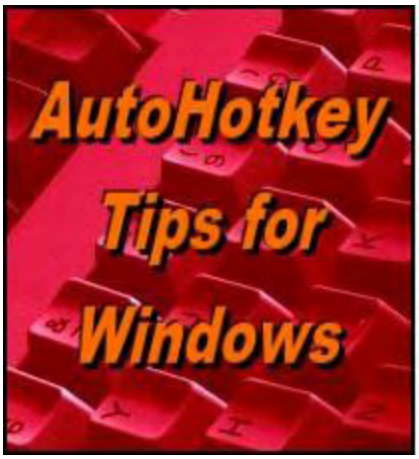


Controlling Windows Programs with AutoHotkey

"Windows Paint Is Used to Demonstrate How to Draw and Fill In a Square with AutoHotkey" by Jack Dunning

Here are some tips which will help you to both automate the drawing simple objects and control other types of applications with AutoHotkey.

This time we're taking a look at how to control an application with AutoHotkey—in particular the Windows Paint program. In the past, I've only occasionally investigated the AutoHotkey commands for controlling mouse and cursor movements. The Paint program is ideal for that type of action while giving insight into how virtually any Windows program may be controlled. The tips given here may apply to another application that you want to automate with AutoHotkey.

There may be times when you want to automatically draw a circle or a box in a paint or drawing program without doing all the tool selection and mouse movements yourself. It's quicker if you can hit a hotkey combination and a box is drawn on the screen ready for repositioning and resizing. This type of command can usually be accomplished with a short, simple AutoHotkey script.

New to AutoHotkey? See our Introduction to AutoHotkey!

Using Keyboard Shortcuts to Navigate a Program

In the past, I've used the Click command in a clumsy attempt to automate e-mail and other programs. (The MouseClick command has similar functionality to the Click command, but "the Click command is generally more flexible and easier to use.") It usually involves finding the coordinates of a button or control within the application window and using *Click* to activate it. The problem is that you must first find the coordinates (usually with Window Spy), then include them in the script. With time I found that it was easier and quicker to use the keyboard shortcuts which are usually built into Windows software. The usual key for activating software keyboard shortcuts is ALT.

Once Paint is loaded into an active window, press ALT. If the program uses a ribbon menu, little shortcut letters or numbers will appear next to the menu options, as shown in Figure 1. If the program uses the classic menu system with dropdown menus, then the first menu item will be selected and the keyboard shortcut letters will be underlined. Try pressing the ALT key while using any program. If one of the two possibilities above does not occur, then you will be forced to use other alternatives—most likely the Click command with the correct coordinates.

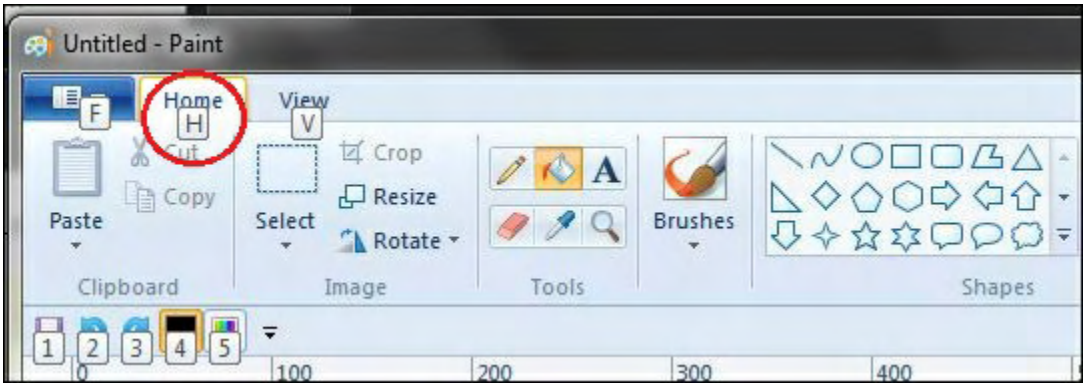


Figure 1. Pressing the ALT key pops up the keyboard shortcut letters or numbers. They appear in the little boxes next to the selection options. In this case, the Home tab is selected by next pressing the H key.

Press the H key and all of the shortcuts for the next level appear (see Figure 2). To select any of the options only requires the pressing of the corresponding letter. If the shortcut uses more than one letter (as in the case of SH for Shapes), the keys are pressed sequentially (*not* simultaneously).

Figure 2. After pressing the H key, the next shortcuts for the various controls appear. The K key selects the Fill with Color tool. The SH sequence selects the Shapes menu.

In this example, we first select the Shapes menu, then move over to and select the rectangle drawing tool. Then we will use the *Click* command to position the cursor on the drawing surface, left-click and hold, then dragging the cursor across the screen to draw a square (see Figure 3). It the script stops at this point the square can be both positioned and sized as appropriate.

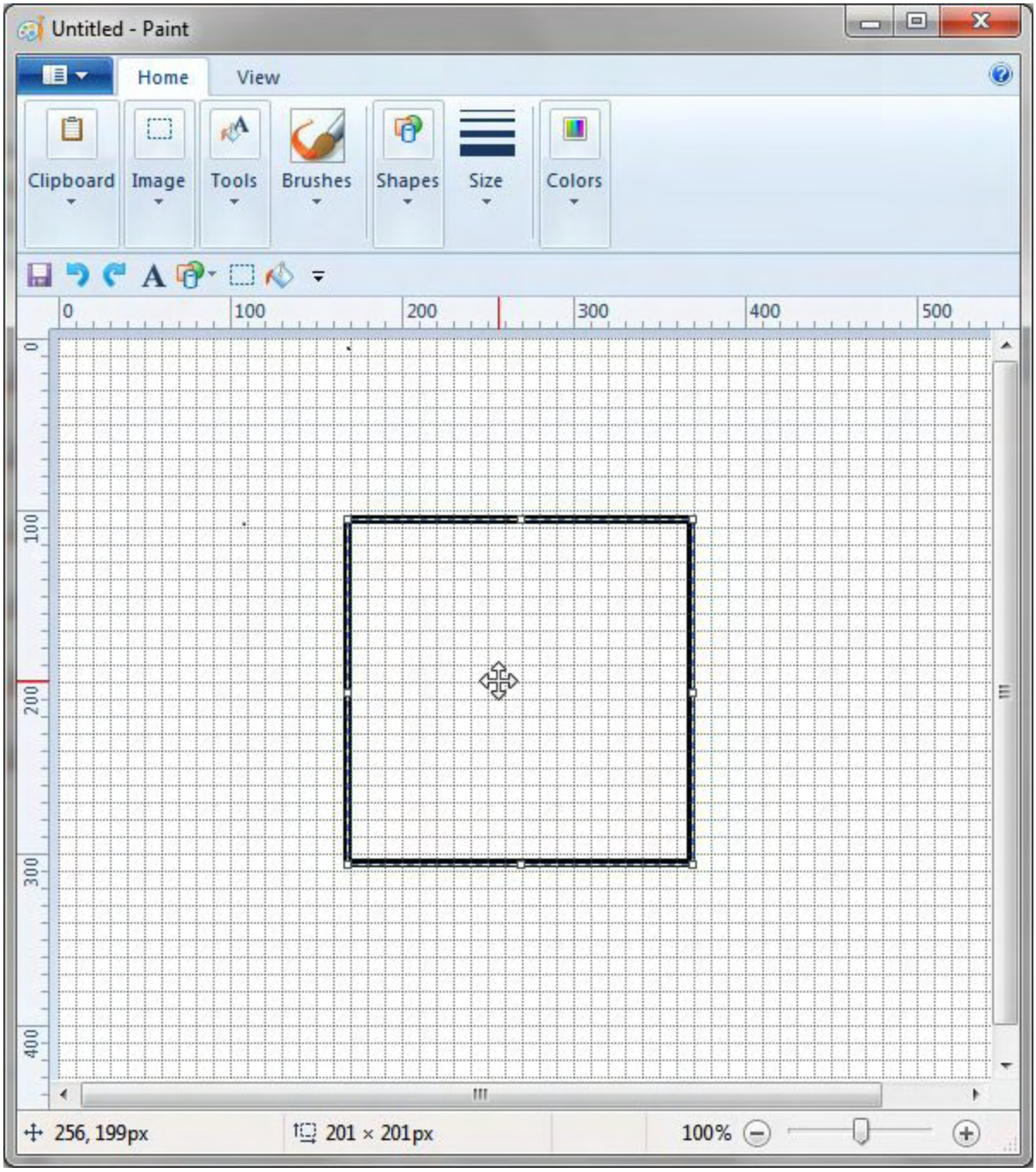


Figure 3. An AutoHotkey script selects the tools and draws a square which can then be repositioned and resized.

One of the advantages of using an AutoHotkey script rather than drawing the square by hand is that the rectangle will be exactly square without any fidgeting. Using a mouse and eyeballing it can be a little iffy—although the size coordinates do show on the bottom status bar.

The sample script then selects the Fill tool, picks an appropriate spot within the square and fills the square in with the selected color (see Figure 4).

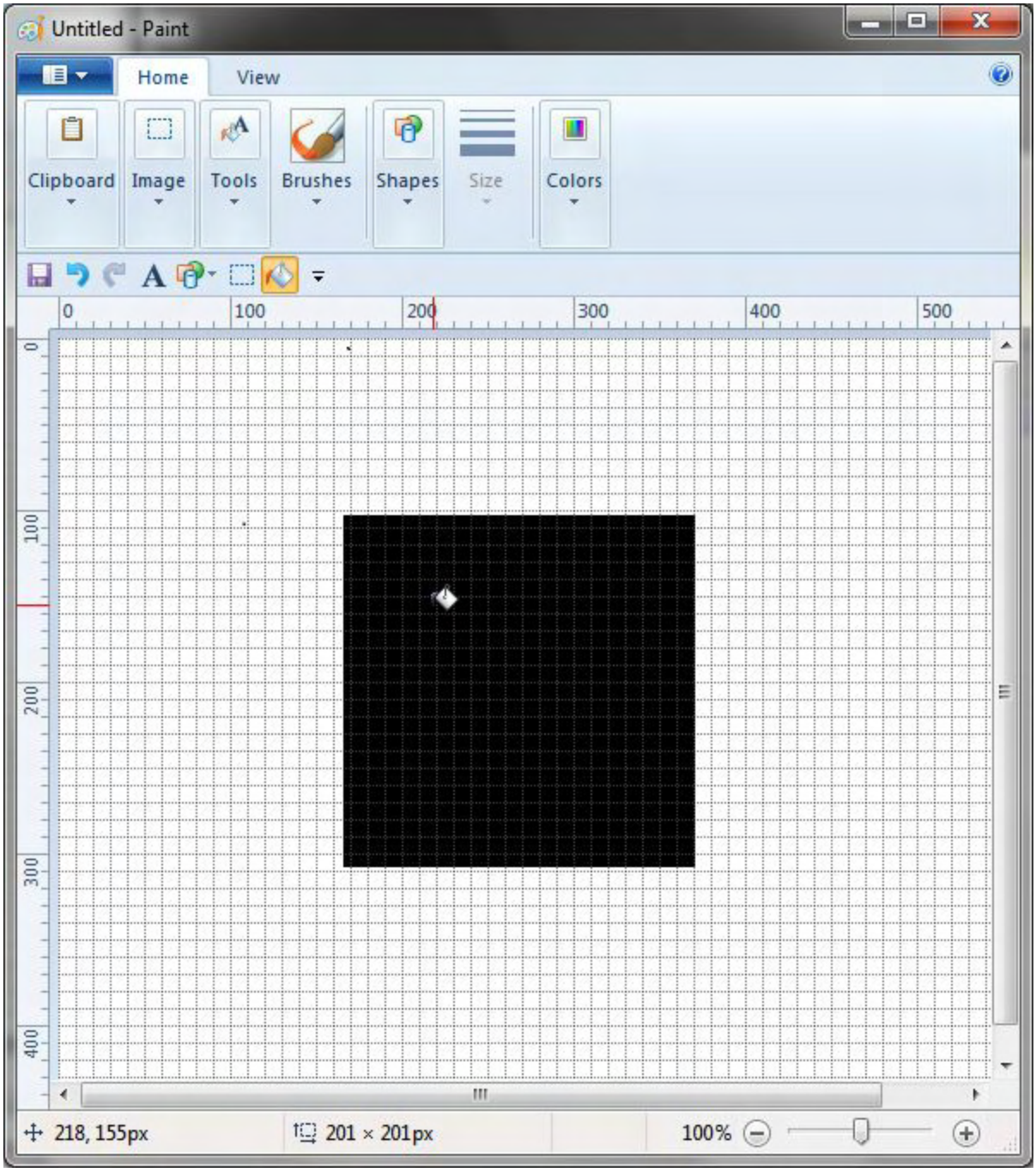


Figure 4. After the Fill tool is selected, the square is filled in with black.

How It Works

This script which draws a square box in Windows Paint, is relatively short. The only requirement is that Paint be loaded and its window active. It uses the SendInput command to execute the shortcut keys, the *Click* command to position the cursor, and the MouseMove command to drag the cursor across the screen:

```
SendInput {Alt}hsh
```

```
Sleep 100
SendInput {Right 3}{Enter}
Sleep 200
Click Down 200,300
MouseMove 200, 200, 50, R
Click Up
SendInput {Alt}hk
Sleep 200
Click 250,350
```

If you only want the box ready to reposition and resize, then cutoff the script after the *Click Up* command.

The Sleep command is strategically placed in the script to slow down the execution. Otherwise, as we've seen so many times in the past, the script may outstrip the processing of the commands and end up skipping some of them. (The time interval is a little arbitrary and may depend upon your processor speed. The rule of thumb is "Whatever works!")

The first line of the script (*SendInput {Alt}hsh*) activates the keys ALT, H, S, and H one at a time as if they are being pressed on the keyboard. This causes the Shapes tools window to be selected with the same shortcut keys. However, by default the first shape (the line drawing tool) is selected and there are no more shortcut keys. After a short pause (*Sleep 100*), other keys are sent to select the appropriate tool.

The *SendInput {Right 3}{Enter}* line is the equivalent of pressing the RIGHT ARROW key three times, then pressing ENTER. This causes the cursor to move right three position, then select the tool with ENTER. The rectangle drawing tool is now selected. Another rest with *Sleep 200*.

Now the script clicks on the drawing surface with *Click Down 200,300*. By default the *Click* command simulates a left mouse button click. The *Down* parameter tells AutoHotkey to continue holding the left-button down. The click position is determined by the *x,y* coordinates *200,300* in pixels. This position is relative to the upper left-hand corner (*0,0*) of the Paint window. (The *x* coordinate is from left to right and the *y* coordinate is from top to bottom.) While the coordinates used here are somewhat random, it is important that they are contained within the Paint drawing area.

In the next line, *MouseMove 200, 200, 50, R*, the *MouseMove* command is used to draw a square 200 pixels by 200 pixels. Since the left mouse button is already down, this simulates a mouse drag. The *50* is the parameter which controls the speed. I only selected 50 because is slows down the drawing enough to watch the square grow. The final R parameter tells AutoHotkey that the coordinates are relative to the last location of the cursor and not the upper left-hand corner of the Paint window.

The line *Click Up* releases the left mouse button. Once the square is drawn, it's time to fill it in.

Just as was done at the beginning of the script, the SendInput command is used to select the Fill tool, *SendInput {Alt}hk*. The K key is the shortcut for selecting the Fill tools after the selection of the Home tab (*ALT*, then *H*).

After waiting an appropriate amount of time, *Sleep 200*, coordinates must be selected within the newly drawn square. Clicking on that spot causes the square to be filled, *Click 250,350*. Done!

This short script can be included in a Hotkey:

```
!d::
  SendInput {Alt}hsh
  Sleep 100
  SendInput {Right 3}{Enter}
  Sleep 200
  Click Down 200,300
  MouseMove 200, 200, 50, R
  Click Up
  SendInput {Alt}hk
  Sleep 200
  Click 250,350
Return
```

This way—once the script is loaded—every time the hotkey combination ALT+D (pressed simultaneously) is used while the Paint window is active, the tools will be selected and a square drawn and filled. (The exclamation point represents ALT when used as part of a hotkey combination.) Be sure to contain the code by placing a *Return* command at the end.

Easier Shortcut Keys

Some programs (Windows Paint is one of them) allow you to move certain tools and features to a quick launch menu. In Paint this is done by right-clicking on the tool and selecting Add to Quick Access Toolbar (see Figure 5). An icon is added to the special toolbar which only requires one click for activation.

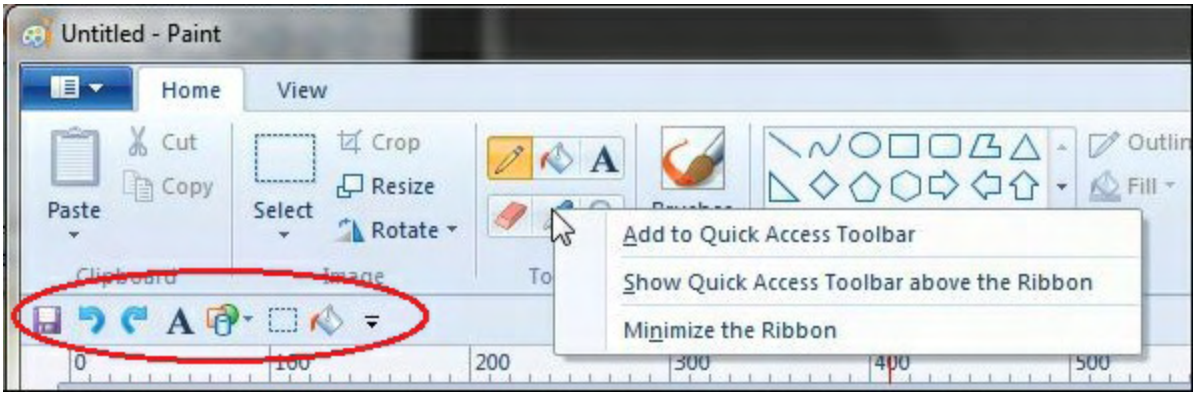


Figure 5. Right-click on a tool and select Add to Quick Access Toolbar to place the hot icon on the menu below.

Now, when the ALT key is pressed, the newly added quick launch buttons are each assigned a number (see Figure 6).

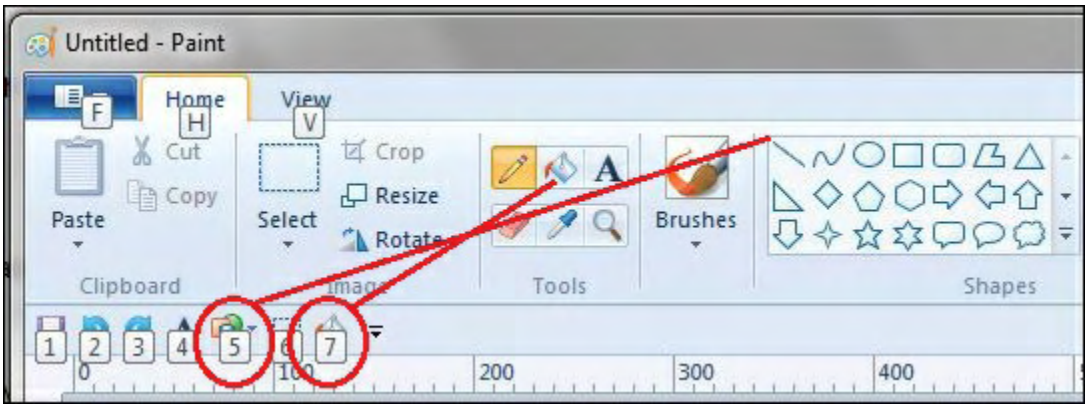


Figure 6. When the ALT key is pressed the quick launch buttons display keyboard shortcut numbers.

Now the AutoHotkey script can be modified to use less keystrokes:

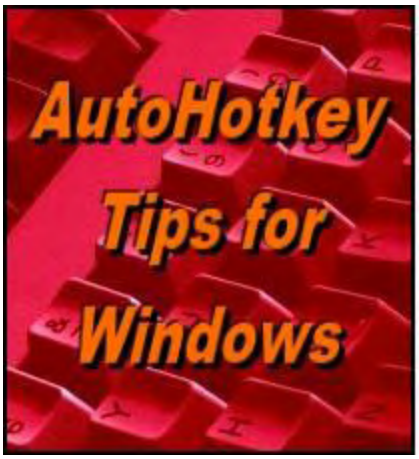
```
 SendInput {Alt}5
SendInput {Right 3}{Enter}
Sleep 200
Click Down 200,300
MouseMove 200, 200, 50, R
Click Up
SendInput {Alt}7
Sleep 200
Click 250,350
```

Notice that the *SendInput {Alt}* line only requires the one number. The upside to this quick approach is that it is faster than using more letters, therefore the *Sleep* time can be shorter. The downside is that the script will be less universal since the quick launch number assignment depends upon what order the buttons were added.

Not sure which e-book format you need for your iPad, Kindle, PC, Mac, Android or other e-book reading device? Get all three formats at once (EPUB for iPad, Android and PCs, MOBI for Amazon Kindle, and PDF for reading or printing on standard notebook size paper) for any of the AutoHotkey e-books at one special price at ComputorEdge E-Books. (Note: If something goes wrong during a download and you run out of downloads, e-mail us or gives us a call and we'll give you more downloads at no extra charge.)

* * *

Free! *AutoHotkey Tricks You Ought To Do With Windows!* This e-book includes both those tips and the reference material (Table of Contents and indexes) from the other three AutoHotkey books. Pick up a copy free and share it with your friends.



Using AutoHotkey to Draw a Straight Line in Windows Paint

"While You May Never Need to Do This, the AutoHotkey Techniques Apply to Many Other Applications" by Jack Dunning

Jack continues investigating using AutoHotkey drawing controls in Windows Paint.

Last week we looked at how Windows Paint (and most other programs) can be controlled with an AutoHotkey script by calling out the software's ALT activated shortcut keys. This time a technique for drawing a straight line with the Pencil tool is highlighted.

New to AutoHotkey? It's absolutely the best free windows utility software ever! See our Introduction to AutoHotkey!

It's not necessary to draw straight lines with the Pencil tool. In fact it is almost impossible to do by hand. The Pencil tool is for freehand drawing and not setup for reshaping as are the various shape tools. It's easy to draw a straight line with the Line shape tool, as shown in Figure 1, but that doesn't add much knowledge to our AutoHotkey tool box. The purpose here is to introduce AutoHotkey tricks which can expand our ability to control the Windows Paint tools. By using the freehand Pencil tool, all the usual user-friendly constraints are eliminated.

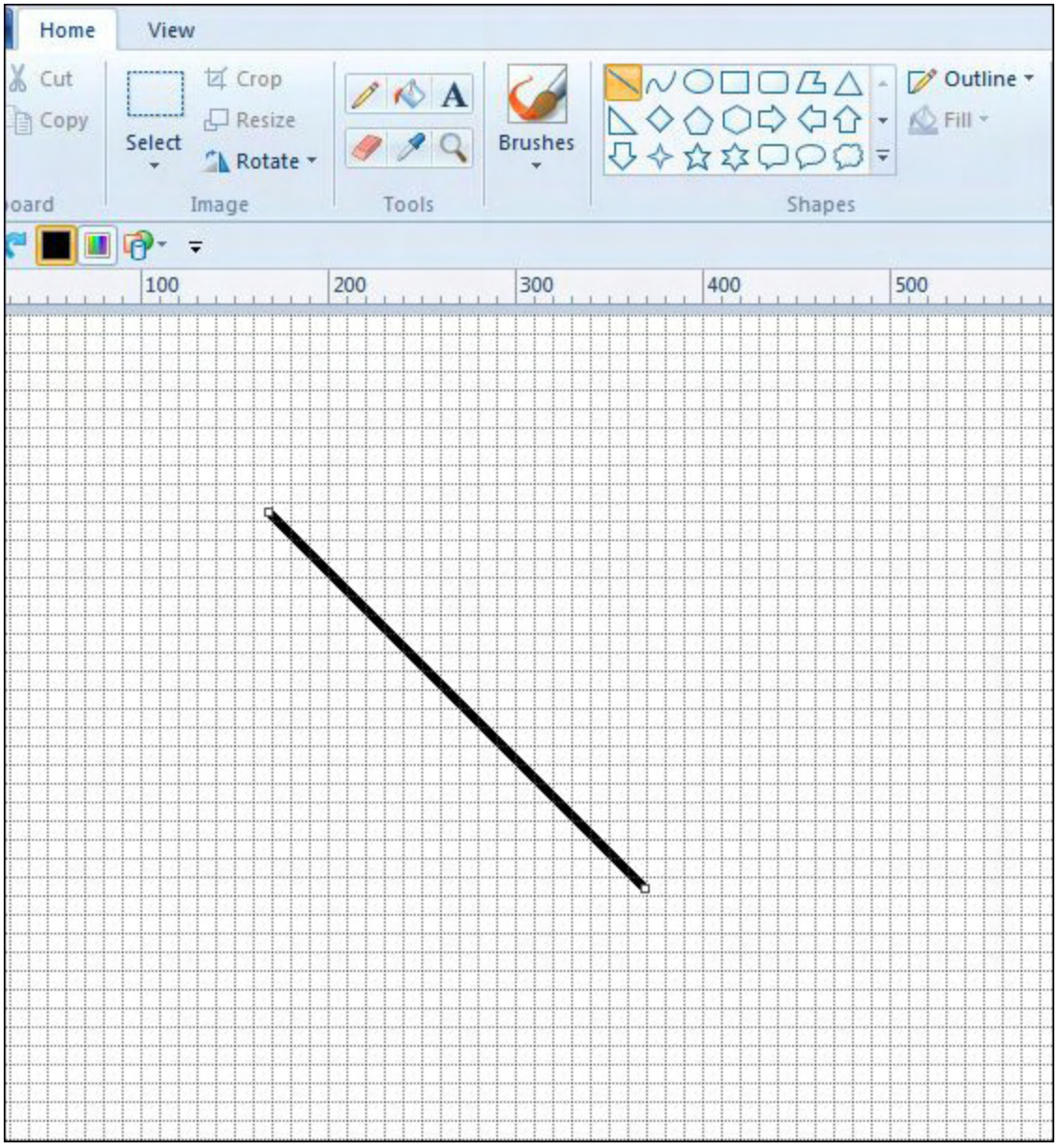


Figure 1. The Line tool is used to draw a straight line.

In this line drawing script, the only change from last week's box drawing script is the use of the HOME key to select the first line shape (the Line tool):

```
SendInput {Alt}hsh
Sleep 100
SendInput {Home}{Enter}
Sleep 200
Click Down 200,300
MouseMove 200, 200, 50, R
Click Up
```

The Line tool always draws a straight line. What kind of challenge is that?

Using the Pencil Tool to Draw a Straight Line

If a modified version of the above script is used to draw a line:

```
SendInput {Alt}hp
Sleep 100
SendInput {Home}{Enter}
Sleep 200
Click Down 200,300
MouseMove 200, 200, 50, R
Click Up
```

the Windows Paint cursor wanders a bit and the line is not straight (see Figure 2). The MouseMove command is used giving AutoHotkey the start and end point, but anywhere in between, the cursor location may randomly vary as long as it is headed for the end point.

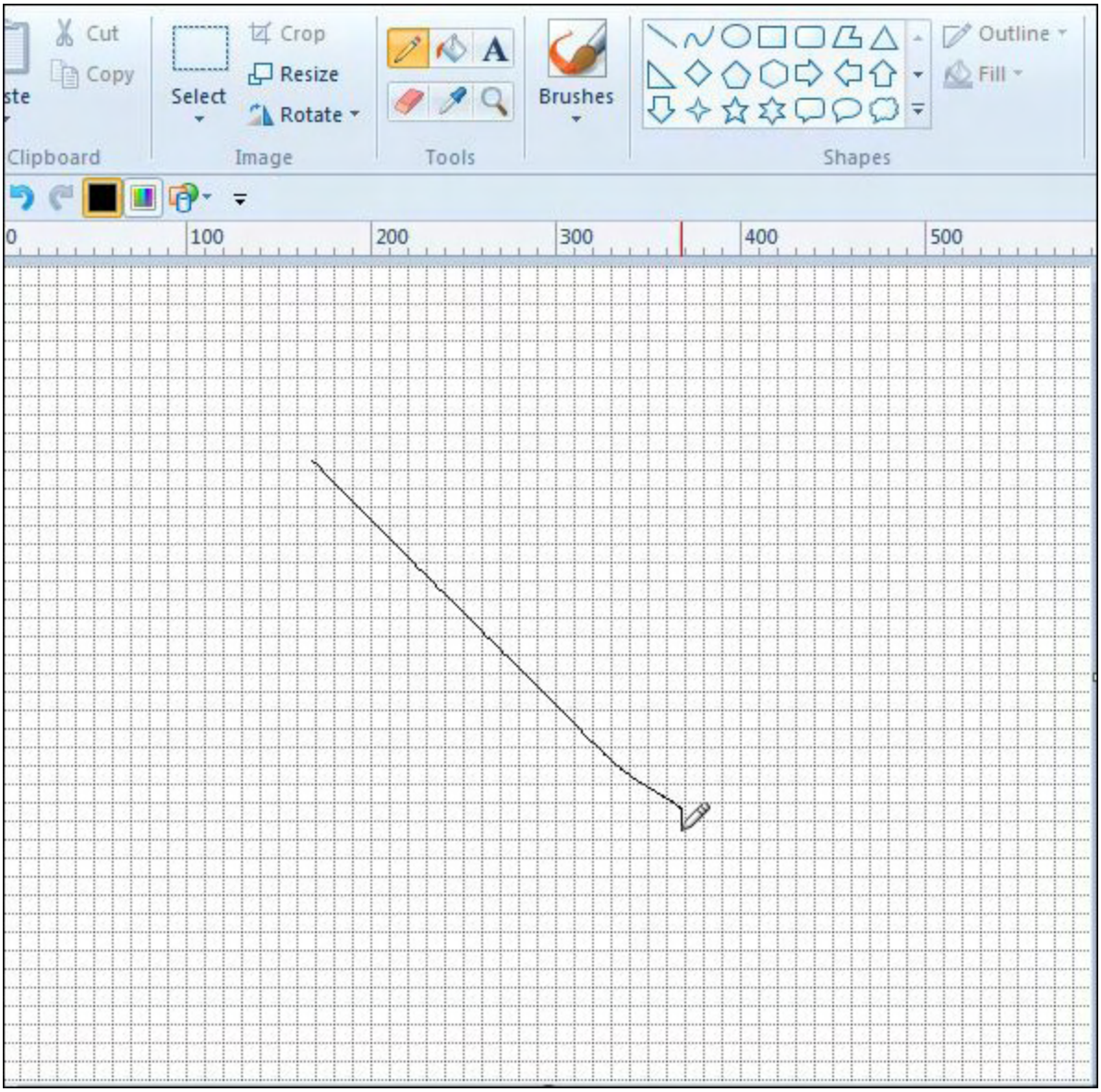


Figure 2. The first AutoHotkey script has a difficult time drawing a straight line with the Pencil tool.

The only solution is to take pixel level control and move the pencil one pixel at a time. That means issuing the *MouseMove* command 200 times. This is done by putting the command in a *Loop*.

Using Loops to Draw Lines

The major advantage to placing the *MouseMove* command inside a Loop command, is that there is absolute control over the cursor location while issuing hundreds (or thousand) of movement commands, yet writing very little code. Without the *Loop*, it would take 200 lines of code to draw the line straight—one for each pixel of movement. With *Loop* the *MouseMove* command is written only once:

```
SendInput {Alt}hp
Sleep 100
SendInput {Home}{Enter}
Sleep 200
Click Down 200,300
Loop, 200
{
  MouseMove 1, 1, 50, R
}
Click Up
```

The *Loop* command increments 200 times (*Loop, 200*) issuing *MouseMove 1, 1, 50, R* (moves only one pixel down and one to the right) on each increment. (The commands that run on each increment of a Loop command are contained between the two curly brackets *{* and *}*.) The result, a dead straight line, is shown in Figure 3.

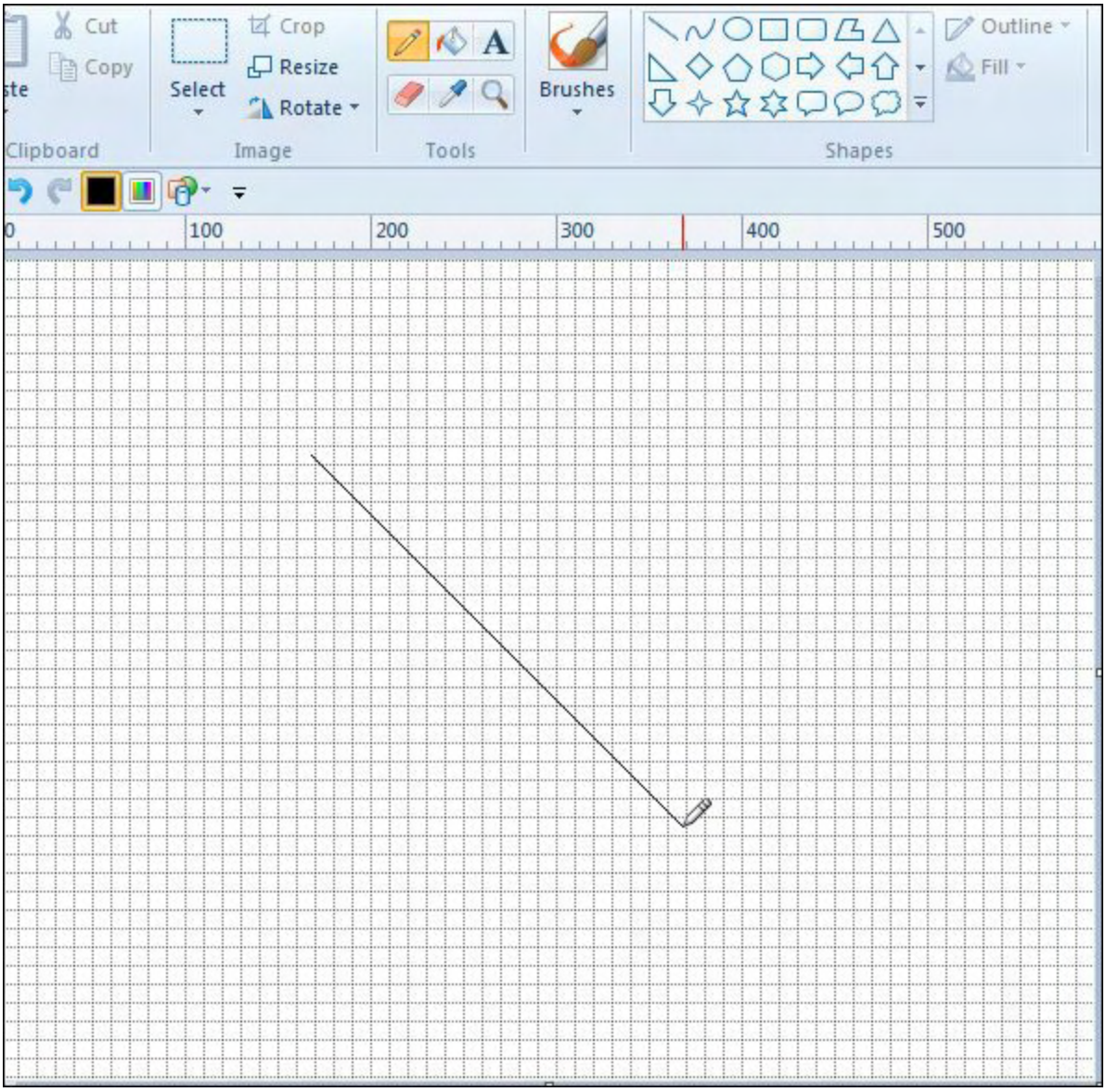


Figure 3. By adding a Loop to the AutoHotkey script the Pencil tool can draw a straight line.

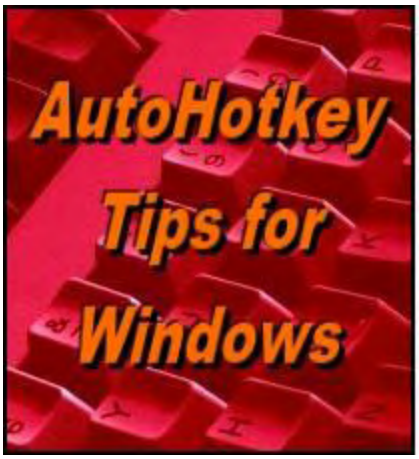
This simple example is the seed for many other much more complicated pencil movements—easily replicated with AutoHotkey, but almost impossible to emulate by hand. The fact that a line can be drawn is no big deal, but, if you have the right formula, you can figure out how to make Paint draw almost any shape with an AutoHotkey script.

* * *

Want a taste of what AutoHotkey can do? Check out these Free AutoHotkey Scripts and Apps

* * *

Free! *AutoHotkey Tricks You Ought To Do With Windows!* This e-book includes both those tips and the reference material (Table of Contents and indexes) from the other books.



Make Your Own Drawing Tools for Windows Paint with AutoHotkey

"Build Functions With AutoHotkey to Create Controls for Windows Paint (Or Any Other Program)" by Jack Dunning

Learn how to use the same line drawing code over and over again by putting it in an AutoHotkey function. Then draw a line anywhere in Windows Paint of any length at any angle using only one line of code.

Last week I discussed a short AutoHotkey script which uses the Pencil tool in Windows Paint to draw a straight line by moving the tool a pixel at a time inside a *Loop*. This is not the best way to draw a line in Windows Paint. The Line tool is easier to use and much quicker. However, the goal here is to learn techniques for controlling programs which can be used in other applications, not always find the one best way to do something in Windows Paint.

New to AutoHotkey? See our Introduction to AutoHotkey!

This week that same drawing technique will be turned into its own tool by putting the AutoHotkey code inside a function. This makes it possible to draw various straight lines without rewriting the code for each instance. Once the function is written, a straight line of any length and orientation can be drawn at any location on the grid with only one line of code. Writing AutoHotkey user-defined functions is a great way to build an app toolkit.

Tip: While testing the line drawing script, I turned the test code into a hotkey by enclosing it between ALT+D (!d*) and the* Return *command. This way the script stays loaded and ready to run whenever ALT+D is used. After I make a change to the script, I save the file and right-click on the System Tray icon selecting Reload This Script. This makes it easy to alter and test the script.*

Creating the DrawLine Function

User-defined functions are one of the best ways to build modular scripts while limiting the amount of code needed. If you find yourself writing the same code over and over again, then it may be time to consider writing it only once and putting it in a function. It is usually as simple as creating variables for the code, putting them in the function as parameters, then calling the function while including values for the parameter.

The short line drawing script from last week uses program shortcuts and mouse cursor control techniques to move the Windows Paint Pencil tool in a straight line (as shown in Figure 1). But it only draws one line in the same location. Let's turn it into a function which can be used an unlimited number of times with many variations.

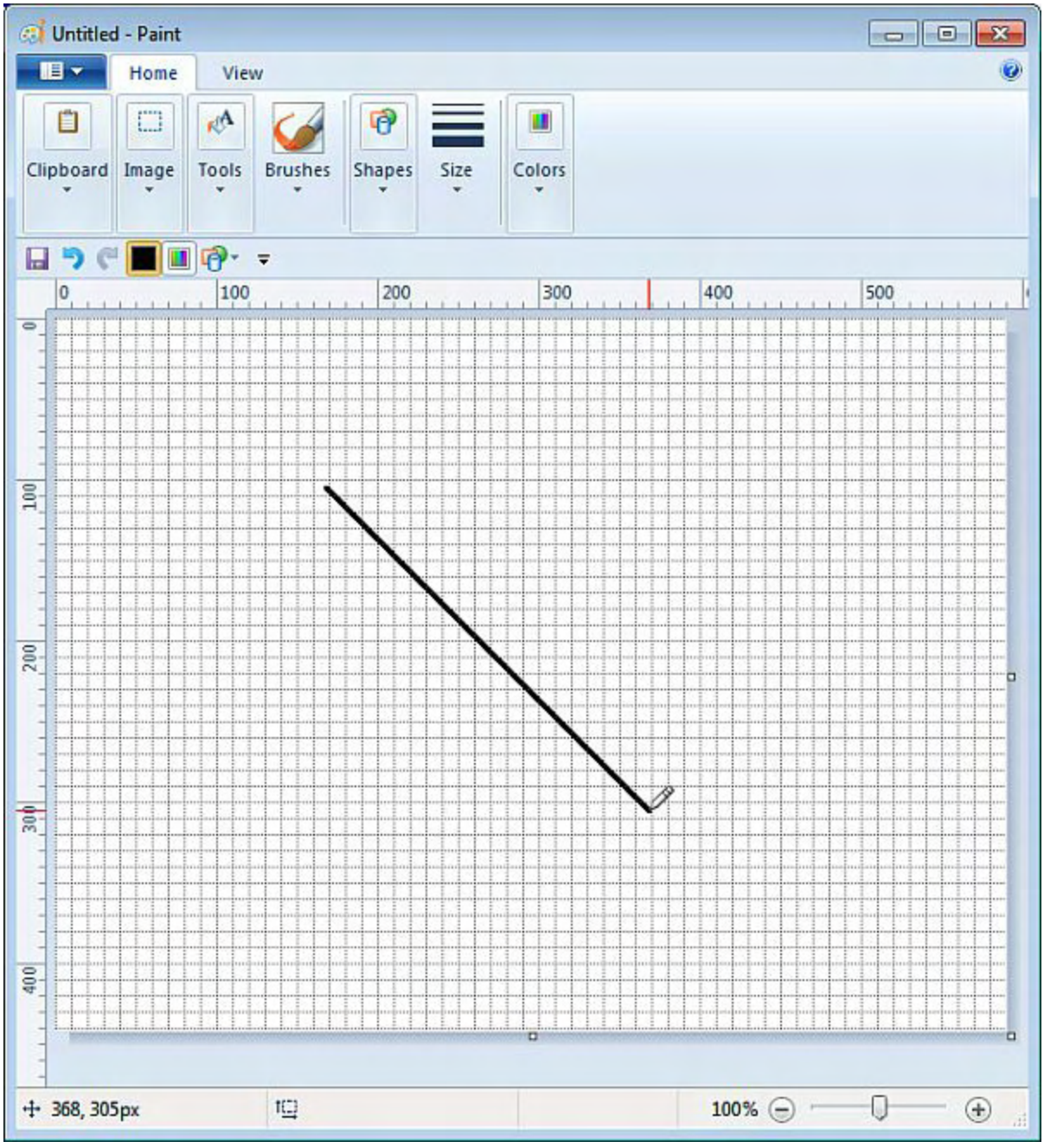


Figure 1. A straight line is drawn in Windows Paint with an AutoHotkey script.

Enclosing the code from last week with ALT+D (*!d*) and the *Return* command creates a hotkey for easy testing:

```
!d::
  SendInput {Alt}hp
  Sleep 100
  SendInput {Home}{Enter}
  Sleep 200
  Click Down 200,300
  Loop, 200
  {
    MouseMove 1, 1, 50, R
  }
  Click Up
Return
```

(This AutoHotkey script is discussed in last week's column.)

An AutoHotkey function is a subroutine which can be run at any time by simply calling the function with the appropriate parameters. Functions are distinguished by the set of parentheses () which always directly follows (no space) the name of the function, i.e. *DrawLine()*. The set of parentheses may or may not contain parameters (values) which are passed to the function to use. If parameters are defined in the function, then the calling function must account for them—even if they have a zero or null value.

In this case, the function *DrawLine()* is defined with five parameters:

```
DrawLine(XPos,YPos,MoveX,MoveY,LineLength)
```

XPos is the starting position x coordinate, *YPos* is the starting position y coordinate, *MoveX* is the end point x coordinate, *MoveY* is the end point y coordinate, and *LineLength* is how many pixels or *Loop* iterations to move the cursor. The entire section of code in the function is place between two curly brackets *{}* with the parameter variables placed at the appropriate spots in the script:

```
DrawLine(XPos,YPos,MoveX,MoveY,LineLength)
{
  SendInput {Alt}hp
  Sleep 200
  SendInput {Home}{Enter}
  Sleep 200
  Click Down %XPos%,%YPos%
  Loop, %LineLength%
  {
    MouseMove %MoveX%, %MoveY%, , R
  }
  Click Up
}
```

Note: Where the script previously included constant numeric values, it now uses the parameter

variables enclosed with the percent sign *%*. When AutoHotkey encounters variables enclosed with two percent signs, e.g. *%XPos%*, the value of the variable (which is included in the calling function) is substituted for the variable. This can be a little confusing in AutoHotkey since within most evaluated expressions the percents signs are not normally used for variables. As a rule, when adding variables to an AutoHotkey command, they are enclosed with two *%* symbols, but within expressions with operators, they are not.

A function can appear almost anywhere in an AutoHotkey script, although it is usually best to locate them toward the end of the file or in a separate file which is loaded with the #Include command. To use the function simply add it to the script as a command:

```
  DrawLine(200,300,1,1,200)   ;Diagonal right down
```

AutoHotkey sees *DrawLine()* with the proper number of parameters separated by commas between the parentheses, recognizes it as a valid previously loaded function, and immediately runs it. The values found within the parentheses are matched respectively to the variables in the functions and used to perform whatever the code directs.

The following script draws the same line as that drawn by the original script above:

```
!d::
  DrawLine(250,350,1,1,200)   ;Diagonal right down
Return

DrawLine(XPos,YPos,MoveX,MoveY,LineLength)
{
  SendInput {Alt}hp
  Sleep 200
  SendInput {Home}{Enter}
  Sleep 200
  Click Down %XPos%,%YPos%
  Loop, %LineLength%
  {
    MouseMove %MoveX%, %MoveY%, , R
  }
  Click Up
}
```

Pressing the ALT+D key combination while Windows Paint is the active window draws a line down and to the right. It starts at *XPos,YPos* (250,350) per the *Click Down* command. Then it uses *MouseMove* command inside a *Loop* to move the cursor *MoveX* pixels to the right and *MoveY* pixels down (1,1) *LineLength* (200) times. The mouse button is then released.

Now it is possible to draw any number of lines with the same function by merely adding a function call for each new line—just changing the parameters:

```
!d::
  DrawLine(250,350,1,1,200)   ;Diagonal right down
  DrawLine(250,350,1,0,200)   ;Horizontal right
```

```
  DrawLine(250,350,0,1,200)   ;Vertical down
  DrawLine(250,350,-1,0,200)  ;Horizontal left
  DrawLine(50,350,1,1,200)    ;Diagonal right down
  DrawLine(50,550,1,0,400)    ;Horizontal right long
  DrawLine(450,550,0,-1,200)  ;Vertical up
  DrawLine(50,550,0,-1,200)   ;Vertical up
Return

DrawLine(XPos,YPos,MoveX,MoveY,LineLength)
{
  SendInput {Alt}hp
  Sleep 200
  SendInput {Home}{Enter}
  Sleep 200
  Click Down %XPos%,%YPos%
  Loop, %LineLength%
  {
    MouseMove %MoveX%, %MoveY%, , R
  }
  Click Up
}
```

This short script draws the image shown in Figure 2, one line segment for each function call.

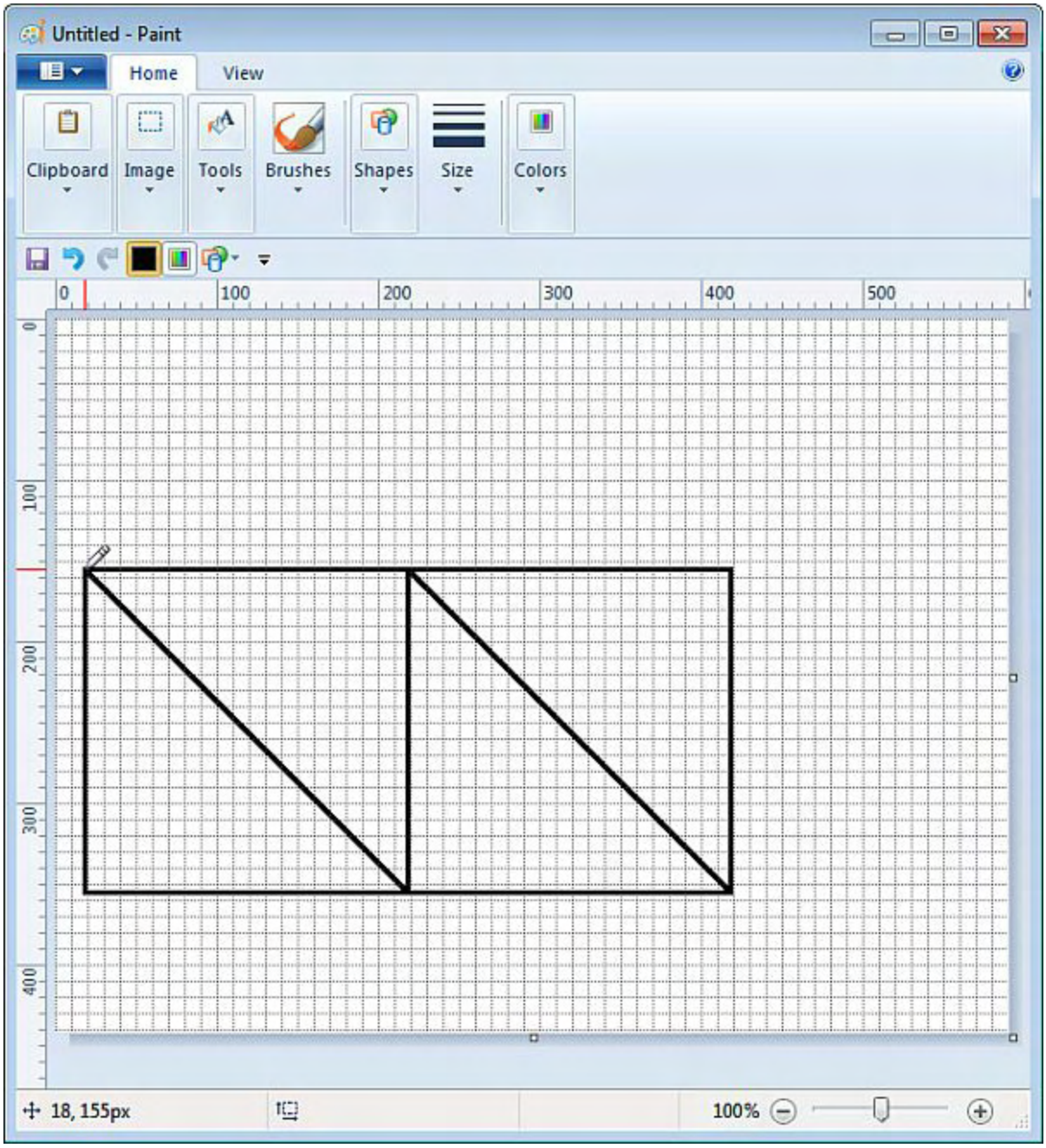
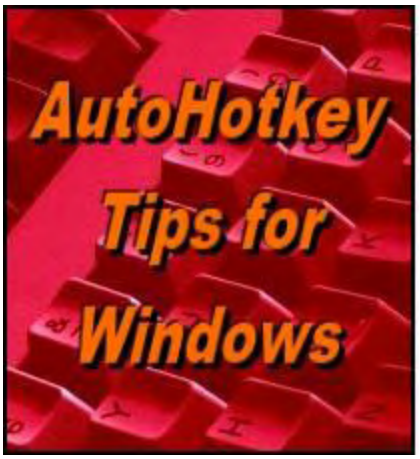


Figure 2. Multiples lines can be drawn with the same code by placing it within a function.

Using functions is one way that AutoHotkey script writers start to build libraries of tools. When properly designed they are easy to use and make apps more powerful with less code. If you find yourself writing the same code multiple times or continuously cutting and pasting code to other sections of the same script, it's time to consider whether you should be using a user-defined function.

* * *

Check out many of the way AutoHotkey can be used! See these [Free AutoHotkey Script.](#)



Automatically Find Buttons by Color in Windows Programs

"Yet, One More Reason to Use AutoHotkey Free Software!" by Jack Dunning

This little know AutoHotkey command will scan a section of a window or image for a specific color. Use it to select colors in Windows Paint.

When controlling the Windows Paint program, the built-in shortcut keys are great for selecting tools. In AutoHotkey it's as simple as sending the appropriate key combinations with the *SendInput* command. However, changing colors is not as easy. While the color editing window can be opened by sending shortcut keys to Paint (*SendInput {Alt}hec*), there is no way to select particular colors with these key sequences. It can only be done with a *Click*.

New to AutoHotkey? See our Introduction to AutoHotkey!

We could predetermine the coordinates of the color needed and use the Click command to select a color, but what if that color has been moved to a different box—which is easy in Windows Paint? Fortunately there is a very cool command in AutoHotkey called PixelSearch which will seek out and deliver the location of any desired color in any window. While I've known of the existence of the *PixelSearch* command, I've never used it before.

The *PixelSearch* command scours a designated area of a window looking for a particular color. If it finds the target, it retrieves the coordinates of that pixel. In Paint those coordinates can be used for a *Click* location to select the color. There are other specialized uses for this AutoHotkey color finding command. From forum comments I've seen that gamers often use the *PixelSearch* command to find the location of particular types of objects, e.g. magical, weapons, on the screen.

Finding a Color in a Window with AutoHotkey

Before searching for a particular color pixel in a window, the *PixelSearch* command requires the color name in hexadecimal code. There are a couple of ways to get this information.

You could use the PixelGetColor command in a short script which would return the color, but for our purposes it's easier to open Windows Spy (right-click on any running AutoHotkey (.AHK) script icon in the System Tray and select "Window Spy"). The always-on-top informational window will open (see Figure 1). (Window Spy is an app which comes with the AutoHotkey installation. It can be used to identify window and control names, cursor locations, colors and other data useful in AutoHotkey scripts.) When hovering the mouse over a window, Window Spy identifies the color of the pixel under the cursor.

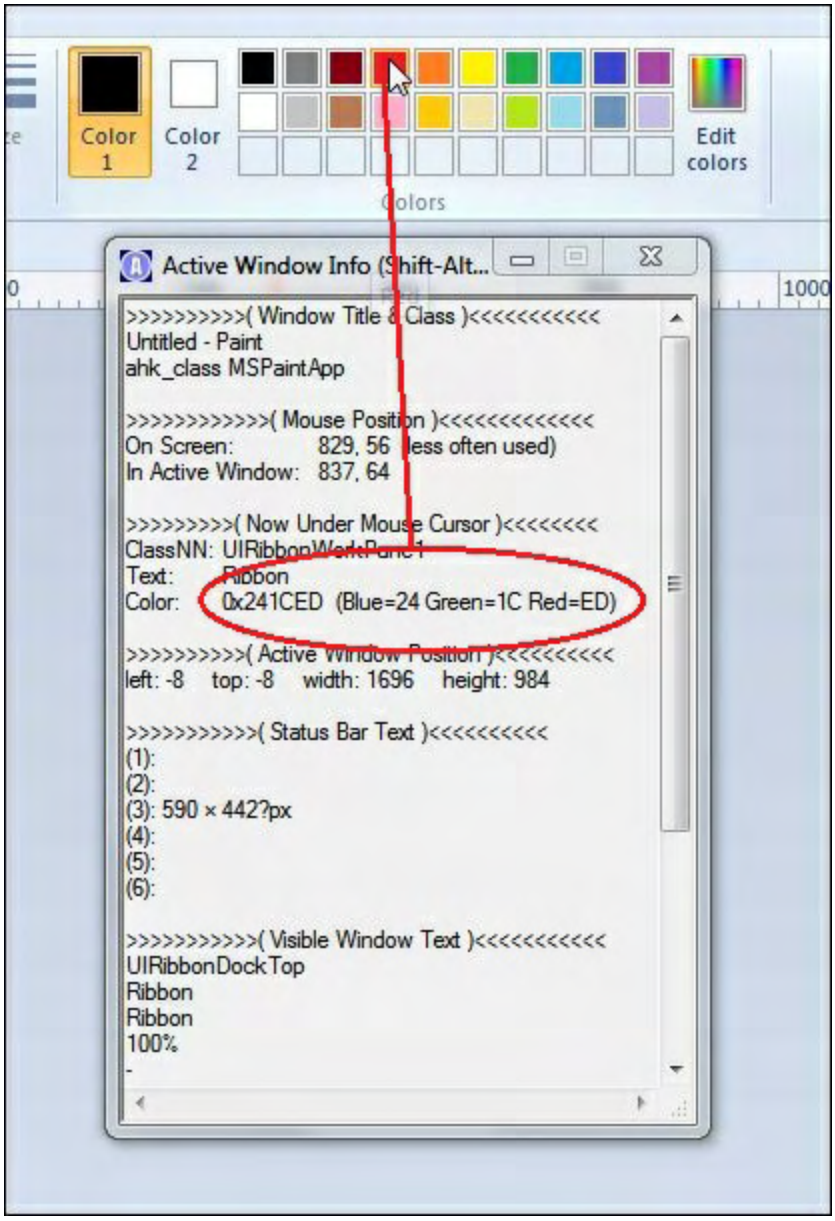


Figure 1. When hovering the mouse over a window, Window Spy identifies the color of the pixel under the cursor.

The Search Area

Window Spy can also be used to locate the coordinates for the search section of the window.

This narrows the search and saves some processor time. The coordinates for the color selection palette in Windows Paint color were identified as shown in Figure 2. The search area is restricted to the color selection palette to ensure that any sections of the window which may also contain the same search color are not included.

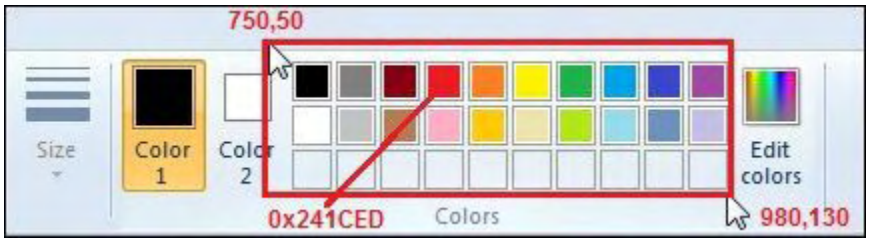
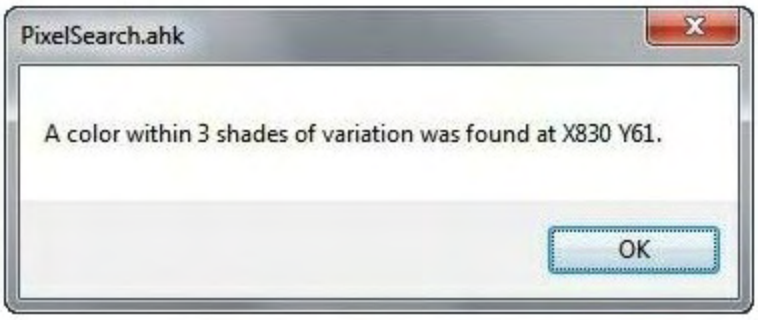


Figure 2. The search area as identified by Window Spy for the PixelSearch command is bound by 750, 50 and 980,130. The color code is 0x241CED.

The *PixelSearch* command is written as follows:

```
PixelSearch, Px, Py, 750, 50, 980, 130, 0x241CED, 3, Fast
```

The coordinates for a pixel found with the color *0x241CED* (red) is saved in the variables *Px* and *Py*. The command uses the same section coordinates and color as shown in Figure 2 above. The number 3 parameter tells AutoHotkey to accept any color which is within three shades of the original. Adding the *Fast* parameter speeds up the search. The *Fast* mode searches line by line from top to bottom stopping on the first successful hit. Figure 3 (top) shows a *MsgBox* of the coordinates returned by the *PixelSearch* command. Figure 3 (bottom) shows where the *Click* command would be issued to select the color red in Windows Paint.



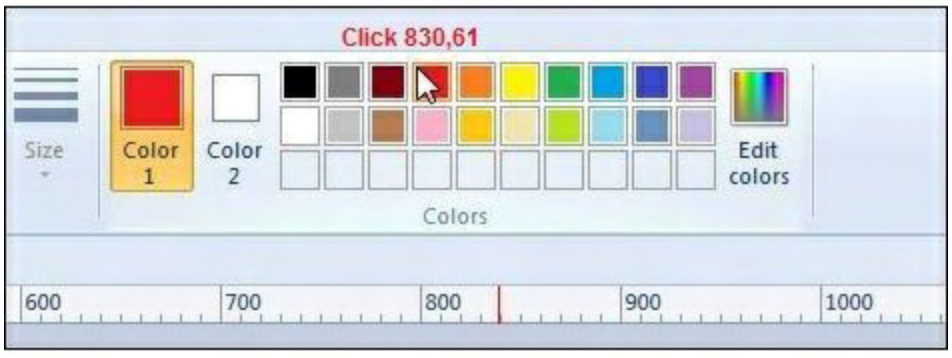


Figure 3. The coordinates of the first matching pixel (top). Where the Click command would select the color red (bottom).

The top portion of Figure 3 is displayed by adding the following snippet of code:

```
If ErrorLevel
    MsgBox, That color was not found in the specified region.
Else
  {
    MsgBox, A color within 3 shades of variation was found at X%Px% Y%Py%.
    Click, %Px%,%Py%
  }
```

If the color is not matched (*If ErrorLevel*), then a message box is displayed stating that the color was not found. Otherwise, the coordinates and the number of shades of variation are shown as in Figure 3 (top) and the Click command is issued at that same location as shown in Figure 3 (bottom).

Warning: Using this technique could be problematic when looking for black or white. Even within the search area there may be pixels (black lines) which will yield a positive result without actually finding the location of the proper selection button.

Make a Box and Fill It with Red

A few weeks ago, I offered a short AutoHotkey script which made a box then filled it with the current color. Using that same script, I've used the *PixelSearch* technique in a modification of that script to find the red selection button, then fill the box with red:

```
!d::
  SendInput {Alt}hsh
  Sleep 100
  SendInput {Right 3}{Enter}
  Sleep 200
  Click Down 200,300
  MouseMove 200, 200, 50, R
  Click Up
  Click 40,200
  PixelSearch, Px, Py, 750, 50, 980, 130, 0x241CED, 3, Fast
  Click, %Px%,%Py%
  SendInput {Alt}hk
  Sleep 200
  Click 250,350
Return
```

The script encloses the lines of code with *!d::* and *Return* creating a hotkey combination (ALT+D) for easier testing and reloading.

Three lines of code have been added to the original script. First, *Click 40,200* is added to

Click outside the drawn box making sure it is deselected before changing the color. Otherwise the box would remain as a selected object and turn red when the new color is selected.

Second the *PixelSearch* command is used to find the red color selection button (*PixelSearch, Px, Py, 750, 50, 980, 130, 0x241CED, 3, Fast*).

Third, the color red is selected (*Click, %Px%,%Py%*).

As shown in the linked previous column, the script concludes by selecting the Fill tool and clicking inside the box to fill is with the color red (see Figure 4).

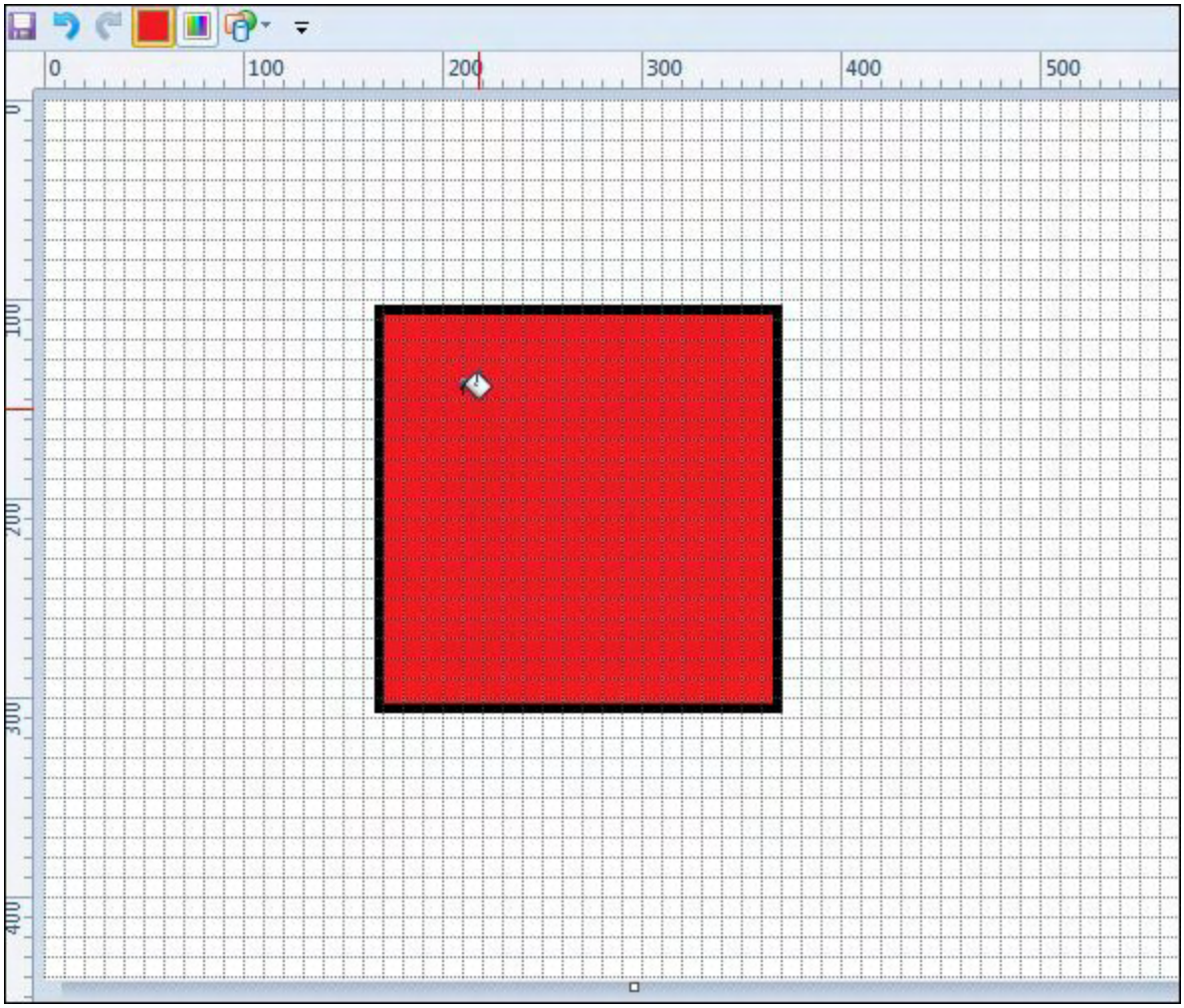
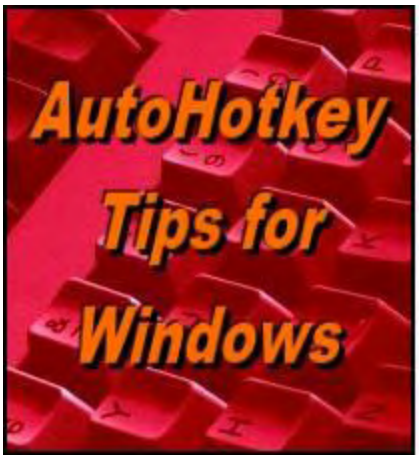


Figure 4. After the script selects the color red, the black box is filled.

This is the first time I've had an occasion to use *PixelSearch*. It's pretty impressive, but I'm not sure how many other applications there are for it. I can see how it might be used for a child's game for learning colors or searching game screens. I suppose that if you're in the right field, the AutoHotkey *Pixel* commands could be important in controlling your software.

* * *



Controlling More Windows Paint Tools with AutoHotkey

"Yet, One More Reason to Use AutoHotkey Free Software!" by Jack Dunning

Another step is taken toward making AutoHotkey tools for controlling Windows programs.

A few weeks ago I wrote an AutoHotkey function which draws a straight line in Windows Paint using the Pencil tool. It was a little slow but it did the job (as long as the mouse isn't moved while the script is in action). Just for fun, I tried out the same function with different drawing tools in Windows Paint with some interesting results. We'll take a look at some tools, modify the function to speed up operations, and show how to block any interference from an inadvertent manual movement of the mouse.

Other Windows Paint Tools

There are quite a few drawing tools which make a variety of shapes. The problem with many of them is their operation is enigmatic. Click and hold to start, then drag. Then click again and drag for the next part. There may be three, four, or more clicks necessary before the shape is completed. Even then it is very difficult to manually replicate a particular shape. The advantage of using a script is exact shapes can be made (once the tool is understood) and repeated as often as necessary. (This is similar to the type of programming needed in any machine control.)

Making Squiggles

I started with the Curve tool which is the second selection in the shapes palette. Using the same script code which introduced the drawing function, the script is modified to use the Curve tool. The resulting image is displayed in Figure 1. It was fascinating to watch the AutoHotkey script do its work as the Curve tool moved and manipulated the image it was drawing. The slow moving tool demonstrated how it responded to the data in the script providing more insight into how it may be used in drawing creations.

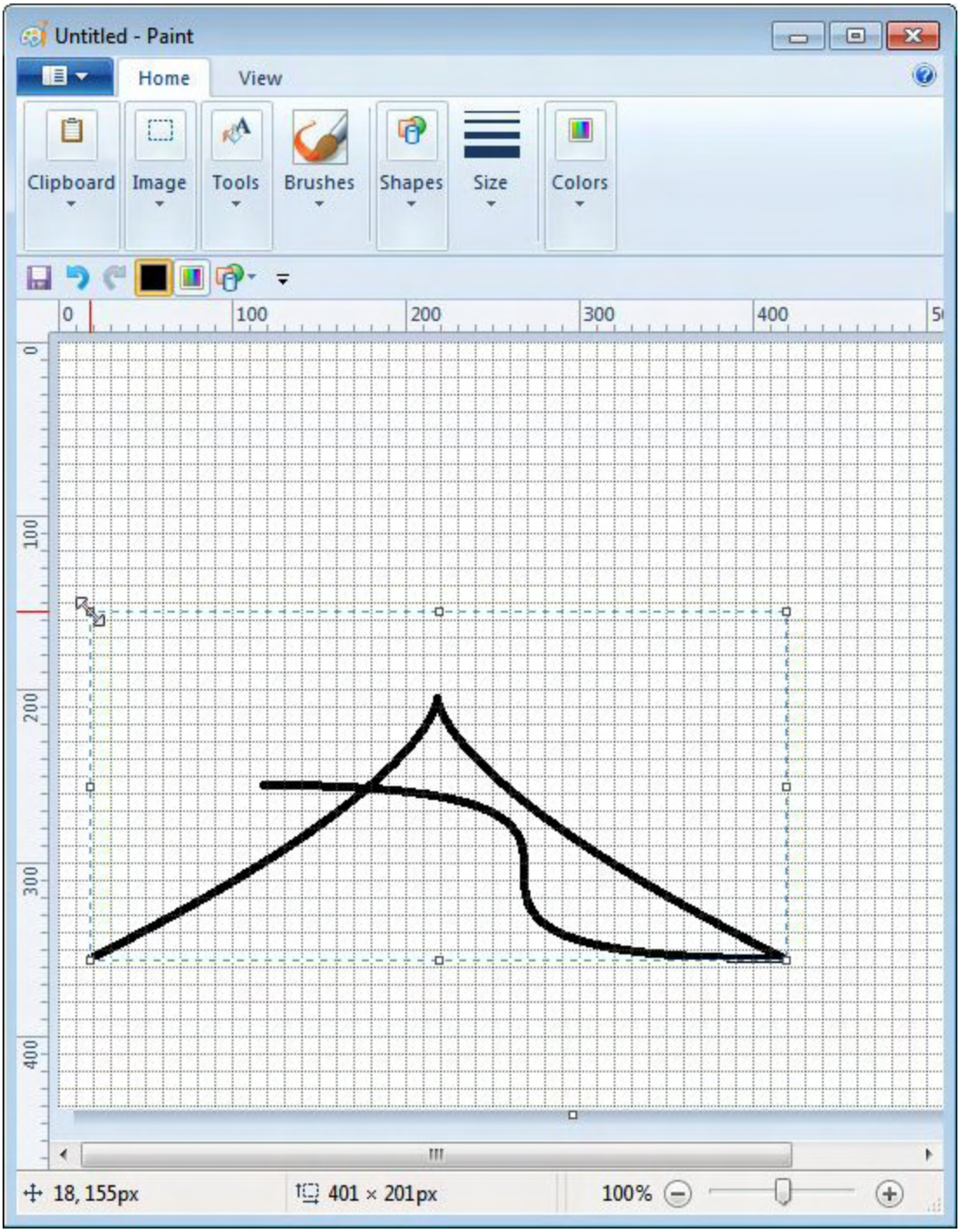


Figure 1. The curve tool is capable of making squiggles and other irregular shapes.

The original script is rewritten to use the Curve tool:

```
!d::
  DrawSquiggle(250,350,1,1,200)
  DrawSquiggle(250,350,1,0,200)
  DrawSquiggle(250,350,0,1,200)
  DrawSquiggle(250,350,-1,0,200)
  DrawSquiggle(50,350,1,1,100)
  DrawSquiggle(50,550,1,0,400)
  DrawSquiggle(450,550,0,-1,200)
  DrawSquiggle(50,550,0,-1,200)
Return

DrawSquiggle(XPos,YPos,MoveX,MoveY,LineLength)
{
  SendInput {Alt}hsh
  Sleep 200
  SendInput {Home}{Right}{Enter}
  Sleep 200
  Click Down %XPos%,%YPos%
  Loop, %LineLength%
  {
    MouseMove %MoveX%, %MoveY%, , R
  }
  Click Up
}
```

Since the Curve tool is just to the right of the Line tool in the shapes palette, it was only necessary to move 1 to the right (*SendInput {Home}{Right}{Enter}*). The function name was changed to *DrawSquiggle()*.

Speeding Up the Function

As with the original function, *DrawSquiggle()* only moves one pixel at a time making the process pretty slow. But unlike the Pencil tool, the Curve tool will always respond in the same manner without using the *Loop*. Therefore the function can be rewritten to eliminate the Loop and use the endpoints (*MoveX,MoveY*) directly:

```
!d::
  DrawSquiggle(250,350,200,200,200)   ;Diagonal right down
  DrawSquiggle(250,350,200,0,200)   ;Horizontal right
  DrawSquiggle(250,350,0,200,200)   ;Vertical down
  DrawSquiggle(250,350,-200,0,200)  ;Horizontal left
  DrawSquiggle(50,350,100,100,100)    ;Diagonal right down
  DrawSquiggle(50,550,400,0,400)    ;Horizontal right long
  DrawSquiggle(450,550,0,-200,200)  ;Vertical up
  DrawSquiggle(50,550,0,-200,200)   ;Vertical up
Return

DrawSquiggle(XPos,YPos,MoveX,MoveY,LineLength)
{
```

```
  SendInput {Alt}hsh
  Sleep 200
  SendInput {Home}{Right}{Enter}
  Sleep 200
  Click Down %XPos%,%YPos%
  MouseMove %MoveX%, %MoveY%, , R
  Click Up
}
```

This script runs much faster since with the *Loop* removed it does not need to increment through each pixel. Rather than *MoveX* and *MoveY* containing single pixel increments (or 0), they now are changed to represent the endpoints. As the *Loop* is removed from the function, the variable *LineLength* is no longer used. It could be removed as a function parameter, but it is harmless to leave it in as long as the number of parameters passed to the function match with the number required by the function. It is possible that in another modification of the function *LineLength* will be needed again.

Temporarily Deactivating the Mouse

If you've run any of these AutoHotkey snippets, you may have noticed that actual mouse movement can cause a problem. If the mouse is accidentally moved manually during the script routine, it can cause the shape to be distorted. The best way to prevent this is to deactivate the mouse while the script is doing its work. This is done with the BlockInput command.

By adding the BlockInput command at the appropriate places in the *DrawSquiggle()* function, the mouse is deactivated only in the script that is actively drawing:

```
DrawSquiggle(XPos,YPos,MoveX,MoveY,LineLength)
{
  BlockInput,MouseMove
  SendInput {Alt}hsh
  Sleep 200
  SendInput {Home}{Right}{Enter}
  Sleep 200
  Click Down %XPos%,%YPos%
  MouseMove %MoveX%, %MoveY%, , R
  Click Up
  BlockInput,MouseMoveOff
}
```

In the beginning of the function, the mouse is turned off with *BlockInput,MouseMove*. At the end of the function the mouse is turned back on with *BlockInput,MouseMoveOff*. The problem of accidental manual mouse movements messing up the image is eliminated.

A Couple More Tools

Out of curiosity I tried the same function (renamed) with a couple of the other tools. I only needed to change the input data a little and the line of code in the function which selects the tool. The first test was run with the Polygon tool (see Figure 2).

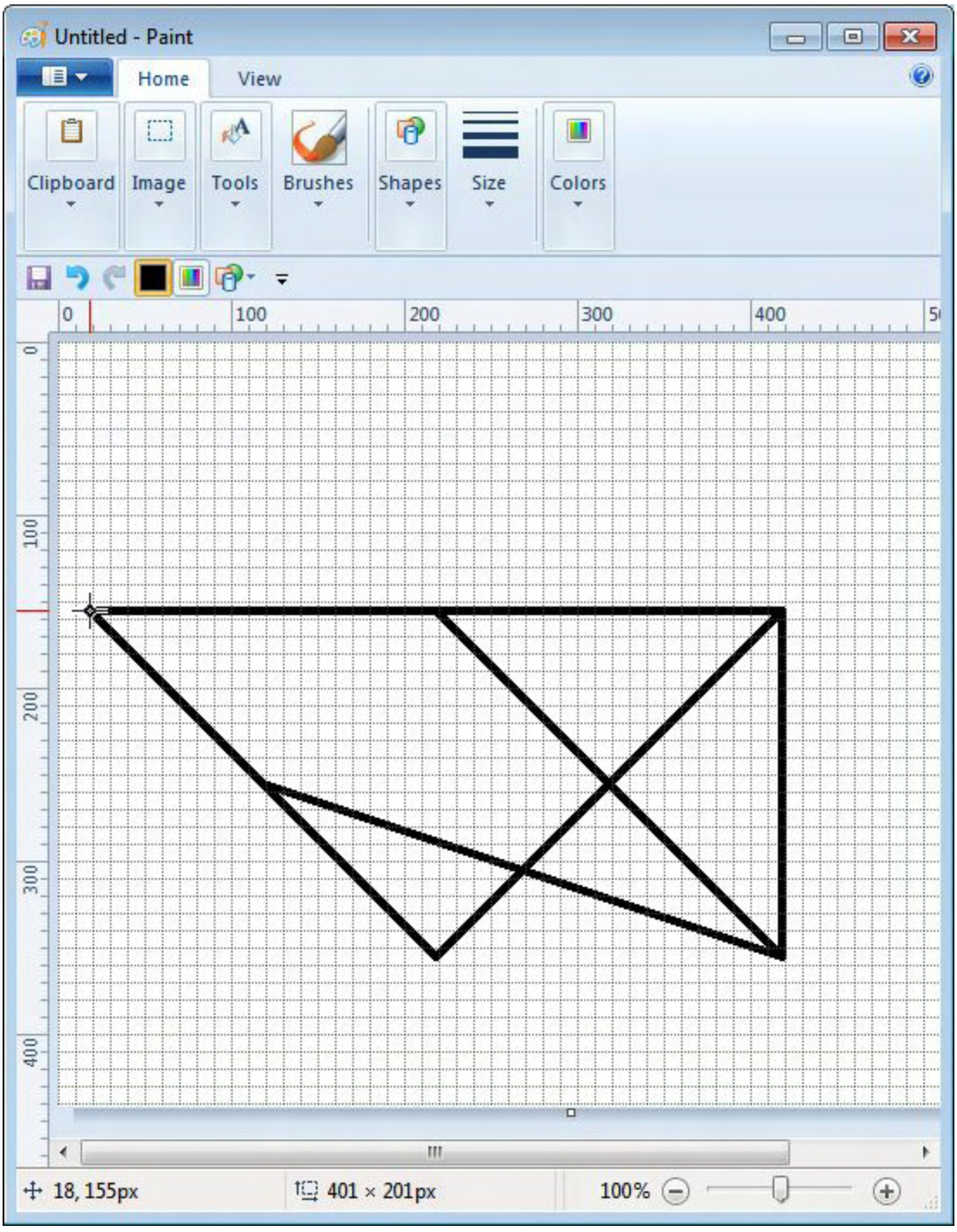


Figure 2. An AutoHotkey script which uses the Polygon tool in Windows Paint.

The input data used with the original script (including the *Loop* in the function) is as follows:

```
!d::
  DrawPolygon(250,350,1,1,200)    ;Diagonal right down
```

```
  DrawPolygon(250,350,1,0,200)   ;Horizontal right
  DrawPolygon(250,350,0,1,200)   ;Vertical down
  DrawPolygon(250,350,-1,0,200)  ;Horizontal left
  DrawPolygon(50,350,1,1,100)    ;Diagonal right down
  DrawPolygon(50,550,1,0,400)    ;Horizontal right long
  DrawPolygon(450,550,0,-1,200)  ;Vertical up
  DrawPolygon(50,550,0,-1,200)   ;Vertical up
Return
```

The tool selection line in the function was changed to select the Polygon (*SendInput {Home} {Right 5}{Enter}*). Note that the name of the function was change to *DrawPolygon()*. This renaming is important for eliminating conflicts if you plan to include all of the various drawing functions in one AutoHotkey script. Ultimately, I plan to make the various routines selectable with a pop-up window—pick the shape, enter size and location, and submit.

The other tool tested was the four-point star where I called the function *DrawNova()* (see Figure 3). Using the same input data as above the selection line was changed to *SendInput {Home}{Right 1}{Down 2}{Enter}* since the tool appears in the third row *{Down 2}* of the shapes palette.

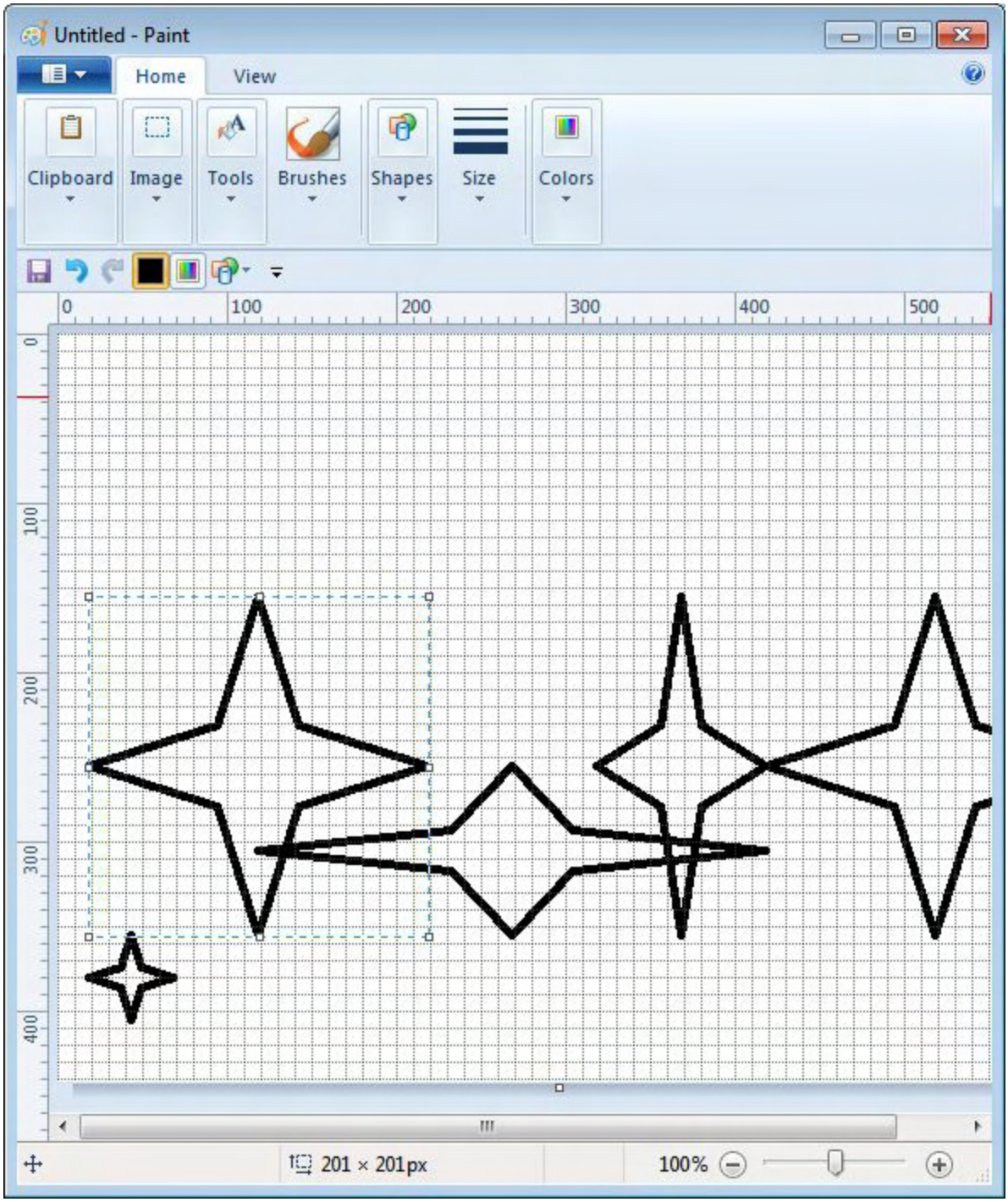


Figure 3. Four-point stars drawn with an AutoHotkey script.

The more I work with controlling programs with AutoHotkey, the more I see the possibilities. One problem with Windows Paint is that it always opens with some of the same saved settings from the last session (font, font size, bold, italics, etc.), but resets other settings to defaults (linewidth, colors, etc.) when the program is reloaded. By using a short AutoHotkey script, the settings can be automatically reset for a particular use after opening Paint. This would save the manual selection process required to reset numerous options with the mouse. These techniques would apply equally to almost any other Windows program which requires a different setup

for different projects.

* * *

The new version of the *AutoHotkey Applications* e-book with the update reference links is now available. Remember, if you purchased from ComputorEdge E-Books and ran out of download links, please contact us to get links for the updated version. If you purchased from Amazon, you should be able to update through your Amazon account.

See out Free AutoHotkey Scripts and Apps for learning AutoHotkey.

Jack is the publisher of ComputorEdge Magazine. He's been with the magazine since first issue on May 16, 1983. Back then, it was called The Byte Buyer. His Web site is www.computoredge.com. He can be reached at computoredge@gmail.com. Jack is now in the process of updating and compiling his hundreds of articles and columns into e-books. Currently available:

Recently released is Jack's FREE AutoHotkey book, AutoHotkey Tricks You Ought to Do with Window, available exclusively at ComputorEdge E-Books in the EPUB for e-readers and tablets, MOBI for Kindle, and PDF for printing formats.

ComputorEdge E-books is offering his AutoHotkey Applications, an idea-generating intermediate level e-book about using the AutoHotkey Graphical User Interface (GUI) command to write practical pop-up apps for your Windows computer. (It's not as hard as it sounds.)

Hidden Windows Tools for Protecting, Problem Solving and Troubleshooting Windows 8, Windows 7, Windows Vista, and Windows XP Computers.

Jack's *A Beginner's Guide to AutoHotkey, Absolutely the Best Free Windows Utility Software Ever!: Create Power Tools for Windows XP, Windows Vista, Windows 7 and Windows 8* and *Digging Deeper Into AutoHotkey*.

Our second compilation of stupid *ComputorEdge* cartoons from 2011 and 2012 is now available at Amazon! *That Does Not Compute, Too! ComputorEdge Cartoons, Volume II: "Do You Like Windows 8 or Would You Prefer an Apple?"*

Special Free Offer at ComputorEdge E-Books! *Jack's Favorite Free Windows Programs: What They Are, What They Do, and How to Get Started!*.

Misunderstanding Windows 8: An Introduction, Orientation, and How-to for Windows 8 (Seventh Edition)!

Windows 7 Secrets Four-in-One E-Book Bundle,
Getting Started with Windows 7: An Introduction, Orientation, and How-to for Using Windows 7,
Sticking with Windows XP—or Not? Why You Should or Why You Should Not Upgrade to Windows 7,
and *That Does Not Compute!*, brilliantly drawn cartoons by Jim Whiting for really stupid gags by Jack about computers and the people who use them.