FREE eBook

LEARNING AutoHotkey

#autohotkey

Table of Contents

About

Chapter 1: Getting started with AutoHotkey

Remarks

AutoHotkey is a free, open-source custom scripting language for Microsoft Windows, initially aimed at providing easy keyboard shortcuts or hotkeys, fast macro-creation and software automation that allows users of most levels of computer skill to automate repetitive tasks in any Windows application. User interfaces can easily be extended or modified by AutoHotkey (for example, overriding the default Windows control key commands with their Emacs equivalents). The Autohotkey installation includes its own extensive help file with an always updated web based version.

You can write mouse or keyboard macros, remap keys, create hotkeys, expand abbreviations, change clipboard contents, and make executables to run hotkey scripts on computers without AutoHotkey installed.

Versions

AutoHotkey 1.0.* - also retroactively known as AutoHotkey Basic, Classic, Vanilla, etc.

(Development was discontinued in 2011; Latest stable: 2009)

Version	Release Date
v1.0.48.05	2009-09-26
v1.0.97.02	2011-04-14

AutoHotkey 1.1.* - previously known as AutoHotkey_L.

(Stable and receives updates regularly)

Version	Release Date
v1.1.24.00	2016-05-22
v1.1.24.01	2016-08-02

AutoHotkey 2.0-a*

(Still in alpha stage)

Version	Release Date
v2.0-a069	2015-10-24
v2.0-a070	2015-11-09
v2.0-a071	2015-12-25
v2.0-a072	2015-12-25
v2.0-a073	2016-02-05
v2.0-a074	2016-03-11
v2.0-a075	2016-06-03

Examples

Installation or Setup

From Autohotkey Site Documentation

1. Go to the AutoHotkey Homepage.
2. Click Download, once downloaded run the executable.
3. During installation of AutoHotkey, you will be asked to choose from UNICODE or ANSI. In short, you would probably want to choose UNICODE. It has support for non-English letters and numbers (characters).
4. Keep going until you see an Install button.
5. Once done, great!

Use as portable software

1. Go to the AutoHotkey's download page.
2. Find the Portable section, choose from UNICODE 32, 64 or ANSI and downloaded.
3. When choosing the destination folder, pick any correct storedevice external or not.
4. Now you can opt to associate .ahk files with Autohotkey.exe
5. Create a plain text file and give it the .ahk extension
6. Then Right-click on the .ahk file in explorer and click Properties.
7. In the file Properties, click the Change button next to the "Opens with" option.
    - After clicking Change, you'll be given a list of programs to open the file, select the program you want to use and then click OK or Apply.
    - If the program you want to select is not listed, click the Browse button and find the Autohotkey executable (.exe) file and click OK to select that program.
8. Now .ahk files will run as if autohotkey was installed, great!

If you have chocolatey installed, run the following command as an admin user

    choco install autohotkey

Alternatively, it can be built from the source code. See here for details: https://github.com/Lexikos/AutoHotkey_L/

Hello World

Show a "Hello World!" in message box.

```
MsgBox, Hello World!
```

Show a "Hello World!" in tooltip.

```
#Persistent
Tooltip, Hello World!
```

Show a "Hello World!" message in the traybar edit.

```
#Persistent
TrayTip,, Hello World!
```

Prints "Hello, World" to Standard Output (stdout).

```
FileAppend, % "Hello, World", *
```

Show "Hello World" in a GUI

```
Gui, Add, Text,, Hello World!
Gui, Show, w200 h200
return

GuiClose:
ExitApp
```

Achieve an effect similar to SplashTextOn

```
Gui, +AlwaysOnTop +Disabled -SysMenu +Owner  ; +Owner avoids a taskbar button.
Gui, Add, Text,, Some text to display.
Gui, Show, NoActivate, Title of Window  ; NoActivate avoids deactivating the currently active
window.
```

How to create a Script

Once you have AutoHotkey installed, you will probably want it to do stuff. AutoHotkey is not magic, we all wish it was, but it is not. So we will need to tell it what to do. This process is called "Scripting".

1. Right-Click on your desktop.
2. Find "New" in the menu.
3. Click "AutoHotkey Script" inside the "New" menu.
4. Give the script a new name. Note: It must end with a .ahk extension. Ex. MyScript.ahk
5. Find the newly created file on your desktop and Right-Click it.
6. Click "Edit Script".
7. A window should have popped up, probably Notepad. If so, SUCCESS!

So now that you have created a script, we need to add stuff into the file. For a list of all built-in commands, function and variables, see section 5. Here is a very basic script containing a Hotkey which types text using the Send command when the hotkey is pressed.

```
^j::
   Send, My First Script
Return
```

We will get more in-depth later on. Until then, here's an explanation of the above code.

- The first line. `^j::` is the Hotkey. `^` means `CTRL`, `j` is the letter j. Anything to the left of `::` are the keys you need to press.
- The second line. `Send, My First Script` is how you `SEND` keystrokes. `SEND` is the command, anything after the comma (,) will be typed.
- The third line. `Return`. Return will become your best friend. It literally STOPS code from going any further, to the lines below. This will prevent many issues when you start having a lot of stuff in your scripts.

8. Save the File.
9. Double-Click the file/icon in the desktop to run it. Open notepad or (anything you can type in) and press Ctrl and J.
10. Hip Hip Hooray! Your first script is done. Go get some reward snacks then return to reading the rest of this tutorial.

Read Getting started with AutoHotkey online: https://riptutorial.com/autohotkey/topic/3532/getting-started-with-autohotkey

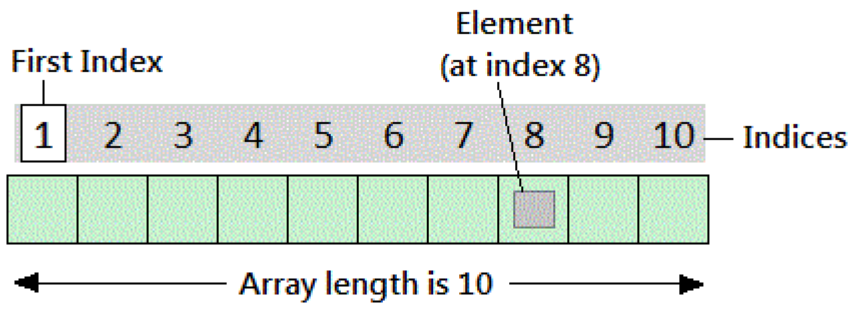
Chapter 2: Arrays

Examples

Creating and Initializing Simple Arrays

Intro

An *array* is a container object that holds a number of values. In the following image you can see an array with size 10, the first element indexed 1 and the last element 10.



Autohotkey offers a few ways of defining and creating arrays.

- Array := []
- Array := Array()

Creating and initializing arrays with N number of items

```
Array := [Item1, Item2, ..., ItemN]
Array := Array(Item1, Item2, ..., ItemN)
```

In Autohotkey, it is possible to have arrays with no items:

```
Array := [] ; works fine.
```

And elements can then later be assigned to it:

```
Array[0] := 1
```

Array size can be determined using a method called `length`:

```
msgbox % array.length()  ; shows 1 in this case.
```

If the array is not empty, **MinIndex** and **MaxIndex/Length** return the lowest and highest index currently in use in the array. Since the lowest index is nearly always 1, MaxIndex usually returns the number of items. However, if there are no integer keys, MaxIndex returns an empty string whereas Length returns 0.

Creating and initializing multi-dimensional arrays

You can create a multi-dimensional array as follows:

```
Array[1, 2] := 3
```

You can create and initialize at the same time time, and inner arrays do not need to be of same length.

```
Array := [[4,5,6],7,8]
```

Arrays like this are also called arrays of arrays.

Filling an array

```
; Assign an item:
Array[Index] := Value

; Insert one or more items at a given index:
Array.InsertAt(Index, Value, Value2, ...)

; Append one or more items:
Array.Push(Value, Value2, ...)
```

The value of an index for an array element can also be a negative integer (-1, 0, 1, 2, 3, 4, ...)

Removing Elements from an array

```
; Remove an item:
RemovedValue := Array.RemoveAt(Index)

; Remove the last item:
RemovedValue := Array.Pop()
```

Adding custom Methods by overriding Array() function

AutoHotkey is a prototype-based programming language, meaning you can override any built-in function/object at anytime. This example demonstrates overriding Array() function in order to add Methods within a custom Class Object.

```
; Overrides Array()
Array(Args*) {
    Args.Base := _Array
    Return Args
}

; Custom Class Object with Methods
Class _Array {

    ; Reverses the order of the array.
    Reverse() {
        Reversed := []
        Loop % This.MaxIndex()
            Reversed.Push(This.Pop())
        Return Reversed
    }

    ; Sums all Integers in Array
    Sum(Sum=0) {
        For Each, Value In This
            Sum += Value Is Integer ? Value : 0
        Return Sum
    }
}

; Arr == ["Hello, World!", 4, 3, 2, 1]
Arr := [1, 2, 3, 4, "Hello, World!"].Reverse()

; SumOfArray == 10
SumOfArray := Arr.Sum()
```

Read Arrays online: https://riptutorial.com/autohotkey/topic/4468/arrays

Chapter 3: Built-in Variables and Functions

Remarks

AutoHotkey comes with many built-in functions and variables which can be used anywhere inside a script.
For a full list including explanations, see:

- List of built-in variables
- List of built-in functions

Examples

Determining the User Idle Time

```
if(A_TimeIdlePhysical > 60000) { ; 60,000 milliseconds
    WinClose, ahk_class Chrome_WidgetWin_1
    MsgBox, Google Chrome was closed due to user inactivity.
}
```

This check could be done periodically, e.g. using `SetTimer`.

Auto-insert current weekday's name

This example inserts/sends the current day of the week's full name (e.g. *Sunday*) whenever `Ctrl` + `Alt` + `D` is pressed:

```
^!d::Send, %A_DDDD%
```

Extract string parts using RegEx

```
myDebt  := 9000
index   := RegExMatch("You owe me $42", "\$(\d+)", dollars)
if(index > 0) { ; indices are usually 1-based in AHK
    myDebt += dollars1
    MsgBox, Current debt: %myDebt%
}
```

Result:

> Current debt: 9042

Trim a string

```
myString := "  hello, Trim()! "
trimmed  := Trim(myString)
```

```
FileAppend, % trimmed "`n", TrimmedStrings.txt
```

Note that `Trim()` will not manipulate the original string, but return a new one which should be stored or output somewhere.

Read Built-in Variables and Functions online: https://riptutorial.com/autohotkey/topic/4469/built-in-variables-and-functions

Chapter 4: Hello World

Examples

Hello World examples

Show a "Hello World!" in message box.

```
MsgBox, Hello World!
```

Show "Hello World!" stored in variable MyString in message box.

```
MyString := "Hello World!"
MsgBox, %MyString%
```

Show a "Hello World!" in tooltip.

```
#Persistent
Tooltip, Hello World!
```

Show a "Hello World!" message in the traybar edit.

```
#Persistent
TrayTip,, Hello World!
```

Show "Hello World!" in a GUI window.

```
Gui, Add, Text,, Hello World!
Gui, Add, Edit,, Hello World!
Gui, Show
return

GuiClose() {
    ExitApp
}
```

Prints "Hello, World!" to Standard Output (stdout).

```
FileAppend, % "Hello, World!", *
```

Simulate typing "Hello, World!" when pressing enter

```
Enter::Send, Hello World{!}
```

Create a new file called HelloWorld.txt

```
FileAppend,, HelloWorld.txt
```

When typing the word "Hello" followed by a space or enter it will be replaced by "Hello World"

```
::Hello::Hello World
```

Read Hello World online: https://riptutorial.com/autohotkey/topic/3547/hello-world

Chapter 5: Hotkey Scripts

Syntax

- keybindings::
- ::abbreviation::
- Return

Parameters

Keybindings	Details
^	Ctrl key
!	Alt key
+	Shift key
#	Windows key
{enter}	send enter key
{tab}	send tab key
*	wildcard, any key can be pressed down
~	key's native function will not be blocked
<symbol	specifies left key (<+ is left shift)
>symbol	specifies right key

Examples

Hotkey

To make a hotkey that sends the key sequence 'Hello World' from pressing `Ctrl` + `J` onto the active window (can be demonstrated in notepad, e.g.)

```
^j::
    Send, Hello World
Return
```

Hotstring

To make a script to replace a phrase use the `::abbreviation::` hotstring syntax. It will replace `btw` with `by the way` whenever you enter `btw` and then the space key.

```
::btw::by the way
```

If you wanted to make a login script to make logging in faster you could make a script like this (the file is not encrypted so any information in your script will be visible to anyone with access to the file).

```
::lmi::user{tab}password{enter}
```

Multiple keypress

To run a script when multiple keys are pressed use the `&` between the keys.

```
Numpad0 & Numpad1::
    MsgBox You pressed 0 and 1
return
```

Context-sensitive Hotkeys and Hotstrings

In order to create a hotkey or hotstring that only triggers when certain windows are active or exist, you can put one or several of the following *directives* before the hotkey definition:

```
#IfWinActive [, WinTitle, WinText]
#IfWinExist [, WinTitle, WinText]
#IfWinNotActive [, WinTitle, WinText]
#IfWinNotExist [, WinTitle, WinText]
```

Example: You want `stackoverflow.com` to be sent whenever you type `so` (and a whitespace after that) in Google Chrome, but ignore the hotstring in any other window.

```
#IfWinActive, ahk_class Chrome_WidgetWin_1
::so::stackoverflow.com
```

By using `#If [, Expression ]`, you can make a hotkey trigger only when an arbitrary expression is true, for example:

```
#If A_Hour < 9
F1::
    MsgBox, It is too early to ask for help!
return
```

Remap keys

The following example remaps the key `Z` to `Y` and vice versa, e.g. if you want to work with the QWERTY layout on a QWERTZ keyboard.

```
z::y
y::z
```

Toggleable hotkeys

Following script enters predefined strings on hotkey presses if the scroll lock is active. This can be useful if you often paste a number of repeating strings. Included hotkey for script refresh (for example if you need to edit paste-able strings).

```
; refresh script hotkey
Numpad9::
    GetKeyState, state, ScrollLock, T
    if ( state = "D" )
        Reload
Return

Numpad1::
    GetKeyState, state, ScrollLock, T
    if ( state = "D" )
        Send,         Hello
Return

Numpad2::
    GetKeyState, state, ScrollLock, T
    if ( state = "D" )
        Send,         World
Return
;...
```

Read Hotkey Scripts online: https://riptutorial.com/autohotkey/topic/3853/hotkey-scripts

Chapter 6: Input Field

Introduction

To get a user's input and store it in a variable, you can use the InputBox command. The script will not continue executing commands until the user either presses 'OK' or 'Cancel'.

'OK' will close the window and save the user's input 'Cancel' will close the window, discarding the user's input

Parameters

InputBox, OutputVar [, Title, Prompt, HIDE, Width, Height, X, Y, Timeout, Default]	What each Option Means
OutputVar	The variable the user's input will be saved to
Title	The name of the input box
Prompt	Text inside of the input box
HIDE	Displays the user's input as asterisks to hide the input - type HIDE to enable
Width	The width of the input box
Height	The height of the input box
X	The amount of pixels from the left edge of the screen that the top-left corner of the input box will be
Y	The amount of pixels from the top edge of the screen that the top-left corner of the input box will be
Timeout	Automatically closes the input box and saves the user's input after this time in miliseconds
Default	The text that will appear in the input box's editable field when it is opened

Remarks

An input box is a GUI item, so it will be treated as a GUI item.

A list of errorlevels for this command:

Errorlevel	What it Means
0	The user pressed the 'OK' button
1	The user pressed the 'Cancel' button
2	The input box timed out

You can find the page for this command on the AutoHotkey documentation here: https://autohotkey.com/docs/commands/InputBox.htm

Examples

Basic Usage Example

```
InputBox, userinput
```

This will store what the user types into the input box in the variable named *userinput*

Passwords

```
InputBox, password, Enter your Password,, HIDE,, 100

Loop, {
  if (errorlevel = 1)
return

  if (password = "password") {
MsgBox, The password is correct.
    return
  } else if (password != "password") {
MsgBox, The password is incorrect.
InputBox, password, Enter your Password,, HIDE,, 100
  }
}
```

This will check if the user has typed "password" in the input box. If the user types the correct value, it will say "The password is correct." and close the input box. If the user types the incorrect value, it will say "The password is incorrect." and reopen the input box. If the errorlevel is 1 (the user pressed cancel), it will terminate the script.

Read Input Field online: https://riptutorial.com/autohotkey/topic/9917/input-field

Chapter 7: Open a File in a Script

Introduction

Different ways to open a file to work with in a script.

Examples

Open a File through Windows Explorer

Inside the script, use the first line to store the very first variable (in this example, `%1%`) with a name to deal with. Example: `OpenWithFile = %1%`

Once you open a file with this script through Windows (Right click on any file on MS Windows and choose 'Open with...' then select the **compiled** version of the script such as script.exe) the name of the choosed file will be stored in this variable and, so, the script will be able to work with it. Example:

```
OpenWithFile = %1%
if OpenWithFile !=
{
FileRead, content, %OpenWithFile%
msgbox %content%
return
}
```

Open a File through SelectFile dialog box

The following example creates a Gui with a single button wich brings the SelectFile dialog box.

```
Gui, Loader: New
Gui, Loader: Add, Button, Default Center w220 vLOAD, LOAD
Gui, Loader: Show, AutoSize Center, Loader

return

LoaderButtonLOAD:
FileSelectFile, LoadedFile, , , ,

if ErrorLevel=1
{
return
}

else
{
    FileRead, content, %LoadedFile%
    msgbox %content%
}
return
```

Open a File through Windows Drag n' Drop

This examples creates a new empty Gui sensible to Drag n' Drop event:

```
Gui, Dropper: New
Gui, Dropper: Font, s10 w700
Gui, Dropper: Add, Text, y80 vText1, Drag the files here
Gui, Dropper: Show, w200 h200 Center, Dropper

return

DropperGuiDropFiles:
DroppedFile:=A_GuiEvent

    FileRead, content, %DroppedFile%
    msgbox %content%

return
```

Read Open a File in a Script online: https://riptutorial.com/autohotkey/topic/9070/open-a-file-in-a-script

Chapter 8: Use functions instead of labels

Remarks

AutoHotkey used to heavily rely on labels until version 1.1.20. It's reliance on labels had very serious disadvantages. The main one being that labels usually execute in the global scope meaning that any variable defined within a label will be globally available. This sounds great until you realize that for example you can't just use other peoples libraries without making sure that their variables don't interfere with yours.
Working in the global scope when not necessary is simply bad practice.

So this is where functions come in. As of version 1.1.20, every AutoHotkey command that accepts a label-name as a parameter, now alternatively accepts a function-name.

Examples

Very basic example demonstrating function use on SetTimer.

```
;Sends the keystroke for the letter "a" every 3 seconds.
#Persistent
SetTimer, SendLetterA, 3000
return

SendLetterA() {
    Send, a
}
```

Advanced use of SetTimer: Calling the same function with different parameters

This is an example of something that would have been straight up impossible with labels. If you execute the same label multiple times at the same time and they rely on variables that are being defined within them, they very likely interfere and cause unexpected behavior.

Here is how to do it with functions:

```
; This script will switch between showing "Hello 1" and "Hello 2"

#Persistent
DisplayMessage_Hello1 := Func("DisplayMessage").bind("Hello 1")
SetTimer, %DisplayMessage_Hello1%, 2000

Sleep, 1000

DisplayMessage_Hello2 := Func("DisplayMessage").bind("Hello 2")
SetTimer, %DisplayMessage_Hello2%, 2000

DisplayMessage(messageToDisplay) {
    TrayTip ; remove other traytips
    TrayTip, Message to display:, %messageToDisplay%
}
```

Here is **how to not do it** (with labels):

```
;This script will never display the message "Hello 1". It will always show "Hello 2".

#Persistent
messageToDisplay := "Hello 1"
SetTimer, DisplayMessage, 2000

Sleep, 1000

messageToDisplay := "Hello 2"
SetTimer, DisplayMessage, 2000

DisplayMessage:
    TrayTip ; remove other traytips
    TrayTip, Message to display:, %messageToDisplay%
Return
```

Gui with functions instead of labels

Example showing how to create Guis using functions instead of labels.

```
Gui, Add, Button, gCtrlEvent vButton1, Button 1
Gui, Add, Button, gCtrlEvent vButton2, Button 2
Gui, Add, Button, gGoButton, Go Button
Gui, Add, Edit, vEditField, Example text
Gui, Show,, Functions instead of labels

CtrlEvent(CtrlHwnd:=0, GuiEvent:="", EventInfo:="", ErrLvl:="") {
    GuiControlGet, controlName, Name, %CtrlHwnd%
    MsgBox, %controlName% has been clicked!
}
GoButton(CtrlHwnd:=0, GuiEvent:="", EventInfo:="", ErrLvl:="") {
    GuiControlGet, EditField
    MsgBox, Go has been clicked! The content of the edit field is "%EditField%"!
}

GuiClose(hWnd) {
    WinGetTitle, windowTitle, ahk_id %hWnd%
    MsgBox, The Gui with title "%windowTitle%" has been closed!
    ExitApp
}
```

Hotkeys with functions instead of labels

Examples of using functions with hotkeys:

```
Hotkey, a, MyFunction ; Calls MyFunction() when a is pressed

MyFunction() {
    MsgBox You pressed %A_ThisHotkey%.
}
```

Or:

```
a::MyFunction()

MyFunction() {
    MsgBox You pressed %A_ThisHotkey%.
}
```

Tray menu actions with functions

```
#Persistent

Menu, Tray, NoStandard ; remove default tray menu entries
Menu, Tray, Add, MyDefaultAction, OnDefaultTrayAction ; add a new tray menu entry
Menu, Tray, Add, Exit, Exit ; add another tray menu entry
Menu, Tray, Default, MyDefaultAction ;When doubleclicking the tray icon, run the tray menu
entry called "MyDefaultAction".


OnDefaultTrayAction() {
    MsgBox, You double clicked the tray icon of this script or you clicked the MyDefaultAction
entry!
}

Exit() {
    MsgBox, You clicked the Exit entry! The script will close itself now.
    ExitApp
}
```

General example of functions vs labels

I'll demonstrate the basic usage of using functions vs using labels+gosub.
In this example we'll implement simple functionality to add two numbers and store them in a variable.

With functions:

```
c := Add(3, 2) ; function call
MsgBox, Result: %c%

Add(a, b) { ; This is a function. Put it wherever you want, it doesn't matter.
    ; the a and b inside of this function are set by the function call above
    Return a+b ; the function will return the result of the expression "a+b"
}
```

With labels (please don't do that):

```
a := 3
b := 2
GoSub, Add ; execute the label "Add" then jump back to the next line here
MsgBox, Result: %c%
Return ; without this, the label would be executed again for no reason.

Add: ; This is a label. Please put them at the bottom of your script and use "Return" in a
```

```
line above.
    c := a+b
Return
```

OnClipboardChange with fucntion/label

The following code is taken from he official AutoHotkey documentation:

Function implementation:

```
#Persistent
OnClipboardChange("ClipChanged")
return

ClipChanged(Type) {
    ToolTip Clipboard data type: %Type%
    Sleep 1000
    ToolTip  ; Turn off the tip.
}
```

Label implementation:

```
#Persistent
return

OnClipboardChange:
ToolTip Clipboard data type: %A_EventInfo%
Sleep 1000
ToolTip  ; Turn off the tip.
return
```

More complicated Gui example with multiple listviews using the same event callback function

This script demonstrates how to receive complicated GUI events from different controls in the same event callback function. We'll be using two ListView controls for that.
Now every time an action is detected on one of those ListView controls, we want a precise description of what happened and have that logged into an edit control in the same GUI.

```
Gui, Add, ListView, gListCtrlEvent vMyFirstListView AltSubmit -ReadOnly R10 w310,
ColumnTitle1|ColumnTitle2|ColumnTitle3
Gui, Add, ListView, gListCtrlEvent vMySecondListView AltSubmit -ReadOnly R10 w310,
ColumnTitle1|ColumnTitle2|ColumnTitle3
Gui, Add, Text, w310, Action Log
Gui, Add, Edit, vLog R7 w310,
Gui, Show,, Functions instead of labels

; Create example entries for the first ListView
Gui, ListView, MyFirstListView
Loop, 10 {
    LV_Add("", "Column-1 | Row-" A_Index ,  "Column-2 | Row-" A_Index,  "Column-3 | Row-"
A_Index)
}
LV_ModifyCol()
```

```
; Create example entries for the second ListView
Gui, ListView, MySecondListView
Loop, 10 {
    LV_Add("", "Column-1 | Row-" A_Index ,  "Column-2 | Row-" A_Index,  "Column-3 | Row-"
A_Index)
}
LV_ModifyCol()


ListCtrlEvent(ctrlHwnd:=0, guiEvent:="", eventInfo:="", errLvl:="") {
    GuiControlGet, ctrlName, Name, %CtrlHwnd%
    whatHappened := "Action detected!`n"
    whatHappened .= "Control handle: " ctrlHwnd "`n"
    whatHappened .= "Control name: " ctrlName "`n"

    If (guiEvent = "DoubleClick") {
        whatHappened .= "`nThe user has double-clicked within the control."
        whatHappened .= "`n> Focused row number: " eventInfo
    } Else If (guiEvent = "R") {
        whatHappened .= "`nThe user has double-right-clicked within the control."
        whatHappened .= "`n> Focused row number: " eventInfo
    } Else If (guiEvent = "ColClick") {
        whatHappened .= "`nThe user has clicked a column header."
        whatHappened .= "`n> Column number: " eventInfo
    } Else If (guiEvent = "D") {
        whatHappened .= "`nThe user has attempted to start dragging a row or icon."
        whatHappened .= "`n> Focused row number: " eventInfo
    } Else If (guiEvent = "d") {
        whatHappened .= "`nThe user has attempted to start right-click-dragging a row or
icon."
        whatHappened .= "`n> Focused row number: " eventInfo
    } Else If (guiEvent = "e") {
        whatHappened .= "`nThe user has finished editing the first field of a row."
        whatHappened .= "`n> Row number: " eventInfo
    } Else If (guiEvent = "Normal") {
        whatHappened .= "`nThe user has left-clicked a row."
        whatHappened .= "`n> Focused row number: " eventInfo
    } Else If (guiEvent = "RightClick") {
        whatHappened .= "`nThe user has right-clicked a row."
        whatHappened .= "`n> Focused row number: " eventInfo
    } Else If (guiEvent = "A") {
        whatHappened .= "`nA row has been activated."
        whatHappened .= "`n> Row number: " eventInfo
    } Else If (guiEvent = "C") {
        whatHappened .= "`nThe ListView has released mouse capture."
    } Else If (guiEvent = "E") {
        whatHappened .= "`nThe user has begun editing the first field of a row."
        whatHappened .= "`n> Row number: " eventInfo
    } Else If (guiEvent = "F") {
        whatHappened .= "`nThe ListView has received keyboard focus."
    } Else If (guiEvent = "f") {
        whatHappened .= "`nThe ListView has lost keyboard focus."
    } Else If (guiEvent = "I") {
        whatHappened .= "`nItem changed. (A row has changed by becoming selected/deselected,
checked/unchecked, etc.)"
        whatHappened .= "`n> Row number: " eventInfo
    } Else If (guiEvent = "K") {
        whatHappened .= "`nThe user has pressed a key while the ListView has focus."
        whatHappened .= "`n> Key pressed: " GetKeyName(Format("vk{:x}", eventInfo))
    } Else If (guiEvent = "M") {
        whatHappened .= "`nItem changed. (A row has changed by becoming selected/deselected,
checked/unchecked, etc.)"
        whatHappened .= "`n> Row number: " eventInfo
    } Else If (guiEvent = "S") {
        whatHappened .= "`nMarquee. The user has started to drag a selection-rectangle around
a group of rows or icons."
    } Else If (guiEvent = "s") {
        whatHappened .= "`nThe user has finished scrolling the ListView."
    }
    GuiControlGet, Log
    GuiControl,, Log, % whatHappened "`n---------------------`n" Log
}

GuiClose(hWnd) {
    WinGetTitle, windowTitle, ahk_id %hWnd%
    MsgBox, The Gui with title "%windowTitle%" is going to be closed! This script will exit
afterwards!
    ExitApp
}
```

Read Use functions instead of labels online: https://riptutorial.com/autohotkey/topic/4062/use-functions-instead-of-labels

Credits

S. No	Chapters	Contributors
1	Getting started with AutoHotkey	blackholyman, Community, depperm, Forivin, Joe DF, Shyam Sundar Shankar, tlm, Vijay
2	Arrays	blackholyman, errorseven
3	Built-in Variables and Functions	MCL
4	Hello World	errorseven, Forivin, Goerman, mikew, vasili111
5	Hotkey Scripts	depperm, MCL, user5226582
6	Input Field	Meh
7	Open a File in a Script	freestock.tk
8	Use functions instead of labels	Forivin