseToolTip.java

```
int winHide(String title)
int winHide(HWND hWnd)
```

Hide the specified window and activate the window which is now top-most in the z-ordering of the windows on the Desktop. Often paired with winShow().


```
boolean winIsApp(String title, String name)
boolean winIsApp(HWND hWnd, String name)
```

Does the path of the application associated with this window contain the name string?

File Example: TestWin.java

```
int winKill(String title)
int winKill(HWND hWnd)
```

Force a window to close. This is a more drastic version of winClose().

See http://www.autoitscript.com/autoit3/docs/functions/WinKill.htm

File Example: TestWin.java

```
int winMaximize(String title)
int winMaximize(HWND hWnd)
```

Maximize the specified window.

File Example: UsePaint.java

```
boolean winMaximized(String title)
boolean winMaximized(HWND hWnd)
```

Is the specified window maximized?

```
boolean winMenuSelectItem(String title, String menuItem)
boolean winMenuSelectItem(HWND hWnd, String menuItem)
boolean winMenuSelectItem(String title, String menuItem,
                                        String submenuItem)
boolean winMenuSelectItem(HWND hWnd, String menuItem,
                                     String submenuItem)
boolean winMenuSelectItem(String title, String item1, String item2,
String item3, String item4, String item5, String item6, String item7,
String item8)
boolean winMenuSelectItem(HWND hWnd,String item1, String item2,
String item3, String item4, String item5, String item6, String item7,
String item8)
```

Invoke a menu item in the specified window.

The original AutoIt function includes a menu item argument and six optional submenu items. In an attempt to simplify matters, Jau offers three versions of winMenuSelectItem(): one that takes only a single menu item argument, a second that takes a menu item and a single submenu item, and thirdly one with all possible arguments included. Unused arguments should be assigned the empty string, "".

See http://www.autoitscript.com/autoit3/docs/functions/WinMenuSelectItem.htm

File Example: UseNotepad.java


```
int winMinimize(String title)
int winMinimize(HWND hWnd)
```

Minimize the specified window. A minimized window can be restored to its original size with winRestore().

File Examples: TestDesktop.java, TestWin.java

```
boolean winMinimized(String title)
boolean winMinimized(HWND hWnd)
```

Is the specified window minimized?

```
int winMove(String title, int x, int y)
int winMove(HWND hWnd, int x, int y)
```

Move the specified window so that its top-left corner is at (x, y). Unlike the AutoIt WinMove() function, resizing is not included in this function; for that use winResize(), or the five-argument version of winMove() described next.

There are several "Edge" move functions and a "To Center" method for specialized forms of moving (see below).

See http://www.autoitscript.com/autoit3/docs/functions/WinMove.htm

File Example: TestWin.java

```
int winMove(String title, int x, int y, int width, int height)
int winMove(HWND hWnd, int x, int y, int width, int height)
```

Moves **and/or resizes** a window.

See http://www.autoitscript.com/autoit3/docs/functions/WinMove.htm

```
int winMoveToLeftEdge(String title)
int winMoveToLeftEdge(HWND hWnd)
int winMoveToRightEdge(String title)
int winMoveToRightEdge(HWND hWnd)
int winMoveToTopEdge(String title)
int winMoveToTopEdge(HWND hWnd)
int winMoveToBottomEdge(String title)
int winMoveToBottomEdge(HWND hWnd)
int winMoveToCenter(String title)
int winMoveToCenter(HWND hWnd)
```

An "Edge" function moves the specified window so that its corresponding border touches the given edge. The "To Center" method positions the center of the window at the center of the Desktop.

File Examples: TestWin.java, UsePaint.java.

UsePaint.java shows how positioning a window at the top-left can help make mouse clicking on a window control a little more reliable. The programmer can assume the window is in a specific position on-screen, which implies that the controls inside a window will (most likely) be at specific mouse positions relative to the screen.

```
int winResize(String title, int width, int height)
int winResize(HWND hWnd, int width, int height)
int winResize(String title, double screenFraction)
int winResize(HWND hWnd, double screenFraction)
```

These methods resize the specified window in two ways – the first approach uses width and height values for the window; the second employs a screenFraction argument (a value between 0.2 and 1). This is useful when a window's new dimension need to be proportional to the Desktop's dimensions. A value of 1 is the same as maximizing the window.

See http://www.autoitscript.com/autoit3/docs/functions/WinMove.htm

File Example: TestWin.java

```
int winRestore(String title)
int winRestore(HWND hWnd)
```

Restore the minimized specified window to its previous size. Used to undo the effects of winMinimize().

File Examples: TestDesktop.java, TestWin.java

```
boolean winSetOnTop(String title, boolean onTop)
boolean winSetOnTop(HWND hWnd, boolean onTop)
```

Change a window's "Always on Top" attribute.

Unfortunately, this attribute isn't the same as "Always Active". "Always on Top" means that the window cannot be covered by another window, even the currently active one (i.e. the one in focus, receiving user input).

See http://www.autoitscript.com/autoit3/docs/functions/WinSetOnTop.htm

I debated whether to implement an "Always Active" method, but it would be ugly code involving background polling of the window ordering on the Desktop with Jau.winListHandles(). It might make Jau send() communication more reliable, but it would also be quite dangerous. For example, what if the always-active application decided to become unresponsive?

```
int winSetState(String title, int flag)
int winSetState(HWND hWnd, int flag)
```

Show, hide, minimize, maximize, or restore a window. Jau includes integer constants for the different possible flag values: SW_SHOW, SW_HIDE, SW_MINIMIZE, SW_MAXIMIZE, SW_RESTORE, SW_ENABLE, SW_DISABLE.

Example:

```
Jau.winSetState(hWnd, Jau.SW_MINIMIZE);
```

Alternatively, the programmer can use winShow(), winHide(), winMinimize(), winMaximize(), winRestore(), or winEnable(), so the previously example becomes:

```
Jau.winMinimize(hWnd);
```

See http://www.autoitscript.com/autoit3/docs/functions/WinSetState.htm

```
int winSetTitle(String title, String newTitle)
int winSetTitle(HWND hWnd, String newTitle)
```

Change the *full* title of the window. This explains the purpose of the first version of this method, since the title argument need only match the start of the window's current title.

See http://www.autoitscript.com/autoit3/docs/functions/WinSetTitle.htm

```
int winSetTrans(String title, double transp)
int winSetTrans(HWND hWnd, double transp)
```

Set the transparency of a window. The transp number can range between 0 and 255, with 0 being invisible and 255 fully opaque. For something simpler, see winSetVisibility().

See http://www.autoitscript.com/autoit3/docs/functions/WinSetTrans.htm

File Example: TestWin.java

```
int winSetVisibility(String title, boolean isVisible)
int winSetVisibility(HWND hWnd, boolean isVisible)
```

Make the window visible or invisible, depending on the boolean argument. For more degrees of invisibility, use winSetTrans().

File Example: TestWin.java

```
int winShow(String title)
int winShow(HWND hWnd)
```

Show a previously hidden window, making it active. Often paired with winHide().

```
int winTitleMatchMode(int flag)
```

Alter the title matching behavior used by the Jau window methods. The flag can be:

- 1 (or MATCH_FROM_START): match the title from the start (the default behavior);
- 2 (or MATCH_ANY): match against any substring in the title;
- 3 (or MATCH_EXACT): use exact title matching;

- 4 (or MATCH_ADV): advanced mode, see the AutoIt documentation "Window Titles & Text (Advanced)" at http://www.autoitscript.com/autoit3/docs/intro/windowsadvanced.htm
- -1 to -4 : case insensitive versions of the matches described above. Their corresponding Jau integer constants are MATCH_FROM_START_NO_CASE, MATCH_ANY_NO_CASE, MATCH_EXACT_NO_CASE, and MATCH_ADV_NO_CASE respectively.

For more information, see the "WinTitleMatchMode" table entry in the AutoItSetOption documentation at http://www.autoitscript.com/autoit3/docs/functions/AutoItSetOption.htm

Example:

```
Jau.winTitleMatchMode(Jau.MATCH_ANY);
            // title match anywhere in the title
HWND hWnd = Jau.run("impress", "LibreOffice");
  // wait for a window which contains "LibreOffice"
  // anywhere in its title
```

File Example: TestRun.java

```
boolean winToBack(String title)
boolean winToBack(HWND hWnd)
```

Change the z-ordering of the specified window so it's moved behind all the other Desktop windows. The focus of the window does not change.

File Example: TestDesktop.java

```
boolean winToForeground(String title)
boolean winToForeground(HWND hWnd)
```

Change the z-ordering of the specified window so it's moved in front of all the other Desktop windows. The focus of the window does not change.

```
boolean winVisible(String title)
boolean winVisible(HWND hWnd)
```

Is the window visible?

```
boolean winWait(String title)
boolean winWait(HWND hWnd)
boolean winWait(String title, int timeout)
boolean winWait(HWND hWnd, int timeout)
```

Pause execution until the requested window exists. Usually, it's more useful to use one of the winWaitActive() methods to wait until the window exists *and* is active.

See http://www.autoitscript.com/autoit3/docs/functions/WinWait.htm


```
HWND winWaitHandle(String title)
HWND winWaitHandle(String title, int timeout)
```

Same as winWait() above, except that the method returns the handle of the window that now exists.

```
boolean winWaitActive(String title)
boolean winWaitActive(String title, int timeout)
boolean winWaitActive(HWND hWnd)
boolean winWaitActive(HWND hWnd, int timeout)
```

Pause execution until the requested window is active.

See http://www.autoitscript.com/autoit3/docs/functions/WinWaitActive.htm

File Examples: UsePaint.java, UseSketchup.java, UseWMP.java, UseWord.java, UseXnView.java

```
HWND winWaitActiveHandle(String title)
HWND winWaitActiveHandle(String title, int timeout)
```

Same as winWaitActive() above, except that the method returns the handle of the window that is now active.

File Examples: UseNotepad.java, UseNotepad2.java, UsePaint.java, UsePP.java, UseWord.java, UseXnView.java

```
boolean winWaitClose(String title)
boolean winWaitClose(HWND hWnd)
boolean winWaitClose(String title, int timeout)
boolean winWaitClose(HWND hWnd, int timeout)
```

Pause execution until the specified window no longer exists.

See http://www.autoitscript.com/autoit3/docs/functions/WinWaitClose.htm

File Example: UseNotepad2.java

```
boolean winWaitNotActive(String title)
boolean winWaitNotActive(HWND hWnd)
boolean winWaitNotActive(String title, int timeout)
boolean winWaitNotActive(HWND hWnd, int timeout)
```

Pause execution until the specified window is no longer active.

See http://www.autoitscript.com/autoit3/docs/functions/WinWaitNotActive.htm

```
Rectangle getScreenBounds()
```

Return the default monitor's screen bounds, which includes its top-left coordinate (which is usually (0, 0)), and its width and height.

6.6. Sending Text/Messages to a Window

```
void send(String sendText)
void send(String sendText, int flag)
void send(HWND hWnd, String sendText)
void send(HWND hWnd, String sendText, int flag)
```

Send simulated keystrokes to the currently active window. The versions of send() with a window handle argument make the specified window active before sending the text. The flag can be set to 1 (so-called *raw mode*) to switch off the special meaning of certain characters.

send() is intended for window communication; use controlSend() for sending text to a control.

send() (and controlSend()) are inherently unreliable because they don't check that the characters in the text are going to the correct window. Even the version of send() which includes a window handle only guarantees that the specified window is active at the *start* of the communication. It is entirely possible that the currently active window can change during the send, because of user or OS intervention. The rest of the message will then be sent to the wrong place. The best we can do is to make each message short, and hope that there isn't a window focus change during its transmission.

The AutoIt documentation for Send() at http://www.autoitscript.com/autoit3/docs/functions/Send.htm includes a full list of the special characters, and numerous examples. Another great source of tips on Send() is at http://www.autoitscript.com/autoit3/docs/appendix/SendKeys.htm

File Examples: TestDesktop.java, TestWin.java, and almost all the "Use" examples.

6.7. Communicating with Controls inside a Window

The control-related methods names are summarized in Table 2.

controlClick	controlCommand	controlFocus	controlGetFocus	controlGet ClassName
controlGetHandle	controlIDDescriptor	controlIDsList	controlIDFromHandle	controlGetPos
controlGetText	controlListView	controlSend	controlSetText	controlTreeView

Table 2. Control Method Names.

Often the same name is used for different methods which perform variants of the same action. Most control-related methods have at least two variants, one where the

control is identified using the window handle and control ID, and another where the window title and control ID are employed. For example:

```
Jau.controlClick(hWnd, 0x0001);
```

and

```
Jau.controlClick("Welcome", 0x0001);
```

To further reduce the size of the table, I've not included variants of a method with slightly different names. For example the name controlGetPos represents the controlGetPos() methods *and* controlGetPosX(), controlGetPosY(), controlGetPosWidth(), and controlGetPosHeight().

One of Jau's aims is to use only control IDs to name controls. The easiest way of finding the ID for a given control is to use a GUI examination tool such as WinSpy++ (http://www.catch22.net/software/winspy-17) or AU3Info , which comes as part of the AutoIt download. I prefer WinSpy++, which seems a bit simpler to use, and a little better at identifying controls inside Window 7 GUIs. For instance, Figure 11 shows WinSpy++ information for the input text field in the Window 7's calculator (its control ID is 0x0096, circled in red). This calculator field cannot be identified by AU3Info.

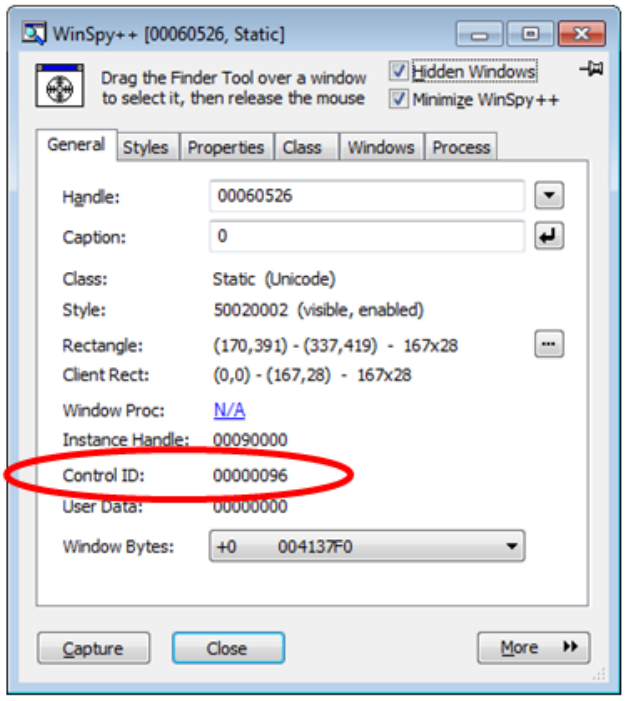

Figure 11. Control ID Information Using WinSpy++.

One important difference between WinSpy++ and AU3Info is that WinSpy++ reports control IDs in hexadecimal, while AUI3Info shows them as decimals. Jau methods will accept IDs in either form, but a hexadecimal must be preceded by '0x' (e.g. see UseCalc.java).

The AutoIt control functions can identify controls in several ways, apart from with IDs, as explained at http://www.autoitscript.com/autoit3/docs/intro/controls.htm. If you want to use this more versatile naming scheme then you can access the AutoIt

control methods via the AutoItX3 class. For instance, the Jau AutoItX3.AU3_ControlGetText() method is equivalent to AutoIt's ControlGetText() function.

```
boolean controlClick(String title, int ctrlID)

boolean controlClick(HWND hWnd, int ctrlID)

boolean controlClick(String title, int ctrlID, String button)

boolean controlClick(HWND hWnd, int ctrlID, String button)

boolean controlClick(String title, int ctrlID, String button,
                                     int numClicks, int x, int y)

boolean controlClick(HWND hWnd, int ctrlID, String button,
                                     int numClicks, int x, int y)
```

Send a mouse click command to a given control in a given window.

The first version of this method, assumes that the left button is being clicked once. The second version allows the button to be specified, which can be the string "left", "right", "middle", "main", "menu", "primary", or "secondary", or the Jau constant BTN_LEFT, BTN_RIGHT, BTN_MIDDLE, BTN_MAIN, BTN_MENU, BTN_PRIMARY, or BTN_SECONDARY. The (x, y) coordinate is the position within the control where the click will occur, which is at its center by default

See http://www.autoitscript.com/autoit3/docs/functions/ControlClick.htm

File Examples: UseCalc.java, UseSketchup.java, UseXnView.java

```
String controlCommand(String title, int ctrlID, String cmd,
                                                String option)

String controlCommand(HWND hWnd, int ctrlID, String cmd,
                                               String option)
```

Send a command to a control.

See http://www.autoitscript.com/autoit3/docs/functions/ControlCommand.htm for a table of the possible commands and their options

File Example: UseNotepad.java

```
boolean controlFocus(String title, int ctrlID)

boolean controlFocus(HWND hWnd, int ctrlID)
```

Set the input focus to be the given control in a window. This is useful prior to sending keyboard information to a control, which by default goes to the control that is currently in focus.

See http://www.autoitscript.com/autoit3/docs/functions/ControlFocus.htm

File Example: UseSketchup.java

```
int controlGetFocus(String title)

int controlGetFocus(HWND hWnd)
```


Returns the control ID of the control that has keyboard focus within a specified window. The ID is returned in decimal form, but this is no problem for Jau which can accept IDs in decimal or hexadecimal.

This method is different from the same-named AutoIt function which returns a ClassNameNN string, which combines the control's type and an instance number (the NN part). Examples include "Edit1"or "Button14". A ClassNameNN string isn't much use in Jau.

Incidentally, WinSpy++ does not list ClassNameNN values, since it's an AutoIt-specific naming scheme that isn't supported by Windows. To obtain ClassNameNN details, you must use AU3Info.

See http://www.autoitscript.com/autoit3/docs/functions/ControlGetFocus.htm and http://www.autoitscript.com/autoit3/docs/intro/controls.htm.

File Example: UseCalc.java

```
String controlGetClassName(String title, int ctrlID)
String controlGetClassName(HWND hWnd, int ctrlID)
```

Return the class name of the control. Examples include "Edit" or "Button". This is not the same as a ClassNameNN string, which also includes an instance number.

See http://www.autoitscript.com/autoit3/docs/intro/controls.htm.

File Example: TestDesktop.java

```
HWND controlGetHandle(String title, int ctrlID)
HWND controlGetHandle(HWND hWnd, int ctrlID)
```

Retrieve the internal handle of a control. A control handle can be converted into a control ID by controlIDFromHandle()

See http://www.autoitscript.com/autoit3/docs/functions/ControlGetHandle.htm

File Example: TestDesktop.java

```
String controlIDDescriptor(int ctrlID)
```

Convert a control ID into an AutoIt control ID string descriptor, which has the form `"[ID:" + ctrlID + "]"`. This might be useful if you want to use the AutoIt control functions via the AutoItX3 class. A descriptor can be utilized in an AutoIt function wherever a control title appears.

For more details, see http://www.autoitscript.com/autoit3/docs/intro/controls.htm. The relevant example on that page is the clicking on a control using the string descriptor for control ID 254.

```
List<Integer> controlIDsList(String title)
List<Integer> controlIDsList(final HWND hWnd)
```


Return a list of the control IDs associated with the specified window.

File Example: TestDesktop.java

```
int controlIDFromHandle(HWND hCtrl)
```

Convert a control handle (e.g. as obtained from controlGetHandle()) into a control ID.

File Example: TestDesktop.java

```
Rectangle controlGetPos(String title, int ctrlID)
Rectangle controlGetPos(HWND hWnd, int ctrlID)
```

Retrieve the position **and size** of a control relative to its window. The Rectangle object contains the (x, y) top-left coordinate of the control *relative to its client window*, and its width and height.

See http://www.autoitscript.com/autoit3/docs/functions/ControlGetPos.htm

```
String controlGetText(String title, int ctrlID)
String controlGetText(HWND hWnd, int ctrlID)
```

Retrieve the text from a control.

See http://www.autoitscript.com/autoit3/docs/functions/ControlGetText.htm

File Examples: TestDesktop.java, UseCalc.java, UseSketchup.java

```
String controlListView(String title, int ctrlID, String cmd,
                                     String option1, String option2)
String controlListView(HWND hWnd, int ctrlID, String cmd,
                                     String option1, String option2)
```

Send a command to a ListView32 control. A list-view control can display a collection of items, arranged in various ways. For general information, see http://msdn.microsoft.com/en-us/library/windows/desktop/bb774735%28v=vs.85%29.aspx

See http://www.autoitscript.com/autoit3/docs/functions/ControlListView.htm for the different forms of command and their options.

File Examples: TestDesktop.java, UseComMgmt.java

```
boolean controlSend(String title, int ctrlID, String sendText)
boolean controlSend(HWND hWnd, int ctrlID, String sendText)
boolean controlSend(String title, int ctrlID, String sendText, int
flag)
boolean controlSend(HWND hWnd, int ctrlID, String sendText, int flag)
```

Send a string of characters to a control. The flag is used to switch on "raw mode" text sending when it's set to 1. The default is "special mode" which allows modifier keys to be used.

controlSend() is very similar to the send() method for communicating with windows, and suffers from the same unreliability problem that were mentioned in section 6.6.

See http://www.autoitscript.com/autoit3/docs/functions/ControlSend.htm and http://www.autoitscript.com/autoit3/docs/functions/Send.htm

File Example:UseCalc.java

```
boolean controlSetText(String title, int ctrlID, String txt)
boolean controlSetText(HWND hWnd, int ctrlID, String txt)
```

Set the text of a control.

See http://www.autoitscript.com/autoit3/docs/functions/ControlSetText.htm

File Example: UseSketchup.java

```
String controlTreeView(String title, int ctrlID, String cmd, String
option1)
String controlTreeView(HWND hWnd, int ctrlID, String cmd, String
option1)
```

Sends a command to a TreeView32 control. A tree-view control displays a hierarchical list of items, such as the headings in a document, the entries in an index, or the files in a folder. For general information, see http://msdn.microsoft.com/en-us/library/windows/desktop/bb759988%28v=vs.85%29.aspx

See http://www.autoitscript.com/autoit3/docs/functions/ControlTreeView.htm for the list of commands and their options.

File Example: UseComMgmt.java

6.8. The Clipboard

```
String clipGet()
String clipGet(int timeout)
```

Retrieve text from the clipboard, but return null if the clipboard is empty. The timeout version will keep polling the clipboard for up to timeout milliseconds looking for text.

See http://www.autoitscript.com/autoit3/docs/functions/ClipGet.htm

File Examples: UseClip.java, UseFirefox.java, UseNotepad.java

```
boolean clipPut(String clip)
```

Write text to the clipboard, returning a boolean indicating whether the write was successful.

See http://www.autoitscript.com/autoit3/docs/functions/ClipPut.htm

File Example: UseClip.java

```
boolean clipClear()
```


Clear the clipboard, returning a boolean indicating whether it was successful.

File Example: UseClip.java

6.9. The Status Bar

The methods in this section can access the *standard* MS status bar, which has the class name msctls_statusbar32 when selected by WindowSpy++. For general information, see http://msdn.microsoft.com/en-us/library/windows/desktop/bb760726%28v=vs.85%29.aspx

Unfortunately, most modern GUIs seem to use non-standard status bars. For example, MS Word 2010's status bar belongs to the class NetUIHWND. One popular application that does use a standard status bar is Notepad++ (http://notepad-plus-plus.org/); another is MS Paint.

```
String statusbarGetText(String title)
String statusbarGetText(HWND hWnd, int ctrlID)
String statusbarGetText(String title, int part)
String statusbarGetText(HWND hWnd, int ctrlID, int part)
```

Retrieve the text from a status bar with the specified control ID. The "part" integer indicates the part of the status bar that is to be read - the default is 1. Status bar parts are explained at http://msdn.microsoft.com/en-us/library/windows/desktop/bb760728%28v=vs.85%29.aspx

See http://www.autoitscript.com/autoit3/docs/functions/StatusbarGetText.htm

6.10. Mouse Manipulation

The button labels used by the mouse methods in this section that are "left", "right", "middle", "main", "menu", "primary", and "secondary", or the Jau constants BTN_LEFT, BTN_RIGHT, BTN_MIDDLE, BTN_MAIN, BTN_MENU, BTN_PRIMARY, or BTN_SECONDARY.

The methods utilizing the mouse cursor's coordinate will use absolute screen coordinates by default, but this can be changed by calling mouseCoordMode().

AU3Recorder is a useful tool when writing programs that require coordinate information. It comes as part of the AutoIt download in AutoIt3/Extras/Au3Record/, and translates the keystrokes and mouse clicks made by the user into AutoIt code (see the documentation at http://www.autoitscript.com/autoit3/scite/docs/AU3Recorder.html). I used this approach to generate an AutoIt script, then converted it manually into the Jau code in UsePaint.java

The methods that require a mouse wheel direction can utilize the strings "up" or "down" or the string constants WHEEL_UP or WHEEL_DOWN.

Methods involving mouse movement speed can be assigned values between 1 (fastest) and 100 (slowest), with the default being 10.


```
int mouseCoordMode(int flag)
```

This method alters the coordinate system used by the other mouse methods when calculating the cursor position. The flag can be:

- 0 (or M_WINDOW) : the coordinate system is set to be relative to the top-left corner of the active window;
- 1 (or M_SCREEN): this changes the coordinate system to be in terms of absolute screen coordinates, which is the default;
- 2 (or M_CLIENT) : the coordinate system is set to be relative to the top-left corner of the *client area* of the active window. This is the area inside the window excluding the title bar, toolbars, status bar, and scroll bars. For example, in Notepad it's the area where the text is displayed.

For more information, see the "MouseCoordMode" table entry in the AutoItSetOption documentation at http://www.autoitscript.com/autoit3/docs/functions/AutoItSetOption.htm.

Example:

```
Jau.mouseCoordMode(Jau.M_CLIENT);
System.out.println("Mouse Position = " + Jau.mouseGetPos());
     // return the mouse position relative to the client area
     // of the active window
```

```
boolean isMouseButton(String button)
```

Is the string passed to isMouseButton() a recognized label for a mouse button?

```
boolean isWheelDir(String wheelDir)
```

Is the string passed to isWheelDir() a recognized label for a mouse wheel direction?

```
boolean mouseClick(String button)

boolean mouseClick(String button, int x, int y)

boolean mouseClick(String button, int x, int y,
                                  int numClicks, int speed)
```

Perform a mouse click. In the first version of the method, the current position of the mouse cursor is used, and one mouse click is issued. The speed can vary between 1 (fastest) and 100 (slowest), with the default being 10.

See http://www.autoitscript.com/autoit3/docs/functions/MouseClick.htm

File Example: UsePaint.java

```
boolean mouseClickDrag(String button, int x1, int y1, int x2, int y2)

boolean mouseClickDrag(String button, int x1, int y1,
                                      int x2, int y2, int speed)
```

Perform a mouse click and drag operation between the two coordinates.

See http://www.autoitscript.com/autoit3/docs/functions/MouseClickDrag.htm

```
boolean mouseDown(String button)
```

Issue a mouse down event at the current mouse position. Often paired with mouseUp().

See http://www.autoitscript.com/autoit3/docs/functions/MouseDown.htm

File Example: UsePaint.java

```
int mouseGetCursorID()
```

Return the cursor ID Number for the current mouse cursor. This ID is unrelated to control IDs.

See http://www.autoitscript.com/autoit3/docs/functions/MouseGetCursor.htm for a list of the possible values.

```
Point mouseGetPos()
```

Return the current position of the mouse cursor. The value is affected by changing the mouse coordinate system with mouseCoordMode().

See http://www.autoitscript.com/autoit3/docs/functions/MouseGetPos.htm

File Examples: TestDesktop.java, UseToolTip.java

```
void mouseMoveBy(int xOffset, int yOffset)
void mouseMoveBy(int xOffset, int yOffset, int speed)
```

Move the mouse cursor by the specified offset from its current position.

```
void mouseMoveTo(int x, int y)
void mouseMoveTo(int x, int y, int speed)
```

Move the mouse cursor to the specified () coordinate. This behavior is similar to the AutoIt MouseMove() function. The location of the coordinate is affected by changing the mouse coordinate system with mouseCoordMode().

See http://www.autoitscript.com/autoit3/docs/functions/MouseMove.htm

File Examples: TestDesktop.java, UsePaint.java

```
boolean mouseUp(String button)
```

Issue a mouse up event at the current mouse position. Often paired with mouseDown().

See http://www.autoitscript.com/autoit3/docs/functions/MouseUp.htm

File Example: UsePaint.java

```
boolean mouseWheel(String wheelDir)
```



```
boolean mouseWheel(String wheelDir, int numClicks)
```

Move the mouse wheel up or down by the specified number of clicks. The default is a single click.

See http://www.autoitscript.com/autoit3/docs/functions/MouseWheel.htm

```
HWND mouseWindowHandle()
```

Return the handle of the top-most window below the current mouse position.

File Examples: TestDesktop.java, UseToolTip.java

6.11. Assorted Others

This section is for methods that don't fit in the other categories. Probably sleep() is the most important of the bunch.

```
int autoItSetOption(String option, int param)
```

Change the operation of various AutoIt functions/parameters.

Jau comes with three specialized option methods which relieve autoItSetOption() of some work – winCaretCoordMode() for caret coordinate mode changing, winTitleMatchMode() for adjusting the behavior of window title matching, and mouseCoordMode() for setting the mouse coordinate system.

See http://www.autoitscript.com/autoit3/docs/functions/AutoItSetOption.htm

```
String chars2String(char[] msg)
```

Convert a character array into a string, replacing any non-ASCII characters by spaces, and trimming white space from the beginning and end of the string. If the result is an empty string, then null is returned.

This method is used extensively inside Jau methods for converting the character arrays required by the AutoItX3 functions into strings.

```
int error()
```

Return the integer error code set by the last AutoIt function, or 0 if no error occurred.

File Example: UseClip.java

```
String getAutoItDLL()
```

Return the path to the 32-bit or 64-bit version of the AutoIt DLL.

This assumes that AutoIt has been installed at its default position in the Windows directory structure, and that the DLL is in the subdirectory AutoItX/. This method is used at Jau initialization time to link to the AutoIt DLL, so if AutoIt is in a non-standard location, then Jau will fail.


```
String hexString(int val)
```

Convert a decimal into a hexadecimal string. This method is useful when the user wishes to print a control ID in Hex format:

```
System.out.println("control ID: " + controlID +
                   "; hex: " + Jau.hexString(controlID));
```

File Example: UseCalc.java

```
boolean isWindows()
```

Is the OS some variety of MS Windows?

```
void msgBox(String title, String text)
```

Display a message dialog window. msgBox() is a simple wrapper around a Java JOptionPane.showMessageDialog() call, intended as a replacement for the most common uses of AutoIt's MsgBox() function.

File Example: UseToolTip.java

```
String registryGet(WinReg.HKEY key, String subkey, String value)
```

Given a registry key and subkey, look up the named registry value. Internally, this method uses a JNA version of Win32's registryGetStringValue() which assumes that the value is of type REG_SZ (i.e. a string). registryGetStringValue() is unable to access other types of registry value, such as a DWORD (an integer).

If you want to manipulate the registry seriously, you'll need to write wrappers for more of JNA's Advapi32Util class, such as registryGetIntValue() (see the documentation at http://twall.github.io/jna/4.1.0/).

```
void sleep(int ms)
```

The Java version of sleeping, using Thread.sleep() instead of the AutoIt Sleep() function. The sleep time is in milliseconds.

File Examples: almost every example uses sleep.

```
int toInt(String valStr)
```

Attempt to parse a string to extract an integer. This method uses Java's Integer.parseInt(), and returns -1 if the parsing fails.

File Examples: TestDesktop.java, UseComMgmt.java

```
String windowsVersion()
```

Return a string containing the OS'es name and version.

File Example:UseCalc.java

7. AutoItX3 Functions

As explained back in section 5, and illustrated in Figure 10, the AutoItX3 class is a JNA Wrapper around the AutoItX DLL.

Unlike the Jau class, AutoItX3 makes no changes to the underlying AutoIt functions. If a programmer wants to code in an AutoIt style (in Java) then the AutoItX3 class is the place to start. Of course, I'd think coding with the Jau methods is better, but it's your choice.

The AutoItX3 style of coding is utilized in the TestPIDs.java and UseToolTip.java examples.

The functions in this section are grouped into the following categories, which are ordered by their importance to Jau. For example, Jau doesn't offer wrappers for any of AutoIt's drive or pixel functions, so I've placed their AutoXItX3 versions near the end. The documentation for a function without a Jau equivalent always includes the phrase "No Jau Wrapper".

1. Running Applications (e.g. AU3_Run())
2. Processes (e.g. AU3_ProcessWait())
3. Windows (e.g. AU3_WinActivate())
4. The Send Function: AU3_Send(); a single function but very important
5. Controls (e.g. AU3_ControlClick())
6. The Mouse (e.g. int AU3_MouseClick())
7. The Clipboard (e.g. AU3_ClipGet())
8. The Status bar (e.g. AU3_StatusbarGetText())
9. Drive Mapping (e.g. AU3_DriveMapAdd())
10. Pixel Functions (e.g. AU3_PixelChecksum())
11. Assorted Others (e.g. AU3_Sleep() and AU3_ToolTip())

After browsing through the functions in this section, you may wonder why they all start with "AU3_"? This is how the functions are named inside the DLL, and those names are reused by JNA. The naming can be confirmed by using a tool such as "DLL Export Viewer" (http://www.nirsoft.net/utils/dll_export_viewer.html) to look inside the DLL. For AutoIt, there's a more informative alternative – the C headers for the library are in AutoItX3\AutoItX\AutoItX3_DLL.h. I passed this file to the JNAerator tool (http://jnaerator.googlecode.com/) to generate a first draft of the AutoItX3 class.

Another thing you may notice during your browsing is that most AutoIt functions come in two 'flavors'. It's usually possible to identify a window (or control) by a string or by a handle. For example, a window can be closed with either the AU3_WinWaitClose() function which takes a string argument or with AU3_WinWaitCloseByHandle() which requires a handle. Aside from the difference in how the window is identified, these functions do the same thing. As a consequence, similar functions like these are grouped together in the following lists.

Since there's a direct link between each of these Java methods and their original AutoIt functions, the following documentation include links to the function information at the AutoIt website.

More Documentation on AutoIt Programming

The online AutoIt documentation is excellent, and starts at http://www.autoitscript.com/autoit3/docs/. The description of the AutoIt functions begins at http://www.autoitscript.com/autoit3/docs/functions.htm but includes non-AutoItX DLL operations, such as math functions in the scripting language. A none Web-based explanation of the functions can be found in the Help files included in the AutoIt download in AutoIt3\AutoItX\AutoItX.chm (in the "COM Interface" section!! and in AutoIt3\AutoIt.chm

Lots of advice on using functions can be found in the AutoIt scripting forum at http://www.autoitscript.com/forum/

7.1. Running Applications

```
int AU3_Run(String program, String szDir, int nShowFlag);
```

Runs an external program, returning its process ID.

Successfully using AU3_Run() often requires a call to Jau.sleep():

```
int pid = AutoItX3.INSTANCE.AU3_Run("calc",
                                    ".", Jau.SW_SHOWNORMAL);
                           // starts the calculator
Jau.sleep(1000);
HWND hWnd = Jau.processGetHandle(pid);
```

If the sleep() is left out then processGetHandle() may return null because the OS isn't given enough time to create a window for the process. Another way of dealing with this is to use AU3_WinWaitActive() or AU3_WinWait() to pause execution until the application's window has become active or exists. These issues are dealt with by Jau.run() without the need for extra lines of code.

See http://www.autoitscript.com/autoit3/docs/functions/Run.htm

File Example: TestPIDs.java

```
int AU3_RunAs(String user, String szDomain, String password,
          int nLogonFlag, String program, String szDir,
                                          int nShowFlag);
```

Runs an external program under the context of a different user. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/RunAs.htm

```
int AU3_RunAsWait(String user, String szDomain, String password,
          int nLogonFlag, String program, String szDir,
                                          int nShowFlag);
```


Runs an external program under the context of a different user and pauses script execution until the program finishes. (No Jau wrapper.)

http://www.autoitscript.com/autoit3/docs/functions/RunAsWait.htm

```
int AU3_RunWait(String program, String szDir, int nShowFlag);
```

Runs an external program and pauses script execution until the program finishes. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/RunWait.htm

7.2. Processes

```
int AU3_ProcessClose(String process);
```

Terminates a named process. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/ProcessClose.htm

```
int AU3_ProcessExists(String process);
```

Checks to see if a specified process exists. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/ProcessExists.htm

```
int AU3_ProcessSetPriority(String process, int nPriority);
```

Changes the priority of a process. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/ProcessSetPriority.htm

```
int AU3_ProcessWait(String process, int timeout);
```

Pauses execution until a given process exists. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/ProcessWait.htm

```
int AU3_ProcessWaitClose(String process, int timeout);
```

Pauses execution until a given process does not exist. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/ProcessWaitClose.htm

7.3. Windows

Every AutoIt window function listed here has a Jau equivalent, which I'd recommend using instead of these.

```
int AU3_WinActivate(String title, String txt);
int AU3_WinActivateByHandle(HWND hWnd);
```


Activates (gives focus to) a window.

See http://www.autoitscript.com/autoit3/docs/functions/WinActivate.htm

```
int AU3_WinActive(String title, String txt);
int AU3_WinActiveByHandle(HWND hWnd);
```

Checks to see if a specified window exists and is currently active.

See http://www.autoitscript.com/autoit3/docs/functions/WinActive.htm

```
int AU3_WinClose(String title, String txt);
int AU3_WinCloseByHandle(HWND hWnd);
```

Closes a window.

See http://www.autoitscript.com/autoit3/docs/functions/WinClose.htm

```
int AU3_WinExists(String title, String txt);
int AU3_WinExistsByHandle(HWND hWnd);
```

Checks to see if a specified window exists. This is *not* the same as the window being active.

See http://www.autoitscript.com/autoit3/docs/functions/WinExists.htm

```
int AU3_WinGetCaretPos(WinDef.POINT point);
```

Returns the coordinates of the caret in the foreground window.

See http://www.autoitscript.com/autoit3/docs/functions/WinGetCaretPos.htm

```
void AU3_WinGetClassList(String title, String txt,
                                    char[] classes, int bufSize);
void AU3_WinGetClassListByHandle(HWND hWnd,
                                    char[] classes, int bufSize);
```

Retrieves the classes from a window.

See http://www.autoitscript.com/autoit3/docs/functions/WinGetClassList.htm

```
int AU3_WinGetClientSize(String title, String txt, RECT rect);
int AU3_WinGetClientSizeByHandle(HWND hWnd, RECT rect);
```

Retrieves the size of a given window's client area.

See http://www.autoitscript.com/autoit3/docs/functions/WinGetClientSize.htm

```
HWND AU3_WinGetHandle(String title, String txt);
```

Retrieves the handle of a window.

See http://www.autoitscript.com/autoit3/docs/functions/WinGetHandle.htm


```
void AU3_WinGetHandleAsText(String title, String txt,
                                        char[] handle, int bufSize);
```

Retrieves the handle of a window as a string.

See http://www.autoitscript.com/autoit3/docs/functions/WinGetHandle.htm

```
int AU3_WinGetPos(String title, String txt, RECT rect);
int AU3_WinGetPosByHandle(HWND hWnd, RECT rect);
```

Retrieves the position **and size** of a given window.

See http://www.autoitscript.com/autoit3/docs/functions/WinGetPos.htm

```
int AU3_WinGetProcess(String title, String txt);
int AU3_WinGetProcessByHandle(HWND hWnd);
```

Retrieves the process ID associated with a window.

See http://www.autoitscript.com/autoit3/docs/functions/WinGetProcess.htm

```
int AU3_WinGetState(String title, String txt);
int AU3_WinGetStateByHandle(HWND hWnd);
```

Retrieves the state of a given window.

See http://www.autoitscript.com/autoit3/docs/functions/WinGetState.htm

```
void AU3_WinGetText(String title, String txt,
                                        char[] retText, int bufSize);
void AU3_WinGetTextByHandle(HWND hWnd, char[] retText, int bufSize);
```

Retrieves the text from a window.

See http://www.autoitscript.com/autoit3/docs/functions/WinGetText.htm

```
void AU3_WinGetTitle(String title, String txt,
                                      char[] winTitle, int bufSize);
void AU3_WinGetTitleByHandle(HWND hWnd, char[] title, int bufSize);
```

Retrieves the full title from a window.

See http://www.autoitscript.com/autoit3/docs/functions/WinGetTitle.htm

```
int AU3_WinKill(String title, String txt);
int AU3_WinKillByHandle(HWND hWnd);
```

Forces a window to close.

See http://www.autoitscript.com/autoit3/docs/functions/WinKill.htm


```
int AU3_WinMenuSelectItem(String title, String txt,
            String item1, String item2, String item3, String item4,
            String item5, String item6, String item7, String item8);

int AU3_WinMenuSelectItemByHandle(HWND hWnd,
            String item1, String item2, String item3, String item4,
            String item5, String item6, String item7, String item8);
```

Invokes a menu item of a window.

http://www.autoitscript.com/autoit3/docs/functions/WinMenuSelectItem.htm

```
void AU3_WinMinimizeAll();
```

Minimizes all windows.

See http://www.autoitscript.com/autoit3/docs/functions/WinMinimizeAll.htm

```
void AU3_WinMinimizeAllUndo();
```

Undoes a previous AU3_WinMinimizeAll() call.

See http://www.autoitscript.com/autoit3/docs/functions/WinMinimizeAllUndo.htm

```
int AU3_WinMove(String title, String txt, int x, int y,
                                          int width, int height);

int AU3_WinMoveByHandle(HWND hWnd, int x, int y,
                                          int width, int height);
```

Moves **and/or resizes** a window.

See http://www.autoitscript.com/autoit3/docs/functions/WinMove.htm

```
int AU3_WinSetOnTop(String title, String txt, int nFlag);

int AU3_WinSetOnTopByHandle(HWND hWnd, int nFlag);
```

Change a window's "Always On Top" attribute.

See http://www.autoitscript.com/autoit3/docs/functions/WinSetOnTop.htm

```
int AU3_WinSetState(String title, String txt, int nFlags);

int AU3_WinSetStateByHandle(HWND hWnd, int nFlags);
```

Shows, hides, minimizes, maximizes, or restores a window.

See http://www.autoitscript.com/autoit3/docs/functions/WinSetState.htm

```
int AU3_WinSetTitle(String title, String txt, String szNewTitle);

int AU3_WinSetTitleByHandle(HWND hWnd, String szNewTitle);
```

Changes the title of a window.

See http://www.autoitscript.com/autoit3/docs/functions/WinSetTitle.htm

```
int AU3_WinSetTrans(String title, String txt, int nTrans);
int AU3_WinSetTransByHandle(HWND hWnd, int nTrans);
```

Sets the transparency of a window.

See http://www.autoitscript.com/autoit3/docs/functions/WinSetTrans.htm

```
int AU3_WinWait(String title, String txt, int timeout);
int AU3_WinWaitByHandle(HWND hWnd, int timeout);
```

Pauses execution of the script until the requested window exists.

See http://www.autoitscript.com/autoit3/docs/functions/WinWait.htm

```
int AU3_WinWaitActive(String title, String txt, int timeout);
int AU3_WinWaitActiveByHandle(HWND hWnd, int timeout);
```

Pauses execution of the script until the requested window is active.

See http://www.autoitscript.com/autoit3/docs/functions/WinWaitActive.htm

```
int AU3_WinWaitClose(String title, String txt, int timeout);
int AU3_WinWaitCloseByHandle(HWND hWnd, int timeout);
```

Pauses execution of the script until the requested window does not exist.

See http://www.autoitscript.com/autoit3/docs/functions/WinWaitClose.htm

```
int AU3_WinWaitNotActive(String title, String txt, int timeout);
int AU3_WinWaitNotActiveByHandle(HWND hWnd, int timeout);
```

Pauses execution of the script until the requested window is not active.

See http://www.autoitscript.com/autoit3/docs/functions/WinWaitNotActive.htm

7.4. The Send Function

```
void AU3_Send(String sendText, int mode);
```

Sends simulated keystrokes to the active window.

See http://www.autoitscript.com/autoit3/docs/functions/Send.htm **and** http://www.autoitscript.com/autoit3/docs/appendix/SendKeys.htm

7.5. Controls

Almost every AutoIt function listed here has a Jau equivalent, which I'd recommend using instead of these.

```
int AU3_ControlClick(String title, String txt, String ctrl,
                String buttonLabel, int numClicks, int x, int y);
```



```
int AU3_ControlClickByHandle(HWND hWnd, HWND hCtrl, String button,
                int numClicks, int x, int y);
```

Sends a mouse click command to a given control.

See http://www.autoitscript.com/autoit3/docs/functions/ControlClick.htm

```
void AU3_ControlCommand(String title, String txt, String ctrl,
            String cmd, String extra, char[] result, int bufSize);

void AU3_ControlCommandByHandle(HWND hWnd, HWND hCtrl, String cmd,
                 String extra, char[] result, int bufSize);
```

Sends a command to a control. The documentation explains the many different commands.

See http://www.autoitscript.com/autoit3/docs/functions/ControlCommand.htm

```
int AU3_ControlDisable(String title, String txt, String ctrl);

int AU3_ControlDisableByHandle(HWND hWnd, HWND hCtrl);
```

Disables or "grays-out" a control. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/ControlDisable.htm

```
int AU3_ControlEnable(String title, String txt, String ctrl);

int AU3_ControlEnableByHandle(HWND hWnd, HWND hCtrl);
```

Enables a "grayed-out" control. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/ControlEnable.htm

```
int AU3_ControlFocus(String title, String txt, String ctrl);

int AU3_ControlFocusByHandle(HWND hWnd, HWND hCtrl);
```

Sets input focus to a given control on a window.

See http://www.autoitscript.com/autoit3/docs/functions/ControlFocus.htm

```
void AU3_ControlGetFocus(String title, String txt,
                            char[] controlWithFocus, int bufSize);

void AU3_ControlGetFocusByHandle(HWND hWnd,
                            char[] controlWithFocus, int size);
```

Returns the classNameNN value for the control that has keyboard focus within a specified window.

See http://www.autoitscript.com/autoit3/docs/functions/ControlGetFocus.htm

```
HWND AU3_ControlGetHandle(HWND hWnd, String ctrl);

void AU3_ControlGetHandleAsText(String title, String txt,
                        String ctrl, char[] handle, int bufSize);
```

Retrieves the handle of a control.

See http://www.autoitscript.com/autoit3/docs/functions/ControlGetHandle.htm

```
int AU3_ControlGetPos(String title, String txt, String ctrl,
                                                    RECT rect);

int AU3_ControlGetPosByHandle(HWND hWnd, HWND hCtrl, RECT rect);
```

Retrieves the position **and size** of a control relative to its window.

See http://www.autoitscript.com/autoit3/docs/functions/ControlGetPos.htm

```
void AU3_ControlGetText(String title, String txt, String ctrl,
                                char[] controlText, int bufSize);

void AU3_ControlGetTextByHandle(HWND hWnd, HWND hCtrl,
                                char[] controlText, int bufSize);
```

Retrieves text from a control.

See http://www.autoitscript.com/autoit3/docs/functions/ControlGetText.htm

```
int AU3_ControlHide(String title, String txt, String ctrl);

int AU3_ControlHideByHandle(HWND hWnd, HWND hCtrl);
```

Hides a control. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/ControlHide.htm

```
void AU3_ControlListView(String title, String txt, String ctrl,
                  String cmd, String extra1, String extra2,
                  char[] result, int bufSize);

void AU3_ControlListViewByHandle(HWND hWnd, HWND hCtrl,
                  String cmd, String extra1, String extra2,
                  char[] result, int bufSize);
```

Sends a command to a ListView32 control. The documentation explains the many different commands.

See http://www.autoitscript.com/autoit3/docs/functions/ControlListView.htm

```
int AU3_ControlMove(String title, String txt, String ctrl,
                               int x, int y, int width, int height);

int AU3_ControlMoveByHandle(HWND hWnd, HWND hCtrl,
                               int x, int y, int width, int height);
```

Moves a control within a window. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/ControlMove.htm

```
int AU3_ControlSend(String title, String txt, String ctrl,
                                          String sendText, int nMode);

int AU3_ControlSendByHandle(HWND hWnd, HWND hCtrl,
                                          String sendText, int Mode);
```

Sends a string of characters to a control.

See http://www.autoitscript.com/autoit3/docs/functions/ControlSend.htm

```
int AU3_ControlSetText(String title, String txt, String ctrl,
                                          String controlText);

int AU3_ControlSetTextByHandle(HWND hWnd, HWND hCtrl,
                                          String controlText);
```

Sets the text of a control.

See http://www.autoitscript.com/autoit3/docs/functions/ControlSetText.htm

```
int AU3_ControlShow(String title, String txt, String ctrl);

int AU3_ControlShowByHandle(HWND hWnd, HWND hCtrl);
```

Shows a control that was hidden. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/ControlShow.htm

```
void AU3_ControlTreeView(String title, String txt, String ctrl,
                  String cmd, String extra1, String extra2,
                  char[] result, int bufSize);

void AU3_ControlTreeViewByHandle(HWND hWnd, HWND hCtrl,
                  String cmd, String extra1, String extra2,
                  char[] result, int bufSize);
```

Sends a command to a TreeView32 control. The documentation explains the many different commands.

See http://www.autoitscript.com/autoit3/docs/functions/ControlTreeView.htm

7.6. The Mouse

```
int AU3_MouseClick(String button, int x, int y,
                                       int numClicks, int speed);
```

Performs a mouse click operation.

See http://www.autoitscript.com/autoit3/docs/functions/MouseClick.htm

```
int AU3_MouseClickDrag(String button, int x1, int y1,
                                       int x2, int y2, int speed);
```

Performs a mouse click and drag operation.

See http://www.autoitscript.com/autoit3/docs/functions/MouseClickDrag.htm

```
int AU3_MouseDown(String button);
```

Issues a mouse down event at the current mouse position.

See http://www.autoitscript.com/autoit3/docs/functions/MouseDown.htm

```
int AU3_MouseGetCursor();
```


Returns the cursor ID Number for the current Mouse Cursor.

See http://www.autoitscript.com/autoit3/docs/functions/MouseGetCursor.htm

```
void AU3_MouseGetPos(WinDef.POINT point);
```

Retrieves the current position of the mouse cursor.

See http://www.autoitscript.com/autoit3/docs/functions/MouseGetPos.htm

```
int AU3_MouseMove(int x, int y, int speed);
```

Moves the mouse pointer.

See http://www.autoitscript.com/autoit3/docs/functions/MouseMove.htm

```
int AU3_MouseUp(String button);
```

Issues a mouse up event at the current mouse position.

See http://www.autoitscript.com/autoit3/docs/functions/MouseUp.htm

```
int AU3_MouseWheel(String direction, int numClicks);
```

Moves the mouse wheel up or down.

See http://www.autoitscript.com/autoit3/docs/functions/MouseWheel.htm

7.7. The Clipboard

```
void AU3_ClipGet(char[] clip, int bufSize);
```

Retrieves text from the clipboard.

See http://www.autoitscript.com/autoit3/docs/functions/ClipGet.htm

```
int AU3_ClipPut(String clip);
```

Writes text to the clipboard.

See http://www.autoitscript.com/autoit3/docs/functions/ClipPut.htm

7.8. The Status bar

```
int AU3_StatusbarGetText(String title, String txt, int part,
                                    char[] statusText, int bufSize);
int AU3_StatusbarGetTextByHandle(HWND hWnd, int part,
                                    char[] statusText, int bufSize);
```

Retrieves the text from a standard status bar control.

See http://www.autoitscript.com/autoit3/docs/functions/StatusbarGetText.htm

7.9. Drive Mapping

```
void AU3_DriveMapAdd(String device, String share, int flags,
                String user, String pwd, char[] result, int bufSize);
```

Maps a network drive. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/DriveMapAdd.htm

```
int AU3_DriveMapDel(String device);
```

Disconnects a network drive. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/DriveMapDel.htm

```
void AU3_DriveMapGet(String device, char[] mapping, int bufSize);
```

Retrieves the details of a mapped drive. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/DriveMapGet.htm

7.10. Pixel Functions

```
int AU3_PixelChecksum(RECT rect, int step);
```

Generates a checksum for a region of pixels. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/PixelChecksum.htm

```
int AU3_PixelGetColor(int x, int y);
```

Returns a pixel color according to the () coordinate. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/PixelGetColor.htm

```
void AU3_PixelSearch(RECT rect, int nCol, int nVar, int step,
                                              WinDef.POINT point);
```

Searches a rectangle of pixels for the pixel color provided. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/PixelSearch.htm

7.11. Assorted Others

```
int AU3_AutoItSetOption(String option, int val);
```

Changes the operation of various AutoIt functions/parameters. Same as AU3_Opt().

See http://www.autoitscript.com/autoit3/docs/functions/AutoItSetOption.htm

```
int AU3_error();
```

Gets the error code set by the last AutoIt function call, or 0 if there was no error.

```
int AU3_IsAdmin();
```

Checks if the current user has full administrator privileges. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/IsAdmin.htm

```
int AU3_Opt(String option, int val);
```

Changes the operation of various AutoIt functions/parameters. Same as AU3_AutoItSetOption().

See http://www.autoitscript.com/autoit3/docs/functions/AutoItSetOption.htm

```
int AU3_Shutdown(int flags);
```

Shuts down the system. The documentation explains the various flags. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/Shutdown.htm

```
void AU3_Sleep(int milliseconds);
```

Pauses execution. The Jau sleep() method uses Java's Thread.sleep(), not this one.

See http://www.autoitscript.com/autoit3/docs/functions/Sleep.htm

```
int AU3_ToolTip(String tip, int x, int y);
```

Creates a tooltip anywhere on the screen. (No Jau wrapper.)

See http://www.autoitscript.com/autoit3/docs/functions/ToolTip.htm

File Example: UseTooltip.java

