

AHK_X11 Help File

AHK_X11 is AutoHotkey for LINUX.
These docs are NOT FOR WINDOWS.
Windows users please go [here](#).

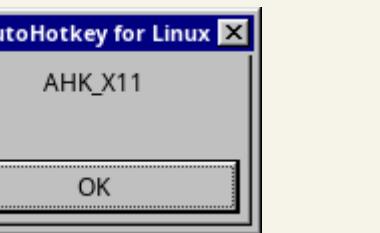
Legend: *Green*, *italic*: AHK_X11 only

Grey, strikethrough: Removed because it makes no sense or is impossible

Red, strikethrough: Not yet supported, but should be. This is **TODO**.

Table of contents

- [**AutoHotkey**](#)
- [Tutorial \(quick start\)](#)
- [FAQ \(Frequently Asked Questions\)](#)
- [Hotkeys](#)
- [Hotstrings & auto-replace](#)
- [Key List \(Keyboard, Mouse, Joystick\)](#)
- [Scripts](#)
- [Variables](#)
- [The "Last Found" Window](#)
- [Alphabetical List of Commands](#)
- [Recent Changes](#)
- [Environment Management](#)
 - [ClipWait](#)
- [File, Directory, and Disk Management](#)
 - [Drive](#)
 - [DriveGet](#)
 - [DriveSpaceFree](#)
 - [FileAppend](#)
 - [FileCopy](#)
 - [FileCopyDir](#)
 - [FileCreateDir](#)
 - [FileCreateShortcut](#)
 - [FileDelete](#)
 - [FileGetAttrib](#)
 - [FileGetShortcut](#)
 - [FileGetSize](#)



Version 1.0.24

https://github.com/phil294/AHK_X11

©2022 Philip Waritschlager

Original Docs (this site): ©2003-2004 Chris Mallett, portions ©AutoIt Team

- [FileGetTime](#)
- [FileGetVersion](#)
- [FileInstall](#)
- [FileMove](#)
- [FileMoveDir](#)
- [FileReadLine](#)
- [FileRead](#)
- [FileRecycle](#)
- [FileRecycleEmpty](#)
- [FileRemoveDir](#)
- [FileSelectFile](#)
- [FileSelectFolder](#)
- [FileSetAttrib](#)
- [FileSetTime](#)
- [IfExist/IfNotExist](#)
- [IniDelete](#)
- [IniRead](#)
- [IniWrite](#)
- [Loop_\(files & folders\)](#)
- [Loop_\(read file contents\)](#)
- [SetWorkingDir](#)
- [SplitPath](#)
- [Flow of Control](#)
 - [#Include/#IncludeAgain](#)
 - [Block](#)
 - [Break](#)
 - [Continue](#)
 - [Else](#)
 - [Exit](#)
 - [ExitApp](#)
 - [Gosub](#)
 - [Goto](#)

What is it?

- Free & open source ([GNU General Public License](#)).
- A simple yet powerful hotkey (shortcut key) scripting language, supporting both mouse and keyboard macros (if you're new to scripting, it might be easier than you think; check out the [quick-start tutorial](#)).
- A [word recognition engine](#) that expands abbreviations as you type them (auto-replace).
- A regular scripting language. The program includes a [script compiler](#) that converts a script to an [EXE executable](#). It also includes [AutoScriptWriter](#), a macro recorder written by Larry Keys.
- A keyboard, joystick, and mouse [remapper](#).
- *AHK_X11 is a complete reimplementations of AutoHotkey for X11-based systems such as most Linux systems. It is still in development as of 2022/08. A lot of work is done but several commands are still missing. You can find installation details below and/or on the GitHub page linked above.*
- *These docs are based on the [original AHK docs from December 2004](#), complemented by annotations (see Legend above - not super accurate though). Black text was not changed and should work (except some code samples). AHK_X11 only aims at implementing the spec from that time. This "Classic AHK" approach means that more modern features such as expressions or functions are not supported.*
- *In short, this doc file should be accurate and tell you everything you need.*

- [Loop](#)
- [Loop_\(files & folders\)](#)
- [Loop_\(parse a string\)](#)
- [Loop_\(read file contents\)](#)

- [Loop_\(registry\)](#)

- [OnExit](#)

- [Pause](#)

- [Return](#)

- [SetBatchLines](#)

- [SetTimer](#)

- [Sleep](#)

- [Suspend](#)

- [GUI, MsgBox, InputBox & Other Dialogs](#)

- [FileSelectFile](#)

- [FileSelectFolder](#)

- [Gui](#)

- [GuiControl](#)

- [GuiControlGet](#)

- [IfMsgBox](#)

- [InputBox](#)

- [MsgBox](#)

- [Progress](#)

- [SplashImage](#)

- [SplashTextOn/SplashTextOff](#)

- [ToolTip](#)

- [TrayTip](#)

- [Keyboard Control](#)

- [Hotkeys](#)

- [#HotkeyInterval](#)

- [#HotkeyModifierTimeout](#)

- [#Hotstring](#)

- [#MaxHotkeysPerInterval](#)

Who might benefit?

- Those wishing to automate repetitive tasks.
- Those having an interest in creating hotkeys that:
 - aren't limited to only those combinations allowed by Windows;
 - are more flexible and capable of greater complexity than those of most other hotkey apps;
 - are more responsive (take effect more quickly) than standard Windows hotkeys.
- Anyone concerned about, or already suffering from Repetitive Stress Injury (RSI). For example, the mouse wheel can be used as an entire substitute for Alt-Tab.

- [#MaxThreads](#)
- [#MaxThreadsBuffer](#)
- [#MaxThreadsPerHotkey](#)
- [Hotkey](#)
- [List Hotkeys](#)
- [Pause](#)
- [Reload](#)
- [Suspend](#)

- [BlockInput](#)
- [ControlSend/ControlSendRaw](#)
- [GetKeyState](#)
- [Key List \(Keyboard, Mouse, Joystick\)](#)
- [KeyHistory](#)
- [KeyWait](#)
- [Input](#)
- [Send/SendRaw](#)
- [SetKeyDelay](#)
- [SetNumScrollCapsLockState](#)
- [SetStoreCapslockMode](#)

- [Math Commands](#)
 - [EnvAdd \(+=, ++\)](#)
 - [EnvDiv \(/=\)](#)
 - [EnvMult \(*=\)](#)
 - [EnvSub \(-=, --\)](#)
 - [If/IfEqual/IfLess/IfGreater](#)
 - [If var \[not\] between Low and High](#)
 - [If var is \[not\] type](#)
 - [Random](#)
 - [SetFormat](#)
 - [Transform](#)

- [Misc. Commands](#)
 - [#NoTrayIcon](#)

More benefits

- Use ELSE with every IF-type command.
- Use blocks (multi-line sections) with IFs and ELSEs (virtually eliminating the need for Goto).
- Use break and continue inside loops (another way to eliminate Goto's).
- Use a set of new commands and benefit from enhancements to old ones.
- Catch silly errors and typos early (at load-time) rather than later (during runtime) when you may be less prepared to deal with them: Each script is thoroughly checked before it begins running.
- Take complete control of your keyboard, joystick, and mouse: AutoHotkey has far better hotkey support than Windows provides (and is more responsive too). It's also more powerful than most commercially available hotkey applications. For example, Windows' built-in hotkeys, such as Win-E and Win-R, can be selectively overridden. Also, any two keys and/or mouse buttons can be combined to become a hotkey.
- Gain a significant boost in performance.

- [#SingleInstance](#)
- [AutoTrim](#)
- [BlockInput](#)
- [CoordMode](#)
- [ClipWait](#)
- [Echo](#)
- [Edit](#)
- [Eval](#)
- [#DefineCommand](#)
- [ListLines](#)
- [ListVars](#)
- [EnvGet](#)
- [EnvSet](#)
- [Menu](#)
- [PixelGetColor](#)
- [PixelSearch](#)
- [Reload](#)
- [SetBatchLines](#)
- [SetEnv \(var = value\)](#)
- [SetTimer](#)
- [SysGet](#)
- [Thread](#)
- [Transform](#)
- [URLDownloadToFile](#)
- [Mouse Control](#)
 - [ControlClick](#)
 - [Click](#)
 - [MouseClick](#)
 - [MouseClickDrag](#)
 - [MouseGetPos](#)
 - [MouseMove](#)
 - [SetDefaultMouseSpeed](#)

Acknowledgements

A special thanks to Jonathan Bennett, whose generosity in releasing AutoIt2 as free software in 1999 served as an inspiration and time-saver for myself and many others worldwide. In addition, many of AutoHotkey's enhancements to the AutoIt2 command set, as well as the Window Spy and the script compiler, were adapted directly from the AutoIt3 source code. So thanks to Jon and the other AutoIt3 authors for those as well.

Finally, AutoHotkey would not be what it is today without [these other individuals](#).

#

[↑ Back To top](#)

- [SetMouseDelay](#)
 - [Process Management](#)
 - [Exit](#)
 - [ExitApp](#)
 - [OnExit](#)
 - [Process](#)
 - [Run/RunWait](#)
 - [RunAs](#)
 - [Shutdown](#)
 - [Sleep](#)
 - [Sound Commands](#)
 - [SoundGet](#)
 - [SoundGetWaveVolume](#)
 - [SoundPlay](#)
 - [SoundSet](#)
 - [SoundSetWaveVolume](#)
 - [String Management](#)
 - [FormatTime](#)
 - [If/IfEqual/IfLess/IfGreater](#)
 - [IfInString/IfNotInString](#)
 - [If var.\[not\] in/contains MatchList](#)
 - [If var is \[not\] type](#)
 - [Loop.\(parse a string\)](#)
 - [SetEnv \(var = value\)](#)
 - [SetFormat](#)
 - [Sort](#)
 - [StringCaseSense](#)
 - [StringGetPos](#)
 - [RegExGetPos](#)
 - [StringLeft/StringRight](#)
 - [StringLen](#)
 - [StringLower/StringUpper](#)
- AutoHotkey Tutorial and Overview**
- This brief introduction will help you start scripting your own macros and hotkeys right away.

- [StringMid](#)
- [StringReplace](#)
- [RegExReplace](#)
- [StringSplit](#)
- [StringTrimLeft/StringTrimRight](#)

- [Window Management](#)

- [Controls](#)
 - [Control](#)
 - [ControlClick](#)
 - [ControlFocus](#)
 - [ControlGet](#)
 - [ControlGetFocus](#)
 - [ControlGetPos](#)
 - [ControlGetText](#)
 - [ControlMove](#)
 - [ControlSend/ControlSendRaw](#)
 - [ControlSetText](#)
 - [Menu](#)
 - [SetControlDelay](#)
 - [WinMenuItemSelect](#)

- [Window Groups](#)

- [GroupActivate](#)
- [GroupAdd](#)
- [GroupClose](#)
- [GroupDeactivate](#)

- [#WinActivateForce](#)

- [DetectHiddenText](#)

- [DetectHiddenWindows](#)

- [IfWinActive/IfWinNotActive](#)

- [IfWinExist/IfWinNotExist](#)

- [SetTitleMatchMode](#)

- [SetWinDelay](#)

Tutorial Contents

- Installation
- Creating a script
- Launching a program or document
- Sending keystrokes & mouse clicks
- Activating and manipulating windows
- Getting input from the user with MsgBox, InputBox, etc.
- Using variables and the clipboard
- Manipulating files and folders
- Overview of other features

Installation

1. Download the program from [here](#)
2. Extract it
3. Run it, for example by double clicking. If this does not work, then you also need to mark it as executable. This usually works via right click / file properties.
4. Now the installer should open. Click install and your setup is complete.
5. Please be aware that so far, there is *no* auto-update function. New features are being added all the time right now, so you should probably repeat the above steps then and again. The documentation (this very site) will always be up to date.

Creating a script

1. Open Windows Explorer and select a directory of your choice.
2. Pull down the File menu and choose New >> AutoHotkey Script (or Text Document).
3. Type a name for the file, ensuring that it ends in .ahk. Example: Test.ahk
4. Right-click the file and choose Edit Script. [Open with...](#) -> Your text editor of choice (but *don't* set it as default)
5. On a new blank line, type the following:
`#z::Run www.google.com`

The # means the Windows key ("Super key"), so the #z means holding down the Windows key then pressing Z to activate a hotkey. The :: means that the subsequent command should be executed whenever this hotkey is pressed, in this case to go to the Google web site. To try out this script, continue as follows:

1. Save and close the file.
2. In Windows Explorer, double-click the script to launch it. A new tray icon appears.
3. Press Win+Z. A web page opens in the default browser.
4. To exit or edit the script, right click its tray icon.

- [StatusBarGetText](#)
- [StatusBarWait](#)
- [WinActivate](#)
- [WinActivateBottom](#)
- [WinClose](#)
- [WinGet](#)
- [WinGetActiveState](#)
- [WinGetActiveTitle](#)
- [WinGetClass](#)
- [WinGetPos](#)
- [WinGetText](#)
- [WinGetTitle](#)
- [WinHide](#)
- [WinKill](#)
- [WinMaximize](#)
- [WinMinimize](#)
- [WinMinimizeAll/WinMinimizeAllUndo](#)
- [WinMove](#)
- [WinRestore](#)
- [WinSet](#)
- [WinSetTitle](#)
- [WinShow](#)
- [WinWait](#)
- [WinWaitActive/WinWaitNotActive](#)
- [WinWaitClose](#)
- [#Directives](#)
 - [#CommentFlag](#)
 - [#DefineCommand](#)
 - [#ErrorStdOut](#)
 - [#EscapeChar](#)
 - [#HotkeyInterval](#)
 - [#HotkeyModifierTimeout](#)

Note: Each script can contain any number of hotkeys (in the absence of hotkeys, a script will perform a series of sequential actions the moment it is launched). In addition, multiple scripts can be running simultaneously, each with its own tray icon.

Launching a program or document

The [Run](#) command is used to launch a program, document, URL, or shortcut, as in the following examples:

Run, gedit

Run, %A_Home%/Documents/Address List.doc

Run, www.yahoo.com

Run, mailto:someone@anywhere.com

A hotkey can be assigned to any of the above examples by including a [hotkey label](#). In the first example below, the assigned hotkey is Win+N, while in the second it is Control+Alt+C:

#n::Run, gedit

^!c::Run, calc.exe

The above examples are known as single-line hotkeys because each consists of only one command. To have more than one command executed by a hotkey, put the first line [beneath](#) the hotkey definition and make the last line a [return](#). For example:

#n::

Run, www.google.com

Run, gedit

return

If the program or document to be run is not integrated with the system, specify its full path to get it to launch:

Run, %A_Home%/bin/program

In the above example, %A_Home% is an [environment variable](#) maintained by the operating system. By using it rather than something like /home/yourname, the script is made more portable, meaning that it would be more likely to run successfully on another computer with another username.

To have the script wait for the program or document to close before continuing, use [RunWait](#) instead of Run. In this example, the [MsgBox](#) command will not run until after the user closes Gedit (untested):

RunWait, gedit

MsgBox, The user has finished (Gedit has been closed).

This topic continues [here](#) with the following advanced information: launching a program in a minimized/maximized/hidden state, specifying a working directory, passing parameters, using system verbs, and discovering a program's exit code.

Sending keystrokes and mouse clicks

Keystrokes are sent to the active (foremost) window by using the [Send](#) command. In the following example, Win+S becomes a hotkey to type a signature. Be sure that a window such as an editor or draft e-mail message is active before pressing the hotkey:

#s::

Send, Sincerely,{enter}John Smith

return

- [#Hotstring](#)
- [#Include/#IncludeAgain](#)
- [#MaxHotkeysPerInterval](#)
- [#MaxMem](#)
- [#MaxThreads](#)
- [#MaxThreadsBuffer](#)
- [#MaxThreadsPerHotkey](#)
- [#NoTrayIcon](#)
- [#Persistent](#)
- [#SingleInstance](#)
- [#WinActivateForce](#)

In the above example, all characters are sent literally except {enter} (which results in a simulated press of the Enter key). The next example illustrates some of the other commonly used special characters:

Send, ^c!{tab}pasted:^v

The above example sends a Control+C followed by an Alt+Tab followed by the string "pasted:" followed by a Control+V. See the [Send](#) command for a complete list of special characters and keys.

Mouse Clicks: To send a mouse click to a window it is first necessary to determine the X & Y coordinates where the click should occur. This can be done with either [AutoScriptWriter](#) or [Window Spy](#), which is included with AutoHotkey. The following steps apply to the Window Spy method:

1. Launch Window Spy from the program's tray-icon menu or the Start Menu.
2. Activate the window of interest either by clicking its title bar, alt-tabbing, or other means (Window Spy will stay "always on top" by design).
3. Move the mouse cursor to the desired position in the target window and write down the mouse coordinates displayed by Window Spy (or press Shift+Alt+Tab to activate Window Spy so that the "frozen" coordinates can be copied and pasted).
4. Use the coordinates discovered above with the [MouseClick](#) command as in this example, which clicks the left mouse button:
MouseClick, left, 112, 223

To move the mouse without clicking, use [MouseMove](#). To drag the mouse, use [MouseClickDrag](#).

Activating and manipulating windows

To activate a window (make it foremost), use [WinActivate](#). To detect whether a window exists, use [IfWinExist](#) or [WinWait](#). The following example illustrates these commands:

```
IfWinExist, Untitled - Notepad
{
    WinActivate
}
else
{
    Run, Notepad
    WinWait, Untitled - Notepad
    WinActivate
}
```

The above example first searches for any existing window whose title starts with "Untitled - Notepad" (case sensitive). If such a window is found, it is activated. Otherwise, Notepad is launched and the script waits for the Untitled window to appear, at which time it is activated. The above example also utilizes the [last found window](#) to avoid the need to specify the window's title with each WinActivate.

Some of the other commonly used window commands are:

- [WinWaitActive](#): Waits for a window to become active (typically used immediately after a [Run](#) command).
- [IfWinActive](#): Checks if a window is currently active.
- [WinClose](#): Closes a window.
- [WinMove](#): Moves or resizes a window.
- [WinMinimize](#), [WinMaximize](#), [WinRestore](#): Minimizes, maximizes, or restores a window, respectively.

Getting input from the user with MsgBox, InputBox, etc.

The following example displays a dialog with two buttons (YES and NO):
`MsgBox`, 4,, Would you like to continue?
`IfMsgBox`, No, return
; Otherwise, the user picked yes.
`MsgBox`, You pressed YES.

Use the `InputBox` command to prompt the user to type a string. Use `FileSelectFile` or `FileSelectFolder` to have the user select a file or folder. For more advanced tasks, use the `Gui` command to create custom data entry forms and user interfaces.

Using variables and the clipboard

All variables are global; that is, their contents may be read or altered by any `subroutine` in the entire script. In addition, there are no data types: all variables contain strings (sequences of characters or digits). Finally, variables do not need to be declared: the program ensures they exist when needed.

To assign a string to a variable, follow these examples:

```
MyVar1 = 123
MyVar2 = my string
```

To compare the contents of a variable to a string, follow these examples:

```
if MyVar2 = my string
{
    MsgBox MyVar2 contains the string "my string".
}
if MyVar1 >= 100
{
    MsgBox MyVar1 contains a number greater than or equal to 100.
}
```

To assign the contents of a variable to a second variable, enclose the first variable's name in percent signs. In the example below, `MyVarConcatenated` will be assigned the string "123 my string" (without the double quotes):

```
MyVarConcatenated = %MyVar1% %MyVar2%
```

To compare the contents of a variable with that of another, consider this example:

```
MyVar = 200
MyVarLimit = 100

if MyVar > %MyVarLimit%
{
    MsgBox The number in MyVar, which is %MyVar%, is beyond the limit of %MyVarLimit%.
```

By contrast, here are some examples of incorrect variable usage:

```
MyVar2 = MyVar1 ; Wrong because it assigns the literal string "MyVar1" to MyVar2.  
if MyVar > MyVarLimit ; Wrong because there should be percent signs around MyVarLimit.  
if %MyVar% > %MyVarLimit% ; Wrong because MyVar should not have percent signs.
```

As seen in the examples above, the rule of thumb is to enclose a variable in percent signs whenever it is used on the right side of an assignment or comparison.

Math: AutoHotkey is currently limited to one math operation per line. For example, an assignment that might be written as "x = y + 5" in another language would need to be done in two steps:

```
x = %y%  
x += 5
```

In the example above, the `+=` symbol means "increase the variable on the left by the number at the right". In a similar way, subtraction, multiplication, and division are accomplished via `-=`, `*=`, and `/=`, respectively. Advanced math operations such as logarithms, trigonometry, raising a number to a power, and bitwise manipulations are possible via the [Transform](#) command.

Clipboard: The variable named Clipboard is a special variable containing the current text on the Windows clipboard. It can be used just like any other variable. For example, the following line replaces the current contents of the clipboard with new text:

```
clipboard = A line of text.`nA second line of text.`n
```

In the above line, ``n` (accent followed by the letter "n") is a special string used to indicate a special character: linefeed. This character starts a new line of text as though the user had pressed Enter.

To append text to the clipboard (or any other variable), follow this example:

```
clipboard = %clipboard% And here is the text to append.
```

See the [clipboard](#) and [variables](#) sections for more details.

Manipulating files and folders

To append text to a file, consider the following example. Note that it uses ``n` (linefeed) to start a new line of text afterward:

```
FileAppend, A line of text to append.`n, C:\My Documents\My Text File.txt
```

Some of the other commonly used file and folder commands are:

- [File-reading Loop](#): Retrieve the lines in a text file, one by one.
- [IfExist](#): Determine whether a file or folder exists.
- [File Loop](#): Retrieve the files and folders contained in a folder, one at a time.
- [FileCopy](#): Copy one or more files. [Use FileCopyDir to copy an entire folder.](#)
- [FileMove](#): Move or rename one or more files. [Use FileMoveDir to move an entire folder.](#)
- [FileDelete](#): Delete one or more files. Use [FileRemoveDir](#) to delete an entire folder.
- [FileRecycle](#): Send one or more files (or an entire folder) to the recycle bin.
- [FileSelectFile](#) and [FileSelectFolder](#): Display a dialog for the user to pick a file or folder.
- [FileSetAttrib](#) and [FileSetTime](#): Change the attributes or timestamp of one or more files.
- [IniRead](#), [IniWrite](#), and [IniDelete](#): Create, access, and maintain standard-format INI files.

- ~~RegRead, RegWrite, RegDelete, and Registry Loop: Work with the Windows registry.~~

Overview of other features

(this section is still under development, see the [command list](#) for an overview of every command

#

AutoHotkey Frequently Asked Questions (FAQ)

When are quotation marks used with commands and their parameters?

Single quotes ('') and double quotes ("") have no special meaning within the AutoHotkey language; they are always treated literally as if they were normal characters. However, when AutoHotkey launches a program or document, the operating system usually requires double quotes around any command-line parameter that contains spaces, such as in this example: Run, gedit "/home/Documents/Address List.txt"

Why is the **Run** command unable to launch my game or program?

Some programs need to be started in their own directories (when in doubt, it is usually best to do so). Example:
Run, /opt/app/bin, /opt/app

I'm having trouble getting my mouse buttons working as hotkeys. Any advice?

The left and right mouse buttons should be assignable normally (for example, "#LButton::" is the Win+LeftButton hotkey). Similarly, the middle button and the turning of the [mouse wheel](#) should be assignable normally except on mice whose drivers directly control those buttons.

The fourth button (XButton1) and the fifth button (XButton2) might be assignable if your mouse driver allows their clicks to be [seen](#) by the system. If they cannot be seen -- or if your mouse has more than five buttons -- you can try configuring the software that came with the mouse (sometimes accessible in the Control Panel or Start Menu) to send a keystroke whenever you press one of these buttons. Such a keystroke can then be defined as a hotkey in a script. For example, if you configure the fourth button to send Control+F12, you can then indirectly configure that button as a hotkey by using ^F12:: in the script.

If you have a five-button mouse whose fourth and fifth buttons cannot be [seen](#), you can try changing your mouse driver to the default driver included with the OS. This assumes there is such a driver for your particular mouse and that you can live without the features provided by your mouse's custom software.

How can symbols and punctuation marks be defined as hotkeys?

Symbols that don't require the shift key to be held down can be defined normally (except for semicolon, which should be defined as `::;). For example, this line would make accent (`) become a hotkey: `::MsgBox, Accent was pressed.

However, symbols that require the shift key to be held down (such as @ and ?) can be defined as hotkeys by including the SHIFT modifier in the definition of the symbol's unshifted key. For example, this line would make Control-Asterisk become a hotkey: ^+8::MsgBox, Ctrl-* was pressed.

How can a combination of modifier keys be made a hotkey?

The modifier symbols must always apply to a named key. For example, to define Ctrl+Alt+Win as a hotkey, use ^!LWin or ^!RWin rather than ^!#. To define the spacebar, use ^Space rather than "^ ". Note that the modifier keys must be pressed before the named key, so in the first example Ctrl+Alt must be held down prior to pressing LWin or RWin.

How can keys or mouse buttons be remapped so that they become different keys?

This is described on the [remapping](#) page.

How can a hotkey be made exclusive to certain program(s)? In other words, I want a certain key to act as it normally does except when a specific window is active.

In the following example, NumpadEnter is made to perform normally except when a window titled "CAD Editor" is active. Note the use of the \$ prefix in "\$NumpadEnter", which is required to let the hotkey "send itself":

```
$NumpadEnter::  
IfWinNotActive, CAD Editor  
{  
    Send, {NumpadEnter}  
    return  
}  
; Otherwise, the desired application is active, so do a custom action:  
Send, abc  
return
```

This next example is more pure than the above, but it will only work if the "CAD Editor" application is designed to ignore the NumpadEnter key itself. The tilde prefix (~) makes NumpadEnter into a non-suppressed hotkey, meaning that the NumpadEnter keystroke itself is always sent to the active window, the only difference being that it triggers a hotkey action:

```
~NumpadEnter::  
IfWinNotActive, CAD Editor  
    return  
Send, abc  
return
```

How can a repeating action be stopped without exiting the script?

To stop an action that is repeating inside a [Loop](#), consider the following example hotkey, which both starts and stops its own repeating action. In other words, pressing the hotkey once will start the Loop. Pressing the same hotkey again will stop it.

```
#MaxThreadsPerHotkey 3  
#z::  
#MaxThreadsPerHotkey 1  
if KeepWinZRunning = y ; This means an underlying thread is already running the loop below.  
{  
    KeepWinZRunning = ; Make it blank to signal that thread's loop to stop.  
    return ; End this thread so that the one underneath will resume and see the change.  
}  
; Otherwise:
```

```
KeepWinZRunning = y
Loop,
{
    ToolTip, Press Win-Z again to stop this from flashing.
    Sleep, 1000
    ToolTip
    if KeepWinZRunning = ; The user signaled the loop to stop by pressing Win-Z again.
        break ; Break out of this loop.
    Sleep, 1000
}
return
```

Note: To pause or resume the entire script at the press of a key, assign a hotkey to the [Pause](#) command as in this example:

[^!p::Pause](#) ; Press Ctrl+Alt+P to pause. Press it again to resume.

How can a prefix key be made to perform its native function rather than doing nothing?

Consider the following example, which makes Numpad0 into a prefix key:

~~Numpad0 & Numpad1::MsgBox, You pressed Numpad1 while holding down Numpad0.~~

~~Now, to make Numpad0 send a real Numpad0 keystroke whenever it wasn't used to launch a hotkey such as the above, add the following hotkey. The \$ prefix is needed to prevent a warning dialog about an infinite loop (since the hotkey "sends itself"). In addition, the below action occurs at the time the key is released:~~

~~\$Numpad0::Send, {Numpad0}~~

How to avoid trouble with ALT hotkeys that use the Send command?

The trouble is caused by the fact that the ALT key tends to activate the menu bar of the active window. The following example shows a workaround that may fix the problem in some cases: [!a::Send, {AltUp}{Alt}Text To Send](#).

How can context sensitive help for AutoHotkey commands be used in any editor?

Rajat created [this script](#).

My keypad has a special 000 key. Is it possible to turn it into a hotkey?

~~You can, but only if you're running Windows NT, 2000, XP, or beyond. This example script makes the 000 key into an equals key. You can change the action by replacing the "Send, =" line with line(s) of your choice.~~

How can the output of a command line operation be retrieved?

A small freeware utility cb.zip captures up to 512 KB of output from a command or program. The text is captured to the clipboard, which a script can access via the clipboard variable. Example:
RunWait %comspec% /c dir | cb.exe
MsgBox %clipboard% *This is possible with the optional option OutputVarStdout.*

How can Winamp be controlled even when it isn't active?

See [Automating Winamp](#).

How to detect when a web page is finished loading?

The technique in the following example will work with MS Internet Explorer for most pages. A similar technique might work in other browsers:

```
Run, www.yahoo.com
MouseMove, 0, 0 ; Prevents the status bar from showing a mouse-hover link instead of "Done".
WinWait, Yahoo! - Microsoft Internet Explorer
WinActivate
StatusBarWait, Done, 30
if ErrorLevel <> 0
    MsgBox, The wait timed out or the window was closed.
else
    MsgBox, The page is done loading.
```

How can dates and times be compared or manipulated?

The [EnvAdd](#) command can add or subtract a quantity of days, hours, minutes, or seconds to a time-string that is in the [YYYYMMDDHH24MISS](#) format. The following example subtracts 7 days from the specified time: EnvAdd, VarContainingTimestamp, -7, days

To determine the amount of time between two dates or times, see [EnvSub](#), which gives an example. Also, the built-in variable [A_Now](#) contains the current local time.

How can the built-in Windows shortcut keys, such as Win+U (Utility Manager) and Win+R (Run), be changed or disabled?

Here are some [examples](#).

How can [MsgBox](#)'s button names be changed?

Here is an [example](#).

How can performance be improved for games or at other times when the CPU is under heavy load?

If you find that [hotkeys](#) and/or [GetKeyState](#) are less responsive than expected while the CPU is under heavy load, raising the script's priority may help. To do this, include the following line near the top of the script, which increases its priority from normal to high. This tells the operating system to pay more attention to it:

~~Process, Priority, , High~~

#

Hotkeys

Hotkeys are defined by creating a subroutine label in a script file with two colons instead of one. In the below example, Win-Y will become a hotkey which implicitly [gosubs](#) the #y label, executing the statements until the first [return](#) or [exit](#) is encountered:

```
#y::  
WinActivate, Untitled - Notepad  
return
```

If a hotkey label needs only a single line, it can be included to the right of the double-colon. In other words, the [return](#) is implicit:

```
#y::WinActivate, Untitled - Notepad
```

You can use the following symbols to define hotkeys:

Sy mb ol	Description
#	Win (Windows Key)
!	Alt
^	Control
+	Shift
<	Use the left key of the pair. e.g. <!a is the same as !a except that only the left Alt key will trigger it.
>	Use the right key of the pair.
<^	# AltGr (alternate graving). If your keyboard layout has an AltGr key instead of a right-Alt key, this series of symbols can usually be used to stand for AltGr (requires Windows NT/2k/XP+). For example: <^>!m::MsgBox You presssed AltGr+m. >!m::MsgBox You pressed Ctrl+Alt+m.
*	Wildcard: Fire the hotkey even if extra modifiers are being held down. Example: *ScrollLock::Run, Notepad ; Launch notepad regardless of whether any control/alt/shift/win keys are also down. This feature is not supported in Win95/98/ME.
~	This hotkey's native function shouldn't be suppressed (hidden from the system) when the hotkey fires. For example, in the below, the mouse button click will be sent to the active window when it normally wouldn't be: ~RButton::Sleep, 1 ~RButton & C::Run, Calc Note: Special hotkeys that are substitutes for alt tab always ignore this setting.

\$	<p># This is usually only necessary if the script uses the Send command to send the keys that comprise the hotkey itself, which would otherwise cause it to fire unintentionally. The exact behavior of the \$ prefix varies depending on operating system: On Windows 95/98/Me and AutoHotkey v1.0.23+: The hotkey is disabled during the execution of its thread and re-enabled afterward. As a side effect, if #MaxThreadsPerHotkey is set higher than 1, it will behave as though set to 1 for such hotkeys. On other operating systems and for all versions of AutoHotkey: The \$ prefix forces the keyboard hook to be used to implement this hotkey, which as a side effect prevents the Send command from triggering it. The \$ prefix is equivalent to having specified #UseHook prior to the definition of this hotkey.</p> <p><i>This flag is the default and cannot be configured. Hotkeys can arbitrarily send their own or other grabbed keys because all hotkeys are disabled while any Send-like action takes place.</i></p>
UP	<p>The word UP may follow the name of a hotkey to cause the hotkey to fire upon release of the key rather than when the key is pressed down. The following example remaps the left Win to become the left Ctrl:</p> <pre>*LWin::Send {LControl down} *LWin Up::Send {LControl up}</pre> <p><i>Please note: This only works unreliable, and in the above example, LControl will be properly emulated, however, the CTRL effect will not be active. It is unsure if this is possible at all with X11 requests; help needed. In other words, you can currently only somewhat remap non-modifier keys such as A or Tab.</i></p> <p><i>"Up" can also be used with normal hotkeys as in this example: ^!r Up::MsgBox You pressed and released Ctrl+Alt+R.</i></p>

(See the [Key List](#) for a complete list of key and mouse/joystick button names)

Hotkey labels can be used as if they were normal labels: You can [Gosub/Goto](#) them and define more than one hotkey label to do the same subroutine, such as in this example:

```
^Numpad0::  
^Numpad1::  
MsgBox, Both of the above hotkeys will execute this subroutine.  
return
```

A key or key combination can be disabled for the entire system by having it do nothing. For example, to disable the right Windows key:
RWIN::return

Hotkeys can be also be created, disabled, or enabled while the script is running by using the [Hotkey](#) command.

Note: The program is [quasi multi-threaded](#), which allows a new hotkey to be launched even when a previous hotkey subroutine is still running. For example, new hotkeys can be launched even while a [MsgBox](#) is being displayed by the current hotkey.

See [advanced features](#) for an overview. Here is some more detailed information:

You can define a custom combination of two keys by using " & " between them. In the below example, you would hold down Numpad0 then press the second key to trigger the hotkey:

```
Numpad0 & Numpad1::AltTab  
Numpad0 & Numpad2::ShiftAltTab
```

In the above example, Numpad0 becomes a prefix key. Prefix keys can also be assigned their own actions such as in this example, but their action will only be triggered when the key is released and didn't modify any hotkeys while it was held down:
Numpad0::Run, calc.exe

In v1.0.10+, if the tilde (~) operator is used with a prefix key even once, that prefix will always be sent through to the active window. For example, in both of the below hotkeys, the right click will be non-suppressed even though only one of the definitions contains a tilde:

```
~RButton & LButton::MsgBox You pressed the left mouse button while holding down the right.  
RButton & WheelUp::MsgBox You turned the mouse wheel up while holding down the right button.
```

AltTab and ShiftAltTab are two of the special commands that are only recognized when used with hotkeys. They can also be used with mouse buttons and mousewheel rotation, as can any other script commands as well. Example:

```
LAlt & LButton::AltTab  
LAlt & WheelDown::AltTab  
LAlt & WheelUp::ShiftAltTab
```

The mouse wheel can be made into an entire substitute for Alt-tab. Clicking the button will display the menu and turning the wheel will navigate through it:

```
MButton::AltTabMenu  
WheelDown::AltTab  
WheelUp::ShiftAltTab
```

The other Alt-tab actions available are AltTabAndMenu (if the menu is displayed, move forward in it; otherwise, display the menu) and AltTabMenuDismiss (close the Alt-tab menu). Currently, all Alt-tab actions must be assigned directly to a hotkey as in the examples above (i.e. they can't be used as though they were commands).

You can force the Numlock, Capslock, and Scrolllock keys to be AlwaysOn or AlwaysOff. For example:

```
SetNumlockState, AlwaysOn
```

Windows' built-in hotkeys such as Win-E (#e) and Win-R (#r) can be individually overridden by assigning them to an action in the script. This feature is not implemented on Win95/98/ME due to the limitations of those OSes. See the override page for details.

Each numpad key can be made to launch two different hotkey subroutines depending on the state of Numlock. Alternatively, a numpad key can be made to launch the same subroutine regardless of the Numlock state, such as in this example:

```
NumpadEnd::  
Numpad1::  
MsgBox, This hotkey is launched regardless of whether Numlock is on.  
return
```

Magic words, abbreviation expansion (auto-replace), and "hot strings" (requires XP/2k/NT): These are described in the [hotstrings section](#).

Hotstrings and Auto-replace

Although hotstrings are mainly used to expand abbreviations as you type them (auto-replace), they can also be used to launch any scripted action. In this respect, they are similar to [hotkeys](#) except that they are typically composed of more than one character (that is, a string).

When the user types a hotstring, its action is triggered, much like a hotkey. However, unlike hotkeys, the keystrokes that comprise a hotstring are never suppressed, that is, they are never hidden from the active window.

To define a hotstring, enclose the triggering abbreviation between pairs of double colons as in this example:

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

In the above example, the abbreviation btw will be automatically replaced with "by the way" whenever the user types it. However, by default the user must complete the abbreviation by typing one of the standard ending characters such as space, period, or enter. Ending characters consist of the following: -**(>[]{}';"/\..?!`n `t** (note that `n is Enter, `t is Tab, and that there is a plain space between `n and `t). This set of characters can be changed by following this example, which sets the new ending characters for all hotstrings, not just the ones physically beneath it:

```
#Hotstring EndChars -[]{}';"/\..?!`n `t
```

The "by the way" example above is known as an auto-replace hotstring because the typed text is automatically erased and replaced by the string specified after the second pair of colons. By contrast, a hotstring may also be defined to perform any custom action as in this example:

```
::btw::  
MsgBox You typed "btw".  
return
```

Even though the above is not an auto-replace hotstring, the abbreviation typed by the user is erased by default. This is done via automatic backspacing, which can be disabled as described in the options below.

Options

A hotstring's default behavior can be changed in two possible ways:

1. The [#Hotstring](#) directive, which affects all hotstrings physically beneath that point in the script. Example:

```
#Hotstring c r ; Case sensitive and "send raw"
```

2. By including one or more of the following options inside a hotstring's first pair of colons. Example:

```
:cr:Btw::By the way ; Case sensitive and "send raw"
```

Note: When specifying more than one option from the list below, spaces may be optionally included between them.

B0 (B followed by a zero): Automatic backspacing is not done to erase the abbreviation typed by the user. Use a plain **B** to turn backspacing back on after it was previously turned off.

C: Case sensitive: When typed by the user, the abbreviation must exactly match the case defined in the script. Use **C0** to turn case sensitivity back off. Note: If you need to define two hotstrings that are identical except for their varying case, use extra spaces between the first pair of colons to distinguish one from the other. This is necessary because hotstrings are labels, and each label must be unique. In the below example, the first hotstring is made distinct from the second by including an extra space between its first pair of colons:

```
:c :ceo::chief executive officer  
:c:CEO::Chief Executive Officer
```

C1: Do not conform to typed case. Use this option to make auto-replace hotstrings case insensitive and prevent them from conforming to the case typed by the user. Use **C0** to make hotstrings conform again.

Hotstrings that conform to typed case will produce the replacement text in all caps if the user typed the abbreviation in all caps. If the user typed only the first letter in caps, the first letter of the replacement will also be capitalized (if it is a letter). If the user typed the case in any other way, the replacement is sent exactly as defined.

Kn: Key-delay: A delay of length 0 is normally done between keystrokes for all hotstrings that use auto-backspacing or auto-replacement. Zero is recommended for most purposes since it is both fast and able to cooperate well with other processes (due to internally doing a [Sleep 0](#)). If a different delay is desired, specify for **n** the new delay. Use -1 to have no delay at all, which is useful to make auto-replacements faster if your CPU is frequently under heavy load from background processes. *Only considered for the automatic backspacing. For the speed of the actual typing, use SetKeyDelay instead.*

O: Omit the ending character of auto-replace hotstrings when the replacement is produced. This is useful when you want a hotstring to be kept unambiguous by still requiring an ending character, but don't actually want the ending character to be shown on the screen. For example, if `:o:ar::aristocrat` is a hotstring, typing "ar" followed by a space will produce "aristocrat" with no trailing space, which allows you to make the word plural or possessive without having to backspace. Use **O0** (the letter O followed by a zero) to turn this option back off.

P: The priority of the hotstring. *This option has no effect on auto-replace hotstrings. Yes it does*

R: Send the replacement text raw, that is, exactly as it appears rather than translating {Enter} to an ENTER keystroke, ^c to Control-C, etc. Use **R0** to turn this option back off.

*****: An ending character (e.g. space, period, or enter) is not required to trigger the hotstring. Use ***0** to turn this option back off. In the example below, the replacement occurs the moment the @ character is typed:
`:@:js@::jsmith@somedomain.com`

?: The hotstring will be triggered even when it is inside another word, that is, when the character typed immediately before it is alphanumeric. For example, if `?:al::airline` is a hotstring, typing "practical" would produce "practicairline". Use **?0** to turn this option back off.

Remarks

Variable references such as %MyVar% are not currently supported within the replacement text. To work around this, don't make such hotstrings auto-replace; instead use the [Send](#) command in the body of the hotstring's subroutine. *Yes they are*

The built-in variable **A_EndChar** contains the ending character that was pressed by the user to trigger the most recent non-auto-replace hotstring. If no ending character was required (due to the * option), it will be blank. This variable is useful when making hotstrings that use the [Send](#) command or whose behavior should vary depending on which ending character was typed by the user. To send the ending character itself, use "SendRaw %A_EndChar%" (SendRaw is used because characters such as !{} would not be sent correctly by the normal [Send](#) command).

Although commas, percent signs and single-colons within hotstring definitions [do not](#) need to be [escaped](#), accents and those semicolons having a space or tab to their left require it. See [escape sequences](#) for a complete list.

Although the [Send command's syntax](#) is supported in auto-replacement text (unless the raw option is used), the hotstring abbreviations themselves do not use this. Instead, specify `n for the ENTER key and `t (or a literal tab) for TAB (see [escape sequences](#) for a complete list). In the following example, the hotstring would be triggered when the user types "ab" followed by a tab: ::ab`t::

Spaces and tabs are treated literally within hotstring definitions. For example, the following would produce two different results:

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

Each hotstring abbreviation can be no more than 30 characters long. The program will warn you if this length is exceeded. By contrast, the length of hotstring's replacement text is limited only by AutoHotkey's maximum line length, which is 16,383 characters.

The order in which hotstrings are defined determines their precedence with respect to each other. In other words, if more than one hotstring matches something typed by the user, only the first one will take effect.

Any backspacing done by the user is taken into account for the purpose of detecting hotstrings. However, the use of arrow keys, PageUp, PageDown, Home, and End to navigate within an editor will cause the hotstring recognition process to reset. In other words, it will begin waiting for an entirely new hotstring.

A hotstring may be typed even when the active window is ignoring the user's keystrokes. In other words, the hotstring will still fire even though the triggering abbreviation is never visible. In addition, the backspace key will undo the most recently typed keystroke even though you can't see the effect.

It is possible to [Gosub](#) or [Goto](#) a hotstring label by including its first pair of colons (including any option symbols) in front of its name. However, if there are no options between the colons, you must escape the colon pair as in this example: `Gosub `::btw`. Although hotstrings are ~~not monitored and will not be triggered during the course of an invisible~~ [Input](#) command, visible Inputs are capable of triggering them.

Hotstrings can never be triggered by keystrokes produced by any AutoHotkey script. This avoids the possibility of an infinite loop wherein hotstrings trigger each other over and over.

The [Input](#) command is more flexible than hotstrings for certain purposes. For example, it allows your keystrokes to be invisible from the active window (such as a game). It also supports non-character ending keys such as Escape.

The [keyboard hook](#) is automatically used by any script that contains hotstrings.

Hotstrings behave identically to hotkeys in the following ways:

- They are affected by the [Suspend](#) command.
 - They obey [#MaxThreads](#) and [#MaxThreadsPerHotkey](#) (but not [#MaxThreadsBuffer](#)).
 - Scripts containing hotstrings are automatically [persistent](#).
 - ~~Non auto-replace~~ hotstrings will create a new [thread](#) when launched. In addition, they will update the built-in hotkey variables such as [A_ThisHotkey](#).
-

#

List of Keys and Mouse/Joystick Buttons

Mouse

LButton - the left mouse button

RButton - the right mouse button

MButton - the middle or wheel mouse button

WheelDown - this is equivalent to rotating the mouse wheel down (toward you)

WheelUp - the opposite of the above

XButton1 - a button that appears only on certain mice

XButton2 - a button that appears only on certain mice

Joystick (requires v1.0.10+)

~~**Joy1 through Joy32:** The buttons of the joystick. To help determine the button numbers for a particular joystick, use this test script. Note that hotkey prefix symbols such as ^ (control) and + (shift) are not supported. Also note that the pressing of joystick buttons always "passes through" to the active window if that window is designed to detect the pressing of joystick buttons.~~

~~Although the following Joystick control names cannot be used as hotkeys, they can be used with GetKeyState:~~

~~**JoyX, JoyY, and JoyZ:** The X (horizontal), Y (vertical), and Z (altitude/depth) axes of the joystick.~~

~~**JoyR:** The rudder or 4th axis of the joystick.~~

~~**JoyU and JoyV:** The 5th and 6th axes of the joystick.~~

~~**JoyPOV:** The point of view (hat) control.~~

~~**JoyName:** The name of the joystick or its driver.~~

~~**JoyButtons:** The number of buttons supported by the joystick (not always accurate).~~

~~**JoyAxes:** The number of axes supported by the joystick.~~

~~**JoyInfo:** Provides a string consisting of zero or more of the following letters to indicate the joystick's capabilities: **Z** (has Z axis), **R** (has R axis), **U** (has U axis), **V** (has V axis), **P** (has POV control), **D** (the POV control has a limited number of discrete/distinct settings), **C** (the POV control is continuous/fine). Example string: ZRUVPD~~

~~**Multiple Joysticks:** If the computer has more than one and you want to use one beyond the first, include the joystick number in front of the control name. For example, 2joy1 is the second joystick's first button.~~

~~**Using Joystick as Mouse:** This script converts a joystick into a two button mouse.~~

Keyboard

Note: The names of the letter and number keys are the same as that single letter or digit. For example: b is the "b" key and 5 is the "5" key.

Space - the spacebar
Tab
Enter (or Return)
Escape (or Esc)
Backspace (or BS)

Delete (or Del)
Insert (or Ins)
Home
End
PgUp
PgDn
Up
Down
Left
Right

ScrollLock
CapsLock
NumLock

NumpadDiv - the slash key
NumpadMult - the asterisk key
NumpadAdd - the plus key
NumpadSub - the minus key
NumpadEnter - the Enter key

The following keys are used when Numlock is OFF:

NumpadDel
NumpadIns
NumpadClear - same physical key as Numpad5 on most keyboards
NumpadUp
NumpadDown
NumpadLeft
NumpadRight
NumpadHome
NumpadEnd
NumpadPgUp
NumpadPgDn

The following keys are used when Numlock is ON:
Numpad0

Numpad1
Numpad2
Numpad3
Numpad4
Numpad5
Numpad6
Numpad7
Numpad8
Numpad9
NumpadDot - the decimal point (period) key

F1 through F24 - The 12 or more function keys at the top of most keyboards.

AppsKey - this is the key that invokes the right-click context menu.

LWin - the left windows key
RWin - the right windows key
Control (or Ctrl)
Alt
Shift

Note: For the most part, these next 6 keys are **not** supported by Windows 95/98/Me. Use the above instead:

LControl (or LCtrl) - the left control key
RControl (or RCtrl) - the right control key
LShift
RShift
LAlt

RAlt -- Note: If your keyboard layout has AltGr instead of RAlt, you can probably use it as a hotkey prefix via <^>! as described [here](#). In addition, "LControl & RAlt:;" would make AltGr itself into a hotkey.

PrintScreen
CtrlBreak
Pause
Break

Help - this probably doesn't exist on most keyboards. It's usually not the same as F1.

Sleep - note that the sleep key on some keyboards might not work with this.

The following exist only on Multimedia or Internet keyboards that have extra buttons or keys:

Browser_Back
Browser_Forward
Browser_Refresh
Browser_Stop
Browser_Search

Browser_Favorites
Browser_Home
Volume_Mute
Volume_Down
Volume_Up
Media_Next
Media_Prev
Media_Stop
Media_Play_Pause
Launch_Mail
Launch_Media
Launch_App1
Launch_App2

Special Keys

If your keyboard has a key not listed above, you might still be able to make it a hotkey by using the following steps (this technique requires v1.0.08+ and is currently not supported on Windows 95/98/Me):

1. Ensure that at least one script is running that has the keyboard hook installed.
2. Double-click that script's tray icon to open its main window.
3. Press one of the "mystery keys" on your keyboard.
4. Select the menu item "View->Key history".
5. Scroll down to the bottom of the page. Somewhere near the bottom of the history list are the key down and key up events for your key. NOTE: Some keys do not generate events and thus will not be visible here. If this is the case, you cannot make that particular key a hotkey because your keyboard driver or hardware handles it at a level too low for AutoHotkey to access. In that case, you can try reconfiguring or removing any extra software that came with your keyboard or changing the keyboard driver to a more standard one such as the one built into the OS.
6. If your key does generate events, make a note of the 3-digit value in the second column of the list (e.g. 159).
7. To define this key as a hotkey, follow this example:
SC159:: ; Replace 159 with your key's value.
MsgBox, %A_Hotkey% was pressed.
return

As an alternative or addition to the above: To remap some other key to become a "mystery key", follow this example (requires v1.0.14+): #c::Send {vkFFsc159} ; Replace 159 with the value discovered above. Replace FF (if needed) with the key's virtual key, which can be discovered in the first column of "View->Key history".

#

How Scripts Are Run -- The Auto-execute Section

Scripts are [optimized](#) and validated when launched. Any syntax errors will be displayed, and they must be corrected before the script can run.

The program loads a script into memory line by line, and each line may be up to 16,383 characters long. After this, it runs the script from the top, continuing until it encounters a [Return](#), [Exit](#), [hotkey/hotstring](#) label, or the physical end of the script (whichever comes first). This top portion of the script is referred to as the *auto-execute* section.

A script that is not [persistent](#) and that lacks [hotkeys](#) and [hotstrings](#) will terminate after the *auto-execute* section has completed. Otherwise, it will stay running in an idle state, responding to events such as hotkeys, hotstrings, [GUI events](#), [custom menu items](#), and [timers](#).

Every subroutine launched by a [hotkey](#), [hotstring](#), [custom menu item](#), [GUI event](#), or [timer](#) starts off fresh with the default values for the following attributes as set in the *auto-execute* section. If unset, the standard defaults will apply (as documented on each of the following pages):

[DetectHiddenWindows](#) [DetectHiddenText](#)
[SetTitleMatchMode](#)
[SetBatchLines](#)
[SetKeyDelay](#)
[SetMouseDelay](#)
[SetWinDelay](#)
[SetControlDelay](#)
[SetDefaultMouseSpeed](#)
[CoordMode](#)
[SetStoreCapslockMode](#)
[AutoTrim](#)
[SetFormat](#)
[StringCaseSense](#)

NOTE: If the *auto-execute* section takes a long time to complete (or never completes), the default values for the above settings will be put into effect after 100 milliseconds. Thus, it's usually best to make any desired changes to the defaults at the top of scripts that contain [hotkeys](#), [hotstrings](#), [timers](#), or [custom menu items](#). Also note that each [thread](#) retains its own collection of the above settings. Changes made to those settings will not affect other [threads](#).

Escape Sequences

AutoHotkey's default [escape character](#) is accent ` (upper left corner of most English keyboards). This avoids the need for backslashes in filenames to be escaped. For example, in a .ahk script a filename can be written as /a/b\c/d rather than needing double-backslashes.

Since commas and percent signs have special meaning in the AutoHotkey language, use `, to specify a literal comma and `% to specify an actual percent sign. One of the exceptions to this is [MsgBox](#), which does not require commas to be escaped. See [#EscapeChar](#) for a complete list of escape sequences.

Comments in Scripts

Scripts can be commented by using a semicolon at the beginning of a line. Comments may also be added to the end of a command, in which case the semicolon must have at least one space or tab to its left. For example:

```
; This line is a full line comment.  
WinActivate, Untitled - Notepad ; This is a comment on the same line as a command.
```

In addition, the /* and */ symbols can be used to comment out an entire section, but only if the symbols appear at the beginning of a line as in this example:

```
/*  
MsgBox, This line is commented out (disabled).  
MessageBox, This one too.  
*/
```

Tip: The first comma of any command may be omitted. Example:

```
MsgBox This is ok.  
MessageBox, This is ok too.
```

Convert a Script to an EXE executable (ahk2exe)

A script compiler is included with the program.

Once a script is compiled, it becomes a standalone executable, meaning that it can be used even on machines where AutoHotkey isn't present (and such EXE executables can be distributed or sold with no restrictions). The compilation process compresses and encrypts all of the following: the script, any files it includes, and any files it has incorporated via the FileInstall command.

Ahk2Exe can be used in the following ways:

1. **GUI Interface:** Run the "Convert .ahk to .exe" item in the Start Menu.
2. **Right-click:** Within an open Explorer window, you can right-click any .ahk file and select "Compile Script" "Compile with ahk_x11" (only available if the script compiler option was chosen when AutoHotkey was installed). This will create an EXE executable (same filename, but without the .ahk) file of the same base filename as the script, which will appear after a short time in the same directory. Note: The EXE file will be produced using the same custom icon and compression level that was last used by Method #1 above and it will not have a password. Portable binaries have no icons on Linux.
3. **Command Line:** The compiler can be run from the command line with the following parameters:
`Ahk2exe.exe /in MyScript.ahk [/out MyScript.exe] [/icon MyIcon.ico] [/pass password] ahk_x11 --compile MyScript.ahk [MyBinary]`
Parameters containing spaces should be enclosed in double quotes. If the "out" file is omitted, the EXE will have the same base filename as the script itself.

Important Notes:

- A password should be specified if you plan to distribute your EXE and don't want anyone to be able to view the source code of your script.
- The commands `#NoTrayIcon` and "`Menu, Tray, ShowMainWindow`" affect the behavior of compiled scripts.

- An EXE can be "decompiled" to retrieve the original script by downloading [Exe2Ahk](#) (this utility should be run from the command prompt). However, any comments (semicolon'd lines) originally present will be lost.
- Custom version info (as seen in Explorer's file properties dialog) can be added to your compiled scripts by using a utility such as [Resource Hacker](#) (freeware) to edit the file "AutoHotkeySC.bin". This file is contained in the compiler subfolder where AutoHotkey was installed. Note: ResHacker will corrupt compiled scripts, which is why only the AutoHotkeySC.bin file should be edited.

Passing Command Line Parameters to a Script

Scripts support command line parameters. The format is:
ahk_x11 [Switches] [Script Filename] [Script Parameters]
CompiledScript [Switches] [Script Parameters]

~~Switches~~ can be zero or more of the following:

~~/f~~ or ~~/force~~ — Launch unconditionally, skipping any warning dialogs.

~~/r~~ or ~~/restart~~ — Indicate that the script is being restarted (this is also used by the [Reload](#) command, internally).

~~/ErrorStdOut~~ — Send syntax errors to stdout rather than displaying a dialog. See [#ErrorStdOut](#) for details.

Script Filename can be omitted if there are no *Script Parameters*. If omitted, it will run (or prompt you to create) [AutoHotkey.ini](#) in the current working directory. [the installer](#).

Script Parameters can be any strings you want to pass into the script (but any string that contains spaces must be enclosed in double quotes). The script sees incoming parameters as the variables %1%, %2%, and so on. In addition, %0% contains the number of parameters passed (0 if none).

If the number of parameters passed into a script varies (perhaps due to the user dragging and dropping a set of files onto a compiled script), the following example can be used to extract them one by one:

```
Loop, %0%
{
    StringTrimRight, param, %A_Index%, 0
    MsgBox, 4,, Parameter #%A_Index% is %param%. Continue?
    IfMsgBox, No
        break
}
```

Script Showcase

See [this page](#) for some useful scripts.

#

Introduction

Variables are areas of memory set aside to hold *values*. A *value* (or *string* as it is sometimes called) can be any series of characters or digits. For example, the following line assigns a value to the variable named MyVar:

```
MyVar = 123abc
```

To later retrieve the contents of the variable, make a *reference* to it by enclosing its name in percent signs, as in this example:

```
MsgBox, The value in the variable named MyVar is %MyVar%
```

However, the parameters of some commands are explicitly defined as input or output variables. In these cases, do **not** enclose the variable in percent signs. For example, neither of the following variables should have percent signs around them: `StringLen`, `OutputVar`, `InputVar`

Variable **names** may consist of numbers, letters, and the following punctuation: # _ @ \$? []

(the square brackets might typically be used for [arrays](#), though this is optional). In addition, variable names may start with a number, or even be entirely numeric.

Notes about variable capacity and memory usage:

- ~~Each variable may contain up to 64 MB of text (this limit can be increased with `#MaxMem`).~~
- When a variable is given a new string longer than its current contents, additional system memory is allocated automatically.
- The memory occupied by a large variable can be freed by setting it equal to nothing, e.g. `var =`

Environment Variables vs. "Normal" Variables

~~AutoHotkey does not store its variables as environment variables. This is because performance would be worse and also because the operating system limits such variables to 32 KBytes. Use EnvSet (not to be confused with SetEnv) to explicitly place a value into an environment variable.~~

~~Any reference to an undefined or blank variable (e.g. %EmptyVar%) resolves to an empty string unless that variable is defined in the environment (e.g. %ProgramFiles%), in which case the environment's value will be retrieved and used.~~

Naming Variables to Improve Maintainability

Since all variables are global, in a large or complex script it is usually best to use unique names for variables to reduce the possibility of bugs and unwanted side-effects. For example, to call a subroutine in a way that mimics passing parameters and receiving a return value, a convention similar to the following might be useful:

```
; Set input "parameters" by using a prefix/acronym to indicate who they're for.  
; "gce" stands for "GetColorExists":  
gce_x1 = 30  
gce_x2 = 50  
gce_y1 = 0
```

```
gce_y2 = 200
Gosub, GetColorExists
MsgBox The subroutine's output is %ColorExists%.
return

GetColorExists:
...
; Set the output for the caller using the variable name indicated by the subroutine's name:
ColorExists = y
...
return
```

Built-in Variables

The following built-in variables can also be used. Most of them are reserved, meaning that their contents cannot be directly altered by the script:

Table of Contents

- Special Characters: [A_Space](#), [A_Tab](#)
- Script Properties: [command line parameters](#), [A_WorkingDir](#), [A_ScriptDir](#), [A_ScriptName](#), (...more...)
- Date and Time: [A_YYYY](#), [A_MM](#), [A_DD](#), [A_Hour](#), [A_Min](#), [A_Sec](#), (...more...)
- Script Settings: [A_BatchLines](#), [A_TitleMatchMode](#), (...more...)
- User Idle Time: [A_TimeIdle](#), [A_TimeIdlePhysical](#)
- GUI Windows and Menu Bars: [A_Gui](#), [A_GuiControl](#), [A_GuiControlEvents](#)
- Hotkeys, Hotstrings, and Custom Menu Items: [A_ThisHotkey](#), [A_EndChar](#), [A_ThisMenuItem](#), (...more...)
- Operating System and User Info: [A_OsVersion](#), [A_ScreenWidth](#), [A_ScreenHeight](#), (...more...)
- Misc: [A_Cursor](#), [A_CaretX](#), [A_CaretY](#), [Clipboard](#), [ErrorLevel](#)
- Loop: [A_Index](#), (...more...)

Special Characters

A_Space	# This variable contains a single space character. See AutoTrim for details.
A_Tab	# This variable contains a single tab character. See AutoTrim for details.

Script Properties

1, 2, 3, etc.	These variables are automatically created whenever a script is launched with command line parameters. They can be changed and referenced just like normal variable names (for example: %1%). The variable %0% contains the number of parameters passed (0 if none). See the script section for details.
A_WorkingDir	# The script's current working directory, which is where files will be accessed by default. The final backslash is not included unless it is the root directory. Two examples: / and /home. Use SetWorkingDir to change the working directory.
A_ScriptDir	# The full path of the directory where the current script is located. The final backslash is not included.
A_ScriptName	# The file name of the current script, without its path, e.g. MyScript.ahk. If the script is compiled , the name will end in .exe <i>usually have no file extension</i> instead. Thus, a good questionable way for a script to detect whether it is running in compiled form is with the following: <code>SplitPath, A_ScriptName,, FileExt if FileExt = ahk MsgBox This script is NOT running in compiled form.</code>
A_ScriptFullPath	# The combination of the above two variables to give the complete file specification of the script, e.g. /a/b/MyScript.ahk
A_LineNumber	#The number of the currently executing line within the script (or one of its #Include files). This line number will match the one shown by ListLines ; it can be useful for error reporting such as this example: MsgBox Could not write to log file (line number %A_LineNumber%).
A_ThisLabel	#The name of the label (subroutine) that is currently executing (blank if none); for example: MyLabel. It is updated whenever the script executes Gosub/Return or Goto. It is also updated for automatically-called labels such as timers, GUI threads, menu items, hotkeys, hotstrings, OnClipboardChange labels, and OnExit labels. However, A_ThisLabel is not updated when execution "falls into" a label from above; when that happens, A_ThisLabel retains its previous value. See also: A_ThisHotkey and IsLabel()
A_AhkVersion	# In versions prior to 1.0.22, this variable is blank. Otherwise, it contains the version of AutoHotkey that is running the script, such as 1.0.22. In the case of a compiled script, the version that was originally used to compile it is reported. Always contains 1.0.24.
A_IsCompiled	#Contains 1 if the script is running as a compiled EXE and an empty string (which is considered false) if it is not.
A_ExitReason	# The most recent reason the script was asked to terminate. This variable is blank unless the script has an OnExit subroutine and that subroutine is currently running or has been called at least once by an exit attempt. See OnExit for details.

Date and Time

A_YYYY	Current 4-digit year (e.g. 2004). Synonymous with A_Year. Note: To retrieve a formatted time or date appropriate for your locale and language, use " FormatTime , OutputVar" (time and long date) or " FormatTime , OutputVar,, LongDate" (retrieves long-format date).
--------	---

A_MM	Current 2-digit month (01-12). Synonymous with A_Mon.
A_DD	Current 2-digit day of the month (01-31). Synonymous with A_MDay.
A_MMM M	# Current month's full name in the current user's language, e.g. July
A MMM	# Current month's abbreviation in the current user's language, e.g. Jul
A_DDDD	# Current day of the week's full name in the the current user's language, e.g. Sunday
A_DDD	# Current day of the week's 3-letter abbreviation in the the current user's language, e.g. Sun
A_WDay	# Current 1-digit day of the week (1-7). 1 is Sunday in all locales.
A_YDay	# Current day of the year (1-366). The value is not zero-padded, e.g. 9 is retrieved, not 009. To retrieve a zero-padded value, use the following: FormatTime , OutputVar, , YDay0
A_YWee k	# Current year and week number according to ISO 8601. Example: 200453. To separate the year from the week, use StringLeft and StringRight . Definition of week number: If the week containing January 1st has four or more days in the new year, it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1. [requires v1.0.24+]
A_Hour	# Current 2-digit hour (00-23) in 24-hour time (for example, 17 is 5pm). To retrieve 12-hour time as well as an AM/PM indicator, follow this example: FormatTime , OutputVar, , h:mm:ss tt
A_Min	# Current 2-digit minute (00-59)
A_Sec	# Current 2-digit second (00-59)
A_Now	# The current local time in YYYYMMDDHH24MISS format.
A_NowU TC	# The current Coordinated Universal Time (UTC) in YYYYMMDDHH24MISS format. UTC is essentially the same as Greenwich Mean Time (GMT).
A_TickC ount	# The number of milliseconds that have elapsed since the computer was rebooted. By storing %A_TickCount% in a variable, elapsed time can later be measured by subtracting that variable from the current %A_TickCount% value, as in this example: start_time = %A_TickCount% Sleep, 1000 elapsed_time = %A_TickCount% elapsed_time -= %start_time% MsgBox, %elapsed_time% milliseconds have elapsed.

Script Settings

A_IsSuspended

Contains 1 if the script is [suspended](#) and 0 otherwise.

A_BatchLines	# (synonymous with A_NumBatchLines) The current value as set by SetBatchLines. Examples: 200 or 10ms (depending on format).
A_TitleMatchMode	# The current mode (1, 2, or 3) set by SetTitleMatchMode.
A_TitleMatchModeSpeed	# The current match speed (fast or slow) set by SetTitleMatchMode.
A_DetectHiddenWindows	# The current mode (On or Off) set by DetectHiddenWindows.
A_DetectHiddenText	# The current mode (On or Off) set by DetectHiddenText.
A_AutoTrim	# The current mode (On or Off) set by AutoTrim.
A_StringCaseSense	# The current mode (On or Off) set by StringCaseSense.
A_FormatInteger	# The current integer format (H or D) set by SetFormat.
A_FormatFloat	# The current floating point number format set by SetFormat.
A_KeyDelay	# The current delay set by SetKeyDelay (always decimal, not hex).
A_WinDelay	# The current delay set by SetWinDelay (always decimal, not hex).
A_ControlDelay	# The current delay set by SetControlDelay (always decimal, not hex).
A_MouseDelay	# The current delay set by SetMouseDelay (always decimal, not hex).
A_DefaultMouseSpeed	# The current speed set by SetDefaultMouseSpeed (always decimal, not hex).
A_IconHidden	# Contains 1 if the tray icon is currently hidden or 0 otherwise. The icon can be hidden via #NoTrayIcon or the Menu command.
A_IconTip	# Blank unless a custom tooltip for the tray icon has been specified via Menu, Tray, Tip -- in which case it's the text of the tip.
A_IconFile	# Blank unless a custom tray icon has been specified via Menu, tray, icon -- in which case it's the full path and name of the icon's file.
A_IconNumber	# Blank if A_IconFile is blank. Otherwise, it's the number of the icon in A_IconFile (typically 1).

User Idle Time

A_TimeIdle	# The number of milliseconds that have elapsed since the system last received keyboard, mouse, or other input. This is useful for determining whether the user is away. This variable will be blank unless the operating system is Windows 2000, XP, or beyond. Physical input from the user as well as artificial input generated by any program or script (such as the Send or MouseMove commands) will reset this value back to zero. Since this value tends to increase by increments of 10, do not check whether it is equal to another value. Instead, check whether it is greater or less than another value. For example: IfGreater, A_TimeIdle, 600000, MsgBox, The last keyboard or mouse activity was at least 10 minutes ago.
A_TimeIdlePhysical	# Same as above but ignores artificial input whenever the corresponding hook (keyboard or mouse) is installed. If neither hook is installed, this variable is equivalent to A_TimeIdle. If only one hook is present, only that one type of artificial input will be ignored. A_TimeIdlePhysical may be more useful than A_TimeIdle for determining whether the user is truly present.

GUI Windows and Menu Bars

A_Gui	# The GUI window number that launched the current thread . This variable is blank unless a Gui control, menu bar item, or event such as GuiClose/GuiEscape launched the current thread.
A_GuiControl	# The name of the variable associated with the GUI control that launched the current thread . If that control lacks an associated variable , A_GuiControl instead contains the first 63 characters of the control's text/caption (this is most often used to avoid giving each button a variable name). A_GuiControl is blank whenever: 1) A_Gui is blank; 2) a GUI menu bar item or event such as GuiClose/GuiEscape launched the current thread; 3) the control lacks an associated variable and has no caption; or 4) The control that originally launched the current thread no longer exists (perhaps due to Gui Destroy).
A_GuiWidth A_GuiHeight	# These variables contain undefined/random values except when referenced in a GuiSize thread. See GuiSize label for details.
A_GuiControlEvent	<p>The type of event that launched the current thread. If the thread was not launched via GUI action, this variable is blank. Otherwise, it contains one of the following strings:</p> <p>Normal: The event was triggered by a single left click or via keystrokes (arrow keys, TAB key, space bar, underlined shortcut key, etc.). This value is also used for menu bar items and the special events such as GuiClose and GuiEscape.</p> <p>DoubleClick: The event was triggered by a double click. Note: The first click of the click pair will still cause a <i>Normal</i> event to be received first. In other words, the subroutine will be launched twice: once for the first click and again for the second.</p> <p>RightClick: Although this feature is not yet implemented, for compatibility with future versions of AutoIt hotkey, you should write your scripts under the assumption that it is possible. For example, do not assume that "if A_GuiControlEvent => Normal" must be a double click. Instead, it is usually best to have the script do nothing if the value is anything other than <i>Normal</i> or <i>DoubleClick</i>.</p> <p>Future/undetermined values: Same comment as above.</p> <p>Slider values: See the slider control for details.</p> <p>Drag and drop contents: See the GuiDropFiles label.</p> <p>Note: Unlike variables such as A_ThisHotkey, each thread retains its own value for A_Gui, A_GuiControl, and A_GuiControlEvent. Therefore, if a thread is interrupted by another, upon being resumed it will still see its original/correct values in these variables.</p>

Hotkeys, Hotstrings, and Custom Menu Items

A_ThisMenuItem	#The name of the most recently selected custom menu item (blank if none).
A_ThisMenu	# The name of the menu from which A_ThisMenuItem was selected.
A_ThisMenuItemPos	A number indicating the current position of A_ThisMenuItem within A_ThisMenu. The first item in the menu is 1, the second is 2, and so on. Menu separator lines are counted. This variable is blank if A_ThisMenuItem is blank or no longer exists within A_ThisMenu. It is also blank if A_ThisMenu itself no longer exists. [requires v1.0.18+]
A_ThisHotkey	#The key name of the most recently executed hotkey (blank if none), e.g. #z. This value will change if the current thread is interrupted by another hotkey, so be sure to copy it into another variable immediately if you need the original value for later use in a subroutine.

A_PriorHotkey	# Same as above except for the previous hotkey. It will be blank if none.
A_TimeSinceThisHotkey	# The number of milliseconds that have elapsed since A_ThisHotkey was pressed. It will be -1 whenever A_ThisHotkey is blank.
A_TimeSincePriorHotkey	# The number of milliseconds that have elapsed since A_PriorHotkey was pressed. It will be -1 whenever A_PriorHotkey is blank.
A_EndChar	# The ending character that was pressed by the user to trigger the most recent non-auto-replace hotstring . If no ending character was required (due to the * option), this variable will be blank. [requires v1.0.17+]

Operating System and User Info

A_OSType	# The type of Operating System being run. Either WIN32_WINDOWS (i.e. Win95/98/ME) or WIN32_NT (i.e. WinNT, Win2k, WinXP, and maybe some beyond those). <i>Currently always set to Linux.</i>
A_OVersion	# One of the following strings: WIN_XP, WIN_2000, WIN_NT4, WIN_95, WIN_98, WIN_ME.
A_Language	# The system's default language, which is one of the strings from the language table . <i>Note: To get the ISO language code instead, run `EnvGet, var, LANG` instead.</i>
A_ComputerName	# The network name of the computer.
A_UserName	# The logon name of the current user.
A_WinDir	# The windows directory (requires v1.0.23+). Example: C:\Windows
A_ProgramFiles	# The Program Files directory (requires v1.0.23+). Example: C:\Program Files
A_Desktop	The full path and name of the folder containing the current user's desktop files.
A_DesktopCommon	The full path and name of the folder containing the all users desktop files. [requires 1.0.24+]
A_StartMenu	# The full path and name of the current user's Start Menu folder. [requires 1.0.24+]
A_StartMenuCommon	# The full path and name of the all users Start Menu folder. [requires 1.0.24+]
A_Programs	# The full path and name of the Programs folder in the current user's Start Menu. [requires 1.0.24+]
A_ProgramsCommon	# The full path and name of the Programs folder in the all users Start Menu. [requires 1.0.24+]
A_Startup	# The full path and name of the Startup folder in the current user's Start Menu.
A_StartupCommon	# The full path and name of the Startup folder in the all users Start Menu. [requires 1.0.24+]
A_MyDocuments	# The full path and name of the current user's "My Documents" folder. Unlike most of the similar variables, if the folder is the root of a drive, the final backslash is not included. For example, it would contain M: rather than M:\
A_Home	<i># The full path and name of the current user's \$HOME folder.</i>
A_IsAdmin	# If the current user has admin rights, this variable contains 1. Otherwise, it contains 0.

A_ScreenWidth A_ScreenHeight	The width and height of the primary monitor, in pixels (e.g. 1024 and 768). To discover the dimensions of other monitors in a multi-monitor system, use SysGet . To instead discover the width and height of the entire desktop (even if it spans multiple monitors), use the following example SysGet , VirtualWidth, 78 SysGet , VirtualHeight, 79 In addition, use SysGet to discover the work area of a monitor, which can be smaller than the monitor's total area because the taskbar and other registered desktop toolbars are excluded.
A_IPAddress1 through 4	# The IP addresses of the first 4 network adapters in the computer. [requires v1.0.10+]

Misc.

A_Cursor	# The type of mouse cursor currently being displayed. It will be one of the following words: AppStarting, Arrow, Cross, Help, IBeam, Icon, No, Size, SizeAll, SizeNESW, SizeNS, SizeNWSE, SizeWE, UpArrow, Wait, Unknown. The acronyms used with the size-type cursors are compass directions, e.g. NESW = NorthEast+SouthWest. The hand-shaped cursors (pointing and grabbing) are classified as Unknown. [requires v1.0.10+]
A_Param1, A_Param2, ...etc	<i>The values of the parameters passed into the current invocation of a custom command (see #DefineCommand). Only available inside the associated label.</i>
Clipboard	The contents of the OS's clipboard, which can be read or written to. See the Clipboard section.
ErrorLevel	See ErrorLevel .

Loop

A_Index	# This is the number of the current loop iteration. For example, the first time the script executes the body of a loop, this variable will contain the number 1. See Loop for details.
A_LoopFileName, etc.	This and other related variables are valid only inside a file-loop .
A_LoopRegName, etc.	This and other related variables are valid only inside a registry loop .
A_LoopReadLine	See file-read loop .
A_LoopField	See parsing loop .

#

The "Last Found" Window

This is the window most recently found by [IfWin\[Not\]Exist](#), [IfWin\[Not\]Active](#), [WinWait\[Not\]Active](#), or [WinWait](#). It can make scripts easier to create and maintain since the WinTitle and WinText of the target window do not need to be repeated for every windowing command. In addition, scripts perform better because they don't need to search for the target window again after it's been found the first time.

The "last found" window can be used by all of the windowing commands except [WinWait](#), [WinActivateBottom](#), and [GroupAdd](#). To use it, simply omit all four window parameters (WinTitle, WinText, ExcludeTitle, and ExcludeText).

Each [thread](#) retains its own value of the "last found" window, meaning that if the [current thread](#) is interrupted by another, when the original thread is resumed it will still have its original value of the "last found" window, not that of the interrupting thread.

Related Notes:

- If multiple windows match the WinTitle/Text criteria of a windowing command such as [WinMove](#), the most recently active window is used.
- Almost all of the windowing commands can be told to operate upon the active window by specifying the letter A as the WinTitle parameter and omitting WinText, ExcludeTitle, and ExcludeText.
- In v1.0.09+, all windowing commands can operate upon a class of windows by specifying for WinTitle the class text shown by Window Spy or retrieved by [WinGetClass](#).
- In v1.0.12+, all windowing commands can operate upon a specific window via its unique ID number, which is retrieved by [WinGet](#).

Examples

```
IfWinExist, Untitled - Notepad
{
    WinActivate ; Automatically uses the window found above.
    WinMaximize ; same
    Send, Some text.{Enter}
    return
}

IfWinNotExist, Calculator
    return
else
{
    WinActivate ; The above "IfWinNotExist" also set the "last found" window for us.
    WinMove, 40, 40 ; Move it to a new position.
    return
}

; Make a hotkey to maximize a window.
; Specify just "A" to make it operate on the active window:
^Up::WinMaximize, A
```

#

AutoHotkey Command List -- Click on a command name for details.

Command	Description
{ ... }	Denotes a block. Blocks are typically used with Else , Loop , and IF-commands.
AutoTrim	Determines whether SetEnv and "var = value" statements remove spaces and tabs.
BlockInput	Disables or enables the user's ability to interact with the computer via keyboard, mouse, and perhaps other input devices.
Break	Exits (terminates) a loop . Valid only inside a loop .
ClipWait	Waits until the clipboard contains text or files.
Continue	Skips the rest of the current loop iteration and begins a new one. Valid only inside a loop .
Control	Makes a variety of changes to a control.
ControlClick	Sends a mouse button or mouse wheel event to a control.
ControlFocus	Sets input focus to a given control on a window.
ControlGet	Retrieves various types of information about a control.
ControlGetFocus	Retrieves which control of the target window has input focus, if any.
ControlGetPos	Retrieves the position and size of a control.
ControlGetText	Retrieves text from a control.
ControlMove	Moves or resizes a control.
ControlSend / ControlSendRaw	Sends simulated keystrokes to a window or control.
ControlSetText	Changes the text of a control.
CoordMode	Sets coordinate mode for various commands to be either relative or screen.
DetectHiddenText	Determines whether invisible text in a window is "seen" for the purpose of finding the window. This affects commands such as IfWinExist and WinActivate .
DetectHiddenWindows	Determines whether invisible windows are "seen" by the script.
Drive	Ejects/retracts the tray in a CD or DVD drive, or sets a drive's volume label.
DriveGet	Retrieves various types of information about the computer's drive(s).
DriveSpaceFree	Retrieves the free disk space of a drive, in Megabytes.
Echo	Prints to the console (standard out).

Edit	Opens the current script for editing in the associated editor.
Else	Specifies the command(s) to perform if an IF-command evaluates to FALSE.
EnvAdd	Sets a variable to the sum of itself plus the given value (can also add or subtract time from a date-time value). Synonymous with: var += value
EnvDiv	Sets a variable to itself divided by the given value. Synonymous with: var /= value
EnvMult	Sets a variable to itself times the given value. Synonymous with: var *= value
EnvSub	Sets a variable to itself minus the given value (can also compare date-time values). Synonymous with: var -= value
Eval	Takes a string argument and runs it as commands.
Exit	Exits the current thread or (if the script is not persistent contains no hotkeys) the entire script.
ExitApp	Terminates the script unconditionally.
DefineCommand	Registers a new command.
FileAppend	Appends text to a text file.
FileCopy	Copies one or more files.
FileCopyDir	Copies a directory and all sub-directories and files (similar to xcopy).
FileCreateDir	Creates a directory/folder.
FileCreateShortcut	Creates a shortcut (.lnk) file.
FileDelete	Deletes one or more files.
FileInstall	Includes the specified file inside the compiled script (Ahk2Exe).
FileReadLine	Reads the specified line from a file and stores the text in a variable .
FileRead	Reads the specified line from a file and stores the text in a variable .
FileGetAttrib	Retrieves a code string representing a file or folder's attributes.
FileGetShortcut	Retrieves information about a shortcut (.lnk) file, such as its target file.
FileGetSize	Retrieves the size of a file in bytes, KB, or MB.
FileGetTime	Retrieves the datetime stamp of a file or folder.
FileGetVersion	Retrieves the version of a file.
FileMove	Moves or renames one or more files.
FileMoveDir	Moves a directory and all sub-directories and files. It can also rename a directory.
FileRecycle	Sends a file or directory to the recycle bin, if possible.
FileRecycleEmpty	Empties the recycle bin.

FileRemoveDir	Deletes a directory/folder.
FileSelectFile	Displays a standard dialog that allows the user to select file(s).
FileSelectFolder	Displays a standard dialog that allows the user to select a folder.
FileSetAttrib	Changes the attributes of one or more files or folders. Wildcards are supported.
FileSetTime	Changes the datetime stamp of one or more files or folders. Wildcards are supported.
FormatTime	Transforms a <code>YYYYMMDDHH24MISS</code> timestamp into the specified date/time format.
GetKeyState	Checks if a keyboard key or mouse/joystick button is down or up. Also retrieves joystick status.
Gosub	Jumps to the specified label and continues execution until Return is encountered.
Goto	Jumps to the specified label and continues execution.
GroupActivate	Activates the next window in a window group that was defined with GroupAdd .
GroupAdd	Adds a window specification to a window group, creating the group if necessary.
GroupClose	Closes the active window if it was just activated by GroupActivate or GroupDeactivate . It then activates the next window in the series. It can also close all windows in a group.
GroupDeactivate	Similar to GroupActivate except activates the next window not in the group.
Gui	Creates and manages windows and controls. Such windows can be used as data entry forms or custom user interfaces.
GuiControl	Makes a variety of changes to a control in a GUI window.
GuiControlGet	Retrieves various types of information about a control in a GUI window.
Hotkey	Creates, modifies, enables, or disables a hotkey while the script is running.
If	Compares a variable to a value.
If var [not] between	Checks whether a variable 's contents are numerically or alphabetically between two values (inclusive).
If var [not] in/contains MatchList	Checks whether a variable 's contents match one of the items in a list.
If var is [not] type	Checks whether a variable 's contents are numeric, uppercase, etc.
IfEqual/IfNotEqual	Compares a variable to a value for equality. Synonymous with: <code>if var = value ...</code> <code>if var <> value</code>
IfExist/IfNotExist	Checks for the existence of a file or directory.
IfGreater/IfGreaterOrEqual	Compares a variable to a value. Synonymous with: <code>if var > value if var >= value</code>
IfInString/IfNotInString	Checks if a variable contains the specified string.
IfLess/IfLessOrEqual	Compares a variable to a value. Synonymous with: <code>if var < value if var <= value</code>
IfMsgBox	Checks which button was pushed by the user during the most recent MsgBox command.

IfWinActive/IfWinNotActive	Checks to see if a specified window exists and is currently active (foreground).
IfWinExist/IfWinNotExist	Checks to see if a specified window exists.
IniDelete	Deletes a value from a standard format .ini file.
IniRead	Reads a value from a standard format .ini file.
IniWrite	Writes a value to a standard format .ini file.
Input	Waits for the user to type a string (not supported on Windows 9x).
InputBox	Displays an input box to ask the user to enter a string.
KeyHistory	Displays script info and a history of the 20 most recent keystrokes and mouse clicks.
KeyWait	Waits for a key or mouse/joystick button to be released or pressed down.
LeftClick	[Obsolete -- use MouseClick for greater flexibility]
LeftClickDrag	[Obsolete -- use MouseClickDrag for greater flexibility]
ListHotkeys	Displays the hotkeys in use by the current script, whether their subroutines are currently running, and whether or not they use the keyboard or mouse hook.
ListLines	Displays the script lines most recently executed.
ListVars	Displays the script's variables : their names and current contents.
EnvGet	Retrieves an environment variable.
EnvSet	Sets an environment variable for usage in Run/RunWait.
Loop (normal)	Perform a series of command repeatedly: either the specified number of times or until break is encountered.
Loop (files & folders)	Retrieves the specified files or folders, one at a time.
Loop (parse a string)	Retrieves substrings (fields) from a string, one by one.
Loop (read file contents)	Retrieves the lines in a text file, one by one (performs better than FileReadLine).
Loop (registry)	Retrieves the contents of the specified registry subkey, one item at a time.
Menu	Creates, deletes, modifies and displays menus and menu items. Changes the tray icon and its tooltip. Controls whether the main window of a compiled script can be opened.
Click	Clicks or holds a mouse button, or turns the mouse wheel.
MouseClick	Clicks or holds a mouse button, or turns the mouse wheel.
MouseClickDrag	Clicks and holds the specified mouse button, moves the mouse to the destination coordinates, then releases the button.
MouseGetPos	Retrieves the current position of the mouse cursor, and optionally which window and control it is hovering over.
MouseMove	Moves the mouse cursor.
MsgBox	Displays a simple message box with one or more buttons and with optional timeout.

OnExit	Specifies a subroutine to run when the script exits.
Pause	Pauses the script's current thread .
PixelGetColor	Retrieves the color of the pixel at the specified x,y screen coordinates.
PixelSearch	Searches a region of the screen for a pixel of the specified color(s).
Process	Performs one of the following operations on a process: checks if it exists; changes its priority; closes it; waits for it to close.
Progress	Creates or modifies a window to display a progress bar.
Random	Generates a pseudo-random number.
Reload	Replaces the currently running instance of the script with a new one.
Repeat...EndRepeat	[Obsolete -- use Loop for greater flexibility]
Return	Returns from a subroutine to which execution had previously jumped via Gosub , Hotkey activation, GroupActivate , or other means.
RightClick	[Obsolete -- use MouseClick for greater flexibility]
RightClickDrag	[Obsolete -- use MouseClickDrag for greater flexibility]
Run	Runs an external program.
RunAs	Specifies a set of user credentials to use for all subsequent uses of Run and RunWait .
RunWait	Runs an external program and waits until it finishes.
Send / SendRaw	Sends simulated keystrokes to the active window.
SetBatchLines	Determines how fast a script will run (affects CPU utilization).
SetCapslockState	Sets the state of the Capslock key. Can also force the key to stay on or off.
SetControlDelay	Sets the delay that will occur after each Control-modifying command.
SetDefaultMouseSpeed	Sets the mouse speed that will be used if unspecified in MouseMove/Click/Drag .
SetEnv	Assigns the specified value to a variable . Synonymous with: var = value
SetFormat	Sets the format of integers and floating point numbers generated by math operations.
SetKeyDelay	Sets the delay that will occur after each keystroke sent by Send or ControlSend .
SetMouseDelay	Sets the delay that will occur after each mouse movement or click.
SetNumlockState	Sets the state of the Numlock key. Can also force the key to stay on or off.
SetScrollLockState	Sets the state of the Scrolllock key. Can also force the key to stay on or off.
SetStoreCapslockMode	Whether to restore the state of CapsLock after a Send .
SetTimer	Causes a subroutine to be launched automatically and repeatedly at a specified time interval.

SetTitleMatchMode	Determines how window titles are searched for by various commands.
SetWinDelay	Sets the delay that will occur after each windowing command.
SetWorkingDir	Changes the script's current working directory.
Shutdown	Shuts down, restarts, or logs off the system.
Sleep	Waits the specified amount of time before continuing.
Sort	Arranges a variable's contents in alphabetical or numerical order.
SoundGet	Retrieves various settings from a sound device (master mute, master volume, etc.)
SoundGetWaveVolume	Retrieves the wave output volume from a sound device.
SoundPlay	Play a sound, video, or other supported file type.
SoundSet	Changes various settings of a sound device (master mute, master volume, etc.)
SoundSetWaveVolume	Changes the wave output volume for a sound device.
SplashImage	Creates or modifies a window to display a JPG, GIF, or BMP image.
SplashTextOn	Creates a customizable text popup window.
SplashTextOff	Closes the above window.
SplitPath	Separates a file name into its name, directory, extension, and drive.
StatusBarGetText	Retrieves the text from a standard status bar control.
StatusBarWait	Waits until a window's status bar contains the specified string.
StringCaseSense	Determines whether string comparisons are case sensitive (default = no).
StringGetPos	Retrieves the position of the specified substring within a string.
RegExStringGetPos	<i>Retrieves the position of the specified substring within a string.</i>
StringLeft	Retrieves a number of characters from the left hand side of a string.
StringLen	Retrieves the count of how many characters are in a string.
StringLower	Converts a string to lowercase.
StringMid	Retrieves a number of characters from the specified position in a string.
StringReplace	Replaces the specified substring with a new string.
RegExReplace	<i>Replaces the specified regular expression substring with a new string.</i>
StringRight	Retrieves a number of characters from the right hand side of a string.
StringSplit	Separates a string into an array of substrings using the specified delimiters.

StringTrimLeft	Removes a number of characters from the left hand side of a string.
StringTrimRight	Removes a number of characters from the right hand side of a string.
StringUpper	Converts a string to uppercase.
Suspend	Disables all or selected hotkeys .
SysGet	Retrieves screen resolution, multi-monitor info, dimensions of system objects, and other system properties.
Thread	Sets the priority or interruptibility threads .
ToolTip	Creates an always-on-top window anywhere on the screen.
Transform	Performs miscellaneous math functions and tasks such as ASCII/Unicode conversion and bitwise operations.
TrayTip	Creates a balloon message window near the tray icon (requires Windows 2000/XP+).
URLDownloadToFile	Downloads a file from the Internet.
WinActivate	Activates (brings to the foreground) a window.
WinActivateBottom	Same as WinActivate except it activates the least recently used (bottommost) matching window rather than the newest.
WinClose	Closes a window.
WinGetActiveStats	Combines the functions of WinGetActiveTitle and WinGetPos into one command.
WinGetActiveTitle	Retrieves the title of the active window.
WinGetClass	Retrieves a window's class name.
WinGet	Retrieves a window's unique ID, process ID/name, or a list of its controls. It can also find all windows of a certain class, title, or text.
WinGetPos	Retrieves the position and size of a given window.
WinGetText	Retrieves the text from a window.
WinGetTitle	Retrieves the full title from a window.
WinHide	Hides a window.
WinKill	Forces a window to close.
WinMaximize	Enlarges a window to its maximum size.
WinMenuItem	Invokes a menu item from the menu bar of a window.
WinMinimize	Collapses a window into a button on the task bar.
WinMinimizeAll	Minimizes all windows.
WinMinimizeAllUndo	Undoes a previous WinMinimizeAll .
WinMove	Changes the position and (optionally) the size of a window.

WinRestore	Unminimizes or unmaximizes a window if it is minimized or maximized.
WinSet	Makes a window "always on top" and/or transparent.
WinSetTitle	Changes the title of a window.
WinShow	Unhides a hidden window.
WinWait	Waits until the requested window exists.
WinWaitActive	Waits until the requested window is active.
WinWaitClose	Waits until the requested window does not exist.
WinWaitNotActive	Waits until the requested window is not active.
#CommentFlag	Changes the script's comment symbol from semicolon to some other string.
#ErrorStdOut	Sends any syntax error that prevents a script from launching to stdout rather than displaying a dialog.
#EscapeChar	Changes the script's escape character (for example: accent vs. backslash).
#HotkeyInterval	Along with #MaxHotkeysPerInterval, specifies the rate of hotkey activations beyond which a warning dialog will be displayed.
#HotkeyModifierTimeout	Affects the behavior of hotkey modifiers: CTRL, ALT, WIN, and SHIFT.
#Hotstring	Changing hotstring options or ending characters.
#Include	Causes the script to behave as though the specified file's contents are present.
#MaxHotkeysPerInterval	Along with #HotkeyInterval, specifies the rate of hotkey activations beyond which a warning dialog will be displayed.
#MaxMem	Sets the maximum capacity of each variable to the specified number of megabytes.
#MaxThreads	Sets the maximum number of simultaneous threads.
#MaxThreadsBuffer	Causes some or all hotkeys to buffer rather than ignore keypresses when their #MaxThreadsPerHotkey limit has been reached.
#MaxThreadsPerHotkey	Sets the maximum number of simultaneous threads per hotkey.
#NoTrayIcon	Disables the showing of a tray icon.
#Persistent	Keeps a non-hotkey script permanently running (that is, until ExitApp is encountered).
#SingleInstance	Prevents more than one instance of a script from existing simultaneously.
#WinActivateForce	Skips the gentle method of activating a window and goes straight to the forceful method.

ClipWait

Waits until the [clipboard](#) contains text or files.

ClipWait [, SecondsToWait]

Parameters

SecondsToWait

If omitted, the command will wait indefinitely. Otherwise, it will wait no longer than this many seconds (can contain a decimal point). Specifying 0 is the same as specifying 0.5.

ErrorLevel

If the wait period expires, [ErrorLevel](#) will be set to 1. Otherwise (i.e. the clipboard contains text), ErrorLevel is set to 0.

Remarks

It's better to use this command than a loop of your own that checks to see if this clipboard is blank. This is because the clipboard is never opened by this command, and thus it performs better and avoids any chance of interfering with another application that may be using the clipboard. *This command checks for non-zero text contents in a loop with slight exponential back-off (max 0.5s).*

This command considers anything convertible to text (e.g. HTML) to be text. It also considers files, such as those copied in an Explorer window via Control-C, to be text. Such files are automatically converted to their filenames (with full path) whenever the [clipboard](#) variable (%clipboard%) is referred to in the script.

While the command is in a waiting state, new [threads](#) can be launched via [hotkey](#), [custom menu item](#), or [timer](#).

Related

[Clipboard](#), [WinWait](#), [KeyWait](#)

Example

```
clipboard = ; Empty the clipboard
Send, ^c
ClipWait, 2
```

```
if ErrorLevel <> 0
{
    MsgBox, The attempt to copy text onto the clipboard failed.
    return
}
Clipboard, clipboard = %clipboard%
return
```

#EnvSet

Writes a value to a [variable](#) contained in the environment.

EnvSet, EnvVar, Value

Parameters

EnvVar	Name of the environment variable to use, e.g. "COMSPEC" or "PATH".
Value	Value to set the environment variable to.

ErrorLevel

[ErrorLevel](#) is set to 1 if there was a problem or 0 otherwise.

Remarks

This command exists separately from SetEnv because unlike AutoIt2, AutoHotkey does not store its variables in the environment. This is because performance would be worse and also because the OS limits environment variables to 32K (AutoHotkey variables, including the [clipboard](#), are essentially unlimited in size).

Unless [EnvUpdate](#) is used, an [environment variable](#) set in this way will only be accessible to programs that the script spawns via [Run](#) or [RunWait](#). Once the script terminates, the variables will cease to exist.

Note: An [EnvGet](#) command is not provided because referencing a blank or undefined variable will automatically trigger that behavior. For example, in the following line, the value of the [ProgramFiles](#) variable (if undefined in the script) will be fetched from the environment:

TargetFile = %ProgramFiles%\Something\setup.ini [Use EnvGet to retrieve env vars. AHK_X11 behaves like #NoEnv in newer AHK versions by default.](#)

Related

[EnvUpdate](#), [SetEnv](#), [Run](#), [RunWait](#)

Example

EnvSet, AutGUI, Some text to put in the variable.

#EnvUpdate

Notifies the OS and all running applications that environment variable(s) have changed.

EnvUpdate

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

Remarks

Refreshes the OS environment. Similar effect as logging off and then on again.

For example, after making changes to the following registry key on Windows NT/2000/XP, EnvUpdate could be used to broadcast the change: HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\Environment

Due to the nature of this feature, there is a slight chance that keyboard and mouse events will lag (be delayed) while the command is running if the keyboard/mouse hooks are installed.

Related

[EnvSet](#), [RegWrite](#)

Example

#Drive [v1.0.22+]

Ejects/retracts the tray in a CD or DVD drive, or sets a drive's volume label.

Drive, Sub-command [, Drive , Value]

The sub-command, drive, and value parameters are dependent upon each other and their usage is described below.

label, Drive [, NewLabel]: Changes Drive's volume label to be NewLabel (if NewLabel is omitted, the drive will have no label). Drive is the drive letter followed by a colon and an optional backslash (might also work on UNC's and mapped drives). Example: Drive, Label, C:, Seagate200

To retrieve the current label, follow this example: DriveGet, OutputVar, Label, C:

lock, Drive [v1.0.24+]: Prevents a drive's eject feature from working until the script or another program unlocks it, or the system is restarted. Example: "Drive, Lock, D:". Most drives cannot be "locked open". However, locking the drive while it is open will probably result in it becoming locked the moment it is closed. This command has no effect on drives that do not support locking (such as most read-only drives), nor is it likely to work on non-IDE drives on Windows 95/98/Me. If a drive is locked by a script and that script exits, the drive will stay locked until another script or program unlocks it, or the system is restarted. If the specified drive does not exist or does not support the locking feature, ErrorLevel is set to 1. Otherwise, it is set to 0.

Unlock, Drive [v1.0.24+]: Reverses the above. On Window NT/2000/XP or later, Unlock needs to be executed multiple times if the drive was locked multiple times (at least for some drives). For example, if "Drive, Lock, D:" was executed three times, three executions of "Drive, Unlock, D:" might be needed to unlock it. Because of this and the fact that there is no way to determine whether a drive is currently locked, it is often useful to keep track of its lock state in a variable.

Eject [, Drive, 1]: Ejects or retracts the tray of a CD or DVD drive (if Drive is omitted, the default CD/DVD drive will be used). To eject the tray, omit the last parameter. To retract/close the tray, specify 1 for the last parameter. Example for retraction: Drive, Eject, D:, 1

If the tray is already in the correct state (open or closed), ErrorLevel is set to 0 (i.e. "no error"). This command will probably not work on a network drive or non-CD/DVD drive; if it fails in such cases or for any other reason, ErrorLevel is set to 1.

It may be possible to detect the previous tray state by measuring the time the command takes to complete. For example, to toggle the tray to the opposite state (open or closed), try using a hotkey such as the following:

```
#e:  
Drive, Eject  
;If the command completed quickly, the tray was probably already ejected.  
;In that case, retract it:  
if A_TimeSinceThisHotkey < 1000 ;Adjust this time if needed.
```

~~Drive, Eject,, 1~~
~~return~~

This command waits for the ejection or retraction to complete before allowing the script to continue. In addition, due to the nature of this command, there might be some keyboard or mouse lag (reduced responsiveness) during the operation if the script executing it has the keyboard and/or mouse hooks installed. This limitation will be resolved in a future release, if possible.

To determine the media status of a CD or DVD drive (playing, stopped, open, etc.), see [DriveGet](#).

-

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

-

Remarks

None.

-

Related

[DriveGet](#), [DriveSpaceFree](#)

-

Example

[Drive, Label, D:, BackupDrive](#)

[Drive, Eject,, 1 ; Retract \(close\) the tray of the default CD or DVD drive.](#)

#[DriveGet](#) [v1.0.10+]

Retrieves various types of information about the computer's drive(s).

[DriveGet, OutputVar, Cmd \[, Value\]](#)

-

Parameters

OutputVar	The name of the variable in which to store the result of <i>Cmd</i> .
Cmd, Value	See list below.

Cmd, Value

The *Cmd* and *Value* parameters are dependent upon each other and their usage is described below. If a problem is encountered *OutputVar* is made blank and *ErrorLevel* is set to 1.

list [, Type]: Sets *OutputVar* to be a string of letters, one character for each drive letter in the system. Example: ACDEZ. If *Type* is omitted, all drive types are retrieved. Otherwise, *Type* should be one of the following words to retrieve only a specific type of drive: CDROM, REMOVABLE, FIXED, NETWORK, RAMDISK, UNKNOWN.

Capacity (or Cap), Path: Retrieves the total capacity of *Path* (e.g. C:\) in megabytes. Use DriveSpaceFree to determine the free space.

Filesystem (or FS), Drive: Retrieves the type of *Drive*'s file system, where *Drive* is the drive letter followed by a colon and an optional backslash, or a UNC name such \\server1\share1. *OutputVar* will be set to one of the following words: FAT, FAT32, NTFS, EDFS (typically indicates a CD), UDF (typically indicates a DVD). *OutputVar* will be made blank and *ErrorLevel* set to 1 if the drive does not contain formatted media.

Label, Drive: Retrieves *Drive*'s volume label, where *Drive* is the drive letter followed by a colon and an optional backslash, or a UNC name such \\server1\share1. To change the label, follow this example: Drive, Label, C:, MyLabel

Serial, Drive: Retrieves *Drive*'s volume serial number expressed as decimal integer, where *Drive* is the drive letter followed by a colon and an optional backslash, or a UNC name such \\server1\share1. See SetFormat for how to convert it to hexadecimal.

Type, Path: Retrieves *Path*'s drive type, which is one of the following words: Unknown, Removable, Fixed, Network, CDROM, RAMDisk.

Status, Path: Retrieves *Path*'s status, which is one of the following words: Unknown (might indicate unformatted/RAW), Ready, NotReady (typical for removable drives that don't contain media), Invalid (*Path* does not exist or is a network drive that is presently inaccessible, etc.)

StatusCD [, Drive] [v1.0.22+]: Retrieves the media status of a CD or DVD drive (if *Drive* is omitted, the default CD/DVD drive will be used). *OutputVar* is made blank if the status cannot be determined. Otherwise, it is set to one of the following words:

not ready	The drive is not ready to be accessed, perhaps due to being engaged in a write operation.
open	The drive contains no disc, or the tray is ejected.
playing	The drive is playing a disc.
paused	The previously playing audio or video is now paused.
seeking	The drive is seeking.
stopped	The drive contains a disc but is not currently accessing it.

This command will probably not work on a network drive or non-CD/DVD drive; if it fails in such cases or for any other reason, *OutputVar* is made blank and *ErrorLevel* is set to 1.

If the tray was recently closed, there may be a delay before the command completes. In addition, due to the nature of this command, there might be some keyboard or mouse lag (reduced responsiveness) during the operation if the script executing it has the keyboard and/or mouse hooks installed. This limitation will be resolved in a future release, if possible.

To eject or retract the tray, see the Drive command.

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

-

Remarks

Some of the commands will accept a network share name as *Path*, such as \\MyServer\\MyShare\

-

Related

[Drive](#), [DriveSpaceFree](#)

-

Example

```
; This is a working example script.
FileSelectFolder, folder, , 3, Pick a drive to analyze.
if folder
    return
DriveGet, list, list
DriveGet, cap, capacity, %folder%
DrivespaceFree, free, %folder%
DriveGet, fs, fs, %folder%
DriveGet, label, label, %folder%
DriveGet, serial, serial, %folder%
DriveGet, type, type, %folder%
DriveGet, status, status, %folder%
MsgBox All Drives. %list%`nSelected Drive: %folder%`nDrive Type: %type%`nStatus: %status%`nCapacity: %cap% M`nFree Space: %free% M`nFilesystem: %fs%`nVolume Label: %label%`nSerial Number: %serial%
```

DriveSpaceFree

Retrieves the free disk space of a drive, in Megabytes.

DriveSpaceFree, OutputVar, Path

Parameters

OutputVar	The variable in which to store the result, which is rounded down to the nearest whole number.
path	Path of drive to receive information from. Since NTFS supports mounted volumes and directory junctions, different amounts of free space might be available in different folders of the same "drive" in some cases.

Remarks

OutputVar is set to the amount of free disk space in Megabytes (rounded down to the nearest Megabyte).

Related

[Drive](#), [DriveGet](#)

Example

DriveSpaceFree, FreeSpace, %A_Home%

FileAppend

Appends text to a text file.

FileAppend, Text, Filename

Parameters

Text	The text to append to the file. This text can include newline characters by using `n.
Filename	The name of the file to be appended, which is assumed to be in %A_WorkingDir% if an absolute path isn't specified. To append in binary mode rather than text mode, prepend an asterisk to the filename. This causes each newline character (`n) to be written as a single linefeed (LF) rather than the Windows standard of CR+LF. Example: *C:\My Unix File.txt

ErrorLevel

[ErrorLevel](#) is set to 1 if there was a problem or 0 otherwise.

Remarks

To overwrite an existing file, delete it with [FileDelete](#) prior to using FileAppend.

Related

[FileReadLine](#), [FileReadLine](#), [IniWrite](#), [file-read loop](#), [FileDelete](#)

Example

FileAppend, Another line. `n, c:\My Documents\Test.txt

FileCopy

Copies one or more files. *File modification date is not preserved, file name matching is case sensitive.*

FileCopy, SourcePattern, DestPattern [, Flag]

Parameters

SourcePattern	The name of a single file or folder, or a wildcard pattern such as /tmp/*.tmp. <i>SourcePattern</i> is assumed to be in %A_WorkingDir% if an absolute path isn't specified.
DestPattern	The name or pattern of the destination, which is assumed to be in %A_WorkingDir% if an absolute path isn't specified. To perform a simple copy -- retaining the existing file name(s) -- specify only the folder name as shown in these functionally identical examples: FileCopy, %A_Home%/*.txt, %A_Home%/My Folder FileCopy, %A_Home%/*.txt, %A_Home%/My Folder/*.*
Flag	(optional) this flag determines whether to overwrite files if they already exist:

ErrorLevel

ErrorLevel is set to the number of files that could not be copied due to an error, or 0 otherwise.

Remarks

The operation will continue even if error(s) are encountered.

To copy an entire folder (including its subfolders), use [FileCopyDir](#).

Related

[FileMove](#), [FileCopyDir](#), [FileMoveDir](#), [FileDelete](#)

Examples

```
FileCopy, C:\My Documents\List1.txt, D:\Main Backup\ ; Make a copy but keep the orig. file name.
```

```
FileCopy, C:\My File.txt, C:\My File New.txt ; Copy a file into the same folder by providing a new name.
```

```
FileCopy, C:\Folder1\*.txt, D:\New Folder\*.bkp ; Copy to new location and give new extension.
```

#FileCopyDir

~~Copies a directory and all sub-directories and files (similar to xcopy).~~

~~FileCopyDir, Source, Dest [, Flag]~~

Parameters

Source	Name of the source directory (with no trailing backslash), which is assumed to be in %A_WorkingDir% if an absolute path isn't specified. For example: C:\My Folder
Dest	Name of the destination dir (with no trailing backslash), which is assumed to be in %A_WorkingDir% if an absolute path isn't specified. For example: C:\Copy of My Folder
Flag	(optional) this flag determines whether to overwrite files if they already exist: 0 = (default) do not overwrite existing files 1 = overwrite existing files

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

Remarks

If the destination directory structure doesn't exist it will be created if possible.

Since the operation will recursively copy a folder along with all its subfolders and files, the result of copying a folder to a destination somewhere inside itself is undefined. To work around this, first copy it to a destination outside itself, then use FileMoveDir to move that copy to the desired location.

Due to the nature of this command, there might be some keyboard or mouse lag (reduced responsiveness) during the operation if the script executing it has the keyboard and/or mouse hooks installed. This limitation will be resolved in a future release, if possible.

Related

[FileMoveDir](#), [FileCopy](#), [FileMove](#), [FileDelete](#), [file loops](#), [FileSelectFolder](#), [SplitPath](#)

Examples

~~FileCopyDir, C:\My Folder, C:\Copy of My Folder~~

~~Example #2: A working script that prompts you to copy a folder.~~

~~FileSelectFolder, SourceFolder, , 3, Select the folder to copy~~

~~if SourceFolder~~

~~return~~

~~Otherwise, continue.~~

~~FileSelectFolder, TargetFolder, , 3, Select the folder IN WHICH to create the duplicate folder.~~

~~if TargetFolder~~

~~return~~

~~Otherwise, continue.~~

```
MsgBox, 4, , A copy of the folder "%SourceFolder%" will be put into "%TargetFolder%". Continue?
If MsgBox, No
    return
SplitPath, SourceFolder, SourceFolderName, Extract only the folder name from its full path.
FileCopyDir, %SourceFolder%, %TargetFolder%\%SourceFolderName%
If ErrorLevel > 0
    MsgBox The folder could not be copied, perhaps because a folder of that name already exists in "%TargetFolder%".
return
```

FileCreateDir

Creates a directory/folder.

FileCreateDir, DirName

Parameters

DirName

Name of the directory to create, which is assumed to be in %A_WorkingDir% if an absolute path isn't specified.

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

Remarks

This command will also create all parent directories given in *DirName* if they do not already exist.

Related

[FileRemoveDir](#)

Example

#FileCreateShortcut

Creates a shortcut (.lnk) file.

~~FileCreateShortcut, Target, LinkFile [, WorkingDir, Args, Description, IconFile, ShortcutKey, IconNumber, RunState]~~

-
Parameters

Target	<p>Name of the file that the shortcut refers to, which should include an absolute path unless the file is integrated with the system (e.g. Notepad.exe). The file does not have to exist at the time the shortcut is created; in other words, shortcuts to invalid targets can be created.</p>
LinkFile	<p>Name of the shortcut file to be created, which is assumed to be in %A_WorkingDir% if an absolute path isn't specified. Be sure to include the .lnk extension. If the file already exists, it will be overwritten.</p>
WorkingDir	<p>Directory that will become <i>Target</i>'s current working directory when the shortcut is launched. If blank or omitted, the shortcut will have a blank "Start in" field and the system will provide a default working directory when the shortcut is launched.</p>
Args	<p>Parameters that will be passed to <i>Target</i> when it is launched. Separate parameters with spaces. If a parameter contains spaces, enclose it in double quotes.</p>
Description	<p>Comments that describe the shortcut (used by the OS to display a tooltip, etc.)</p>
IconFile	<p>The full path and name of the icon to be displayed for <i>LinkFile</i>. It must either be an ico file or the very first icon of an EXE or DLL.</p>
ShortcutKey	<p>A single letter, number, or the name of a single key from the key list (mouse buttons and other non-standard keys might not be supported). Do not include modifier symbols. Currently, all shortcut keys are created as CTRL+ALT shortcuts. For example, if the letter B is specified for this parameter, the shortcut key will be CTRL+ALT+B.</p> <p>For Windows 9x: A reboot might be required to get the shortcut key into effect. Alternatively, you can open the properties dialog for the shortcut and recreate the shortcut key to get it into effect immediately.</p>
IconNumber	<p>In v1.0.24+, to use an icon in <i>IconFile</i> other than the first, specify that number here. For example, 2 is the second icon.</p>

RunState

In v1.0.24+, to have *Target* launched minimized or maximized, specify one of the following digits:

- 1: Normal (this is the default)
- 3: Maximized
- 7: Minimized

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

Remarks

Target might not need to include a path if the target file resides in one of the folders listed in the system's PATH environment variable.

The *ShortcutKey* of a newly created shortcut will have no effect unless the shortcut file resides on the desktop or somewhere in the Start Menu. If the *ShortcutKey* you choose is already in use, your new shortcut takes precedence.

You cannot define the window run-state to be "Minimized" or "Maximized"; the shortcuts always use "Normal."

An alternative way to create a shortcut to a URL is the following example, which creates a special URL shortcut. Change the first two parameters to suit your preferences:

`IniWrite, http://www.google.com, C:\My Shortcut.url, InternetShortcut, URL`

The following may be optionally added to assign an icon to the above:

`IniWrite, <IconFile>, C:\My Shortcut.url, InternetShortcut, IconFile`

`IniWrite, 0, C:\My Shortcut.url, InternetShortcut, IconIndex`

In the above, replace 0 with the index of the icon (0 is used for the first icon) and replace <IconFile> with a URL, EXE, DLL, or ICO file. Examples: C:\Icons.dll, C:\App.exe, http://www.somedomain.com/ShortcutIcon.ico

The operating system will treat a .URL file created by the above as a real shortcut even though it is a plain text file rather than a .LNK file.

Related

[FileGetShortcut](#), [FileAppend](#)

Example

;The letter "i" in the last parameter makes the shortcut key be Ctrl+Alt+I:

`FileCreateShortcut, Notepad.exe, %HOMEDRIVE%%HOMEPATH%\Desktop\My Shortcut.lnk, C:\,"%A_ScriptFullPath%", My Description, C:\My Icon.ico, i`

#FileDelete

Deletes one or more files.

FileDelete, FilePattern

Parameters

FilePattern

The name of a single file or a wildcard pattern such as /tmp/*.tmp. *FilePattern* is assumed to be in %A_WorkingDir% if an absolute path isn't specified.

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

Remarks

None

Related

[FileRecycle](#), [FileCopy](#), [FileMove](#)

Example

FileDelete, /a/b/*.tmp

#FileGetAttrib

~~Retrieves a code string representing a file or folder's attributes.~~

FileGetAttrib, OutputVar [, Filename]

-
Parameters

OutputVar

~~The name of the variable in which to store the retrieved text.~~

Filename

~~The name of the target file, which is assumed to be in %A_WorkingDir% if an absolute path isn't specified. If omitted, the current file of the innermost enclosing File-Loop will be used instead.~~

-
ErrorLevel

~~ErrorLevel is set to 1 if there was a problem or 0 otherwise.~~

-
Remarks

~~String returned could contain a combination of these letters "RASHNDOCT"~~

R=READONLY

A=ARCHIVE

S=SYSTEM

H=HIDDEN

N=NORMAL

D=DIRECTORY

O=OFFLINE

C=COMPRESSED

T=TEMPORARY

-
Related

~~FileSetAttrib, FileGetTime, FileSetTime, FileGetSize, FileGetVersion, File-Loop~~

-
Example

FileGetAttrib, OutputVar, C:\New Folder

#FileGetShortcut [v1.0.24]

Retrieves information about a shortcut (.lnk) file, such as its target file.

```
FileGetShortcut, LinkFile [, OutTarget, OutDir, OutArgs, OutDescription, OutIcon, OutIconNum, OutRunState]
```

Parameters

LinkFile	Name of the shortcut file to be analyzed, which is assumed to be in %A_WorkingDir% if an absolute path isn't specified. Be sure to include the .lnk extension.
OutTarget	Name of the variable in which to store the shortcut's target (not including any arguments it might have). Example: C:\WINDOWS\system32\notepad.exe
OutDir	Name of the variable in which to store the shortcut's working directory. Example: C:\My Documents. To expand any environment variables that might be present (such as %windir%), follow this example: Transform, OutDir, Deref, %OutDir%
OutArgs	Name of the variable in which to store the shortcut's parameters (blank if none).
OutDescription	Name of the variable in which to store the shortcut's comments (blank if none).
OutIcon	Name of the variable in which to store the filename of the shortcut's icon (blank if none).
OutIconNum	Name of the variable in which to store the shortcut's icon number within the icon file (blank if none). This value is most often 1, which means the first icon.
OutRunState	Name of the variable in which to store the shortcut's initial launch state, which is one of the following digits: 1: Normal 3: Maximized 7: Minimized

ErrorLevel

If there was a problem—such as *LinkFile* not existing—all the output variables are made blank and ErrorLevel is set to 1. Otherwise, ErrorLevel is set to 0.

Remarks

Any of the output variables may be omitted if the corresponding information is not needed.

Related

[FileCreateShortcut](#), [SplitPath](#)

Example

```
FileSelectFile, file,,, Pick a shortcut to analyze., Shortcuts (*.lnk)
if file =
    return
FileGetShortcut, %file%, OutTarget, OutDir, OutArgs, OutDesc, OutIcon, OutIconNum, OutRunState
Transform, OutDir, Deref, %OutDir%, Resolve any environment variables, such as %WinDir%.
MsgBox %OutTarget%`n%OutDir%`n%OutArgs%`n%OutDesc%`n%OutIcon%`n%OutIconNum%`n%OutRunState%
```

FileGetSize

Retrieves the size of a file in bytes, KB, or MB.

FileGetSize, OutputVar [, Filename, Units]

Parameters

OutputVar	The name of the variable in which to store the retrieved size (rounded down to the nearest whole number).
Filename	The name of the target file, which is assumed to be in %A_WorkingDir% if an absolute path isn't specified. If omitted, the current file of the innermost enclosing File-Loop will be used instead.
Units	If present, this parameter causes the result to be returned in units other than bytes: K = kilobytes M = megabytes

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

Remarks

Files over 4 gigabytes in size are supported (even if *Units* is bytes).

If the target file is a directory, the size will be reported as whatever the OS believes it to be (probably zero in all cases).

Related

[FileGetAttrib](#), [FileSetAttrib](#), [FileGetTime](#), [FileSetTime](#), [FileGetVersion](#), [File-loop](#)

Example

FileGetSize, size, %A_Home%/test.doc ; Retrieve the size in bytes.

FileGetSize, size, /tmp/test.doc, K ; Retrieve the size in Kbytes.

#FileGetTime

Retrieves the datetime stamp of a file or folder:

~~FileGetTime, OutputVar [, Filename, WhichTime]~~

Parameters

OutputVar	The name of the variable in which to store the retrieved date time in format YYYYMMDDHH24MISS. The time is your own local time, not UTC/GMT.
Filename	The name of the target file, which is assumed to be in %A_WorkingDir% if an absolute path isn't specified. If omitted, the current file of the innermost enclosing File Loop will be used instead.
WhichTime	Which timestamp to retrieve: M = Modification time (this is the default if the param is omitted)

-
ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

-
Remarks

A file's last access time might not be as precise on Win9x as it is on NT based operating systems.

See YYYYMMDDHH24MISS for a discussion of dates and times.

-
Related

~~FileSetTime~~, FormatTime, If var is [not] type, FileGetAttrib, FileSetAttrib, FileGetSize, FileGetVersion, File loop, EnvAdd, EnvSub

-
Example

~~FileGetTime~~, OutputVar, C:\My Documents\test.doc ; Retrieves the modification time by default.

~~FileGetTime~~, OutputVar, C:\My Documents\test.doc, C ; Retrieves the creation time.

#FileGetVersion

Retrieves the version of a file.

~~FileGetVersion~~, OutputVar [, filename]

-
Parameters

OutputVar

The name of the variable in which to store the retrieved size. The value is rounded down to the nearest whole number.

Filename

The name of the target file, which is assumed to be in %A_WorkingDir% if an absolute path isn't specified. If omitted, the current file of the innermost enclosing File-Loop will be used instead.

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

Remarks

Most non-executable files (and even some EXEs) won't have a version, and thus the OutputVar will be blank in these cases.

Related

[FileGetAttrib](#), [FileSetAttrib](#), [FileGetTime](#), [FileSetTime](#), [FileGetSize](#), [File-loop](#)

Example

```
FileGetVersion, version, C:\My Application.exe
```

```
FileGetVersion, version, %ProgramFiles%\AutoHotkey\AutoHotkey.exe
```

#FileInstall

Includes the specified file inside the compiled script (Ahk2Exe).

```
FileInstall, Source, Dest, Flag
```

Parameters**Source**

The full path and name of the file to add to the compiled EXE. It **must not** contain double quotes, variable references (e.g. %ProgramFiles%), or wildcards.

Dest

When Source is extracted from the EXE, this is the name of the file to be created. It is assumed to be in %A_WorkingDir% if an absolute path isn't specified. Unlike Source, variable references may be used.

Flag

(optional) this flag determines whether to overwrite files if they already exist.

~~0 = (default) do not overwrite existing files~~
~~1 = overwrite existing files~~

-
ErrorLevel

~~ErrorLevel is set to 1 if there was a problem or 0 otherwise.~~

-
Remarks

~~This command is a directive for the Ahk2Exe compiler that allows you to add extra files to the resulting compiled script. Later, when the compiled script is run, the files are extracted back out onto the disk.~~

~~The file Source is added during script compilation . When the compiled script is executed and the same "FileInstall" command has been reached, the file is then extracted to Dest.~~

~~Files added to a script are compressed and also encrypted.~~

~~If this command is used in an normal (uncompiled) script, a simple file copy will be performed instead — this will help the testing of scripts that will eventually be compiled.~~

-
Related

[FileCopy](#)

-
Example

~~FileInstall, C:\My Documents\My File.txt, %ProgramFiles%\My Application\Readme.txt, 1~~

#FileMove

~~Moves or renames one or more files.~~

FileMove, SourcePattern, DestPattern [, Flag]

-
Parameters

SourcePattern	The name of a single file or a wildcard pattern such as C:\Temp*.tmp. SourcePattern is assumed to be in %A_WorkingDir% if an absolute path isn't specified.
DestPattern	The name or pattern of the destination, which is assumed to be in %A_WorkingDir% if an absolute path isn't specified. To perform a simple move — retaining the existing file name(s) — specify only the folder name as shown in these functionally identical examples: FileMove, C:*.txt, C:\My Folder FileMove, C:*.txt, C:\My Folder*.*
Flag	(optional) this flag determines whether to overwrite files if they already exist: 0 = (default) do not overwrite existing files 1 = overwrite existing files

-

ErrorLevel

ErrorLevel is set to the number of files that could not be moved due to an error, or 0 otherwise.

-

Remarks

The operation will continue even if error(s) are encountered.

If the source and destination paths are on different volumes a copy and delete operation is performed rather than a move.

To move or rename a folder, use FileMoveDir.

-

Related

[FileCopy](#), [FileCopyDir](#), [FileMoveDir](#), [FileDelete](#)

-

Examples

FileMove, C:\My Documents\List1.txt, D:\Main Backup\ ; Move the file without renaming it.

FileMove, C:\File Before.txt, C:\File After.txt ; Rename a single file.

FileMove, C:\Folder1*.txt, D:\New Folder*.bkp ; Move and rename files to a new extension.

#FileMoveDir

Moves a directory and all sub-directories and files. It can also rename a directory.

FileMoveDir, Source, Dest [, Flag]

Parameters

Source	Name of the source directory (with no trailing backslash), which is assumed to be in %A_WorkingDir% if an absolute path isn't specified. For example: C:\My Folder
Dest	Name of the destination directory (with no trailing backslash), which is assumed to be in %A_WorkingDir% if an absolute path isn't specified. For example: D:\My Folder
Flag	<p>(options) Specify one of the following single characters:</p> <p>0 – (default) do not overwrite existing files 1 – overwrite existing files R – Rename the directory rather than moving it. Although "renaming" normally has the same effect as "moving", it is helpful in cases where you want "all or none" behavior; that is, when you don't want the operation to be only partially successful if Source or one of its files is locked (in use). Although this method cannot move Source onto a different volume, it can move it to any other directory on its own volume. The operation will fail if Dest already exists as a file or directory. [Requires v1.0.22+]</p>

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

Remarks

If the source and destination are on different volumes or UNC paths, a copy/delete operation will be performed rather than a move.

If the destination already exists and the overwrite flag is specified, the source directory will be moved **inside** the destination.

Due to the nature of this command, there might be some keyboard or mouse lag (reduced responsiveness) during the operation if the script executing it has the keyboard and/or mouse hooks installed. This limitation will be resolved in a future release, if possible.

Related

~~FileRead, FileCopyDir, FileCopy, FileMove, FileDelete, FileLoops, FileSelectFolder, SplitPath~~

-

Example

~~FileMoveDir, C:\My Folder, D:\My Folder ; Move to a new drive.~~
~~FileMoveDir, C:\My Folder, C:\My Folder (renamed), +~~

FileReadLine

Reads the specified line from a file and stores the text in a [variable](#).

FileReadLine, OutputVar, Filename, LineNum

Parameters

OutputVar	The name of the variable in which to store the retrieved text.
Filename	The name of the file to access, which is assumed to be in %A_WorkingDir% if an absolute path isn't specified.
LineNum	Which line to read (1 is the first, 2 the second, and so on).

ErrorLevel

[ErrorLevel](#) is set to 1 if there was a problem or 0 otherwise.

Remarks

Although any leading and trailing tabs and spaces present in the line will be written to *OutputVar*, the newline character (`n) at the end of the line will not. Tabs and spaces can be trimmed from both ends of any variable by assigning it to itself while [AutoTrim](#) is off (the default). For example: MyLine = %MyLine%

~~Lines up to 65,534 characters long can be read. If the length of a line exceeds this, the remaining characters cannot be retrieved by this command (use a file read loop instead).~~

Related

[FileAppend](#), [file-read loop](#), [IniRead](#)

Example

```
i = 0
Loop
{
    i += 1
    FileReadLine, line, %A_Home%/Documents/ContactList.txt, %i%
    if ErrorLevel <> 0
        break
    MsgBox, 4,, Line %%i%% is "%line%". Continue?
    IfMsgBox, No
        return
}
MsgBox, The end of the file has been reached or there was a problem.
return
```

FileRead

Reads the specified file into a [variable](#).

FileRead, OutputVar, Filename

Parameters

OutputVar	The name of the variable in which to store the retrieved text.
Filename	The name of the file to access, which is assumed to be in %A_WorkingDir% if an absolute path isn't specified.

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

Example

```
i = 0
    FileRead, text, %A_Home%/Documents/ContactList.txt
    MsgBox, text is "%text%"
```

#FileRecycle

Sends a file or directory to the recycle bin, if possible.

FileRecycle, FilePattern

Parameters

FilePattern

The name of a single file or a wildcard pattern such as /tmp/*.tmp. *FilePattern* is assumed to be in [%A_WorkingDir%](#) if an absolute path isn't specified.
To recycle an entire directory, provide its name without a trailing backslash.

ErrorLevel

[ErrorLevel](#) is set to 1 if there was a problem or 0 otherwise.

Remarks

None

Related

[FileRecycleEmpty](#), [FileDelete](#), [FileCopy](#), [FileMove](#)

Example

FileRecycle, %A_Home%/temp files/*.tmp

FileRecycleEmpty

Empties the recycle bin.

FileRecycleEmpty *[, DriveLetter]*

Parameters

DriveLetter

If omitted, the recycle bin for all drives is emptied. Otherwise, specify a drive letter such as C:\

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

Remarks

This command requires MS Internet Explorer 4 or greater to be installed.

Related

[FileRecycle](#), [FileDelete](#), [FileCopy](#), [FileMove](#)

Example

FileRecycleEmpty

#FileRemoveDir

Deletes a directory/folder.

FileRemoveDir, DirName [, Recurse?]

Parameters

DirName	Name of the directory to delete, which is assumed to be in %A_WorkingDir% if an absolute path isn't specified.
Recurse?	0 (default): Do not remove files and sub-directories contained in <i>DirName</i> . In this case, if <i>DirName</i> is not empty, no action will be taken and ErrorLevel will be set to 1. 1: Remove all files and subdirectories (like the DOS DelTree command).

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

Remarks

None

Related

[FileCreateDir](#), [FileDelete](#)

Example

```
FileRemoveDir, C:\Download\Temp  
FileRemoveDir, C:\Download\Temp, 1
```

FileSelectFile

Displays a standard dialog that allows the user to select file(s).

FileSelectFile, OutputVar [, Options, RootDir, Prompt, **Filter**]

Parameters

OutputVar	The name of the variable in which to store the name of the folder selected by the user. This will be made blank if the user cancels the dialog (i.e. does not wish to select a file).
Options	If omitted, it will default to zero, which is the same as having NONE of the below options. Otherwise, add up one or more of the desired values (for example, to use options 1 and 16, specify a value of 17): 1 – File Must Exist 2 – Path Must Exist 4 – Allow MultiSelect (see Examples for how to extract the individual files) 8 – Prompt to Create New File 16 – Prompt to OverWrite File If the "Prompt to Overwrite" option is present without the "Prompt to Create" option, the dialog will contain a Save button rather than an Open button (this behavior is due to a quirk in Windows). In v1.0.14+, specify the letter S as the first character to cause the dialog to always contain a Save button (e.g. S16 or just S).
RootDir	Root (starting) directory, which is assumed to be a subfolder in %A_WorkingDir% if an absolute path isn't specified. If omitted or blank, the starting directory will be a default that might depend on the OS version. In WinNT/2k/XP and beyond, it will likely be the directory most recently selected by the user during a prior use of FileSelectFile. In v1.0.09+, to specify a default filename for the dialog's edit field, add that filename after the directory as in this example: C:\My Documents\My Pictures\Default Image Name.gif. Only the naked filename (with no path) will be used as the default.
Prompt	Text displayed in the window to instruct the user what to do. If omitted or blank, it will default to "Select File - %A_SCRIPTNAME%" (i.e. the name of the current script).
Filter	Indicates which types of files are shown by the dialog. Example: Documents (*.txt) Example: Audio (*.wav; *.mp2; *.mp3) If omitted, the filter defaults to All Files (*.*) An option for Text Documents (*.txt) will also be available in the dialog's "files of type" menu.

Otherwise, the filter uses the indicated string but also provides an option for All Files (*.*) in the dialog's "files of type" menu. To include more than one file extension in the filter, separate them with semicolons as illustrated in the example above.

Remarks

OutputVar will be set to the full path and name of file chosen by the user. If the user didn't select a file (e.g. pressed CANCEL), *OutputVar* will be made blank.

If *Options* permit it and the user selected more than one file, *OutputVar* will be set to a list of items, each of which is followed by a newline (`n) character (including the last item). The first item in the list is the directory that contains all of the selected files.

The other items are the selected filenames, not including their path. For example:

C:\My Documents\Test [this is the directory in which all the below files reside]

test1.txt [these names are the naked filenames: no path info]

test2.txt

... etc

(See Examples below for more information)

See the *RootDir* parameter above for how to put a default filename in the dialog's edit field.

Related

[FileSelectFolder](#), [MsgBox](#), [InputBox](#), [ToolTip](#), [SplitPath](#)

Example

```
FileSelectFile, SelectedFile, 3, , Open a file, Text Documents (*.txt)
if SelectedFile =
    MsgBox, The user didn't select anything.
else
    MsgBox, The user selected the following:`n%SelectedFile%

; MULTI-SELECT EXAMPLE:
FileSelectFile, files, 7 ; 7 = Multiselect existing files.
if files =
{
    MsgBox, The user pressed cancel.
    return
}
Loop, parse, files, `n
{
    if A_LoopField = ; A blank field marks the end of the list.
        break
    if a_index = 1
```

```

        MsgBox, The selected files are all contained in %A_LoopField%.
    else
    {
        MsgBox, 4, , The next file is %A_LoopField%. Continue?
        IfMsgBox, No, break
    }
}
return

```

FileSelectFolder

Displays a standard dialog that allows the user to select a folder.

FileSelectFolder, OutputVar [, RootPath, Options, Prompt]

Parameters

OutputVar	The name of the variable in which to store the user's selected folder. This will be made blank if the user cancels the dialog (i.e. does not wish to select a folder).
RootPath	Root (starting) directory, which must include an absolute path, e.g. %A_Home%/Documents/Pictures If omitted or blank, it will default to "My Computer".
Options	One of the following numbers: 0: None of the below options is enabled. 1 (default): A button is provided that allows the user to create new folders. This option has no effect unless the operating system is Windows Me/2000/XP or later. In v1.0.09+, add 2 to the above number to provide an edit field that allows the user to type the name of a folder. For example, a value of 3 for this parameter provides both an edit field and a "make new folder" button. If the user types an invalid folder name in the edit field, OutputVar will be set to the folder selected in the navigation tree rather than what the user entered, at least on Windows XP.
Prompt	Text displayed in the window to instruct the user what to do. If omitted or blank, it will default to "Select Folder - %A_SCRIPTNAME%" (i.e. the name of the current script).

Remarks

None.

Related

[FileSelectFile](#), [MsgBox](#), [InputBox](#), [ToolTip](#), [FileCopyDir](#), [FileMoveDir](#), [SplitPath](#)

Example

```
FileSelectFolder, OutputVar, , 3
if OutputVar =
    MsgBox, You didn't select a folder.
else
    MsgBox, You selected folder "%OutputVar%".
```

FileSetAttrib

Changes the attributes of one or more files or folders. Wildcards are supported.

FileSetAttrib, Attributes [, FilePattern, OperateOnFolders?, Recurse?]

Parameters

Attributes	The attributes to change (see Remarks).
FilePattern	The name of a single file or folder, or a wildcard pattern such as C:\Temp*.tmp. <i>FilePattern</i> is assumed to be in %A_WorkingDir% if an absolute path isn't specified. If omitted, the current file of the innermost enclosing File-Loop will be used instead.
OperateOnFolders?	0 (default) Folders are not operated upon (only files). 1 All files and folders that match the wildcard pattern are operated upon. 2 Only folders are operated upon (no files).

	Note: If FilePattern is a single folder rather than a wildcard pattern, it will always be operated upon regardless of this setting.
Recurse?	0 (default) Subfolders are not recursed into. 1 Subfolders are recursed into so that files and folders contained therein are operated upon if they match <FilePattern>. All subfolders will be recursed into, not just those whose names match <FilePattern>.

ErrorLevel

ErrorLevel is set to the number of files that failed to be changed (if any) or 0 otherwise.

Remarks

The Attributes parameter consists of a collection of operators and attribute letters.

Operators:

+	Turn on the attribute
-	Turn off the attribute
^	Toggle the attribute (set it to the opposite value of what it is now)

Attribute letters:

R = READONLY

A = ARCHIVE

S = SYSTEM

H = HIDDEN

~~N = NORMAL (this is only valid when used without any other attributes)~~

O = OFFLINE

T = TEMPORARY

X = EXECUTABLE

~~Note: Currently, the compression state of files cannot be changed with this command.~~

Currently, only executable rights (for the current user) can be toggled with this (it switches between 755 and 644 internally). Linux file system permission rights are much more capable than that. For more complex use cases, use chown or chmod with something like: RunWait, chmod 666 filename

ErrorLevel

ErrorLevel is set to the number of files that failed to be changed (if any) or 0 otherwise.

Related

[FileGetAttrib](#), [FileGetTime](#), [FileSetTime](#), [FileGetSize](#), [FileGetVersion](#), [File-loop](#)

Examples

FileSetAttrib, +RH, C:\MyFiles*.* , 1 ; +RH is identical to +R+H

FileSetAttrib, ^H, C:\MyFiles ; Toggle the folder's "hidden" attribute.

FileSetAttrib, -R+A, C:\New Text File.txt

FileSetAttrib, +A, C:*.ini, , 1 ; Recurse through all .ini files on the C drive.

FileSetTime

Changes the datetime stamp of one or more files or folders. Wildcards are supported.

FileSetTime [, YYYYMMDDHH24MISS, FilePattern, WhichTime, OperateOnFolders?, Recurse?]

Parameters

YYYYMMDDHH24MISS	If omitted, it defaults to the current time. Otherwise, specify the time to use for the operation (see Remarks for the format). Years prior to 1601 are not supported.
FilePattern	The name of a single file or folder, or a wildcard pattern such as /tmp/*.tmp. <i>FilePattern</i> is assumed to be in %A_WorkingDir% if an absolute path isn't specified. If omitted, the current file of the innermost enclosing File-Loop will be used instead.
WhichTime	Which timestamp to set: M = Modification time (this is the default if the param is omitted) C=Creation time A=Last access time
OperateOnFolders?	0 (default) Folders are not operated upon (only files). 1 All files and folders that match the wildcard pattern are operated upon.

	<p>2 Only folders are operated upon (no files).</p> <p>Note: If <i>FilePattern</i> is a single folder rather than a wildcard pattern, it will always be operated upon regardless of this setting.</p>
Recurse?	<p>0 (default) Subfolders are not recursed into.</p> <p>1 Subfolders are recursed into so that files and folders contained therein are operated upon if they match <i>FilePattern</i>. All subfolders will be recursed into, not just those whose names match <i>FilePattern</i>.</p>

ErrorLevel

ErrorLevel is set to the number of files that failed to be changed (if any) or 0 otherwise.

Remarks

Under Windows 95/98/ME, changing the timestamp of folders is not supported. Attempts to do so are ignored.

A file's last access time might not be as precise on FAT16 & FAT32 volumes as it is on NTFS volumes.

The elements of the YYYYMMDDHH24MISS format are:

YYYY	The 4-digit year
MM	The 2-digit month (01-12)
DD	The 2-digit day of the month (01-31)
HH24	The 2-digit hour in 24-hour format (00-23) e.g. 09 is 9am, 21 is 9pm
MI	The 2-digit minutes (00-59)
SS	The 2-digit seconds (00-59)

If only a partial string is given for YYYYMMDDHH24MISS (e.g. 200403), any remaining element that has been omitted will be supplied with the following default values:

MM: Month 01

DD: Day 01

HH24: Hour 00

MI: Minute 00

SS: Second 00

The built-in variable **A_Now** contains the current local time in the above format.

Note: Date-time values can be compared, added to, or subtracted from via [EnvAdd](#) and [EnvSub](#). Also, it is best to not use greater-than or less-than to compare times unless they are both the same string length. This is because they would be compared as numbers, e.g. 20040201 is always numerically less (but chronologically greater) than 200401010533. So instead use [EnvSub](#) to find out whether the amount of time between them is positive or negative.

Related

Example

; Set the modification time to the current time for all matching files:
FileSetTime, , /tmp/*.txt

; Set the modification date (time will be midnight):
FileSetTime, 20040122, %A_Home%/Documents/test.doc

; Set the creation date. The time will be set to 4:55pm:
FileSetTime, 200401221655, %A_Home%/Documents/test.doc, C

; Change the mod-date of all files that match a pattern.
; Any matching folders will also be changed due to the last param:
FileSetTime, 20040122165500, /tmp/*.*., M, 1

IfExist / IfNotExist

Checks for the existence of a file or directory.

IfExist, FilePattern
IfNotExist, FilePattern

Parameters

FilePattern	The path, filename, or file pattern to check. <i>FilePattern</i> is assumed to be in %A_WorkingDir% if an absolute path isn't specified.
-------------	--

Remarks

None

Related

[Blocks](#), [Else](#), [File-loops](#)

Examples

```
IfExist, /mnt/drive
    MsgBox, The drive exists.
IfExist, %A_Home%/Documents/*.txt
    MsgBox, At least one .txt file exists.
IfNotExist, /tmp/FlagFile.txt
    MsgBox, The target file does not exist.
```

#IniDelete

Deletes a value from a standard format .ini file.

IniDelete, Filename, Section [, Key]

Parameters

Filename	The name of the .ini file, which is assumed to be in %A_WorkingDir% if an absolute path isn't specified.
Section	The section name in the .ini file, which is the heading phrase that appears in square brackets (do not include the brackets in this parameter).
Key	The key name in the in the .ini file. If omitted (in v1.0.10+), the entire Section will be deleted.

ErrorLevel

[ErrorLevel](#) is set to 1 if there was a problem or 0 otherwise.

Remarks

A standard ini file looks like:

```
[SectionName]  
Key=Value
```

Related

[IniRead](#), [IniWrite](#), [RegDelete](#)

Example

```
IniDelete, /tmp/myfile.ini, section2, key
```

IniRead

Reads a value from a standard format .ini file.

IniRead, OutputVar, Filename, Section, Key [, Default]

Parameters

OutputVar	The name of the variable in which to store the retrieved value. If the value cannot be retrieved, the variable is set to the value indicated by the <i>Default</i> parameter (described below).
Filename	The name of the .ini file, which is assumed to be in %A_WorkingDir% if an absolute path isn't specified.
Section	The section name in the .ini file, which is the heading phrase that appears in square brackets (do not include the brackets in this parameter).
Key	The key name in the in the .ini file.
Default	The value to store in <i>OutputVar</i> if the requested key is not found. If omitted, it defaults to the word ERROR.

ErrorLevel

Remarks

A standard ini file looks like:

```
[SectionName]  
Key=Value
```

Related

[IniDelete](#), [IniWrite](#), [RegRead](#)

Example

```
IniRead, OutputVar, /tmp/myfile.ini, section2, key  
MsgBox, The value is %OutputVar%.
```

IniWrite

Writes a value to a standard format .ini file.

IniWrite, Value, Filename, Section, Key

Parameters

Value	The string or number that will be written to the right of Key's equals sign.
Filename	The name of the .ini file, which is assumed to be in %A_WorkingDir% if an absolute path isn't specified.
Section	The section name in the .ini file, which is the heading phrase that appears in square brackets (do not include the brackets in this parameter).
Key	The key name in the in the .ini file.

ErrorLevel

[ErrorLevel](#) is set to 1 if there was a problem or 0 otherwise.

Remarks

A standard ini file looks like:

```
[SectionName]  
Key=Value
```

Related

[IniDelete](#), [IniRead](#), [RegWrite](#)

Example

```
IniWrite, this is a new value, /tmp/myfile.ini, section2, key
```

Loop (files & folders)

Retrieves the specified files or folders, one at a time.

```
Loop, FilePattern [, IncludeFolders?, Recurse?]
```

Parameters

FilePattern	The name of a single file or folder, or a wildcard pattern such as /tmp/*.tmp. <i>FilePattern</i> is assumed to be in %A_WorkingDir% if an absolute path isn't specified. Both asterisks and question marks are supported as wildcards. If this parameter is a single file or folder (i.e. no wildcards) and <i>Recurse</i> is set to 1, more than one match will be found if the specified file name appears in more than one of the folders being searched.
-------------	---

IncludeFolders?	0 (default) Folders are not retrieved (only files). 1 All files and folders that match the wildcard pattern are retrieved. 2 Only folders are retrieved (no files).
Recurse?	0 (default) Subfolders are not recursed into. 1 Subfolders are recursed into so that files and folders contained therein are retrieved if they match <i>FilePattern</i> . All subfolders will be recursed into, not just those whose names match <i>FilePattern</i> .

Remarks

A file-loop is useful when you want to operate on a collection of files and/or folders, one at a time.

All matching files are retrieved, ~~including hidden files~~ (*unsure*). By contrast, OS features such as the `DIR /s` command omit hidden files by default.

Files on an NTFS file system are probably always retrieved in alphabetical order. Files on other file systems are retrieved in no particular order. To ensure a particular ordering, use the `Sort` command as shown in the Examples below.

The following variables exist within any file-loop. If an inner file-loop is enclosed by an outer file-loop, the innermost loop's file will take precedence:

A_LoopFileName	The name of the file or folder currently retrieved (without the path).
A_LoopFilePath	The full path and name of the file/folder currently retrieved. However, if <i>FilePattern</i> contains a relative path rather than an absolute path, the path here will also be relative.
A_LoopFileShortName	The 8.3 short name, or alternate name of the file. If the file doesn't have one (due to the long name being shorter than 8.3 or perhaps because short name generation is disabled on the NTFS file system), <i>A_LoopFileName</i> will be retrieved instead.
A_LoopFileDir	The full path of the directory in which <i>A_LoopFileName</i> resides. However, if <i>FilePattern</i> contains a relative path rather than an absolute path, the path here will also be relative.
A_LoopFileTimeModified	The time the file was last modified. Format <code>YYYYMMDDHH24MISS</code> .
A_LoopFileTimeCreated	The time the file was created. Format <code>YYYYMMDDHH24MISS</code>.
A_LoopFileTimeAccessed	The time the file was last accessed. Format <code>YYYYMMDDHH24MISS</code> .
A_LoopFileAttrib	The attributes of the file currently retrieved.
A_LoopFileSize	The size in bytes of the file currently retrieved. Files larger than 4 gigabytes are supported.
A_LoopFileSizeKB	The size in Kbytes of the file currently retrieved.
A_LoopFileSizeMB	The size in Mbytes of the file currently retrieved.

See [Loop](#) for information about `Blocks`, `Break`, `Continue`, and the *A_Index* variable (which exists in every type of loop).

Related

[Loop](#), [Break](#), [Continue](#), [Blocks](#), [FileSetAttrib](#), [FileSetTime](#)

Examples

```
; Example #1:  
Loop, %ProgramFiles%\*.txt, , 1 ; Recurse into subfolders.  
{  
    MsgBox, 4, , Filename = %A_LoopFilePath%`n`nContinue?  
    IfMsgBox, No  
        break  
}
```

```
; Example #2: Calculate the size of a folder, including the files in all its subfolders:  
SetBatchLines, -1 ; Make the operation run at maximum speed.  
FolderSizeKB = 0  
FileSelectFolder, WhichFolder  
Loop, %WhichFolder%\*.*,, 1  
    FolderSizeKB += %A_LoopFileSizeKB%  
MsgBox Size of %WhichFolder% is %FolderSizeKB% KB.
```

```
; Example #3: Retrieve file names sorted by name (see next example to sort by date):  
FileList = ; Initialize to be blank.  
Loop, C:\*.*  
    FileList = %FileList%%A_LoopFileName%`n  
Sort, FileList, R ; The R option sorts in reverse order. See Sort for other options.  
Loop, parse, FileList, `n  
{  
    if A_LoopField = ; Ignore the blank item at the end of the list.  
        continue  
    MsgBox, 4,, File number %A_Index% is %A_LoopField%. Continue?  
    IfMsgBox, No  
        break  
}
```

```
; Example #4: Retrieve file names sorted by modification date:  
FileList =
```

```
Loop, %UserProfile%\Recent\*.* , 1 ; UserProfile exists as an environment variable on some OSes.
    fileList = %fileList%`A_LoopFileName%`n
Sort, fileList ; Sort by date.
Loop, parse, fileList, `n
{
    if A_LoopField = ; Omit the last linefeed (blank item) at the end of the list.
        continue
    StringSplit, fileItem, A_LoopField, %A_Tab% ; Split into two parts at the tab char.
    MsgBox, 4,, The next file (modified at %fileItem1%) is:`n%fileItem2%`n`nContinue?
    IfMsgBox, No
        break
}
```

```
; Example #5: Copy only the source files that are newer than their counterparts
; in the destination:
CopyIfNewer:
; Caller has set the variables CopySourcePattern and CopyDest for us.
Loop, %CopySourcePattern%
{
    copy_it = n
    IfNotExist, %CopyDest%\%A_LoopFileName% ; Always copy if target file doesn't yet exist.
        copy_it = y
    else
    {
        FileGetTime, time, %CopyDest%\%A_LoopFileName%
        EnvSub, time, %A_LoopFileTimeModified%, seconds ; Subtract the source file's time from the destination's.
        if time < 0 ; Source file is newer than destination file.
            copy_it = y
    }
    if copy_it = y
    {
        FileCopy, %A_LoopFilePath%, %CopyDest%\%A_LoopFileName%, 1 ; Copy with overwrite=yes
        if ErrorLevel <> 0
            MsgBox, Could not copy "%A_LoopFilePath%" to "%CopyDest%\%A_LoopFileName%".
    }
}
Return
```

Loop (read file contents)

Retrieves the lines in a text file, one by one (performs better than [FileReadLine](#)).

Loop, Read, InputFile [, OutputFile, FutureUse]

Parameters

Read	This parameter must be the word READ.
InputFile	The name of the file whose contents will be read by the loop, which is assumed to be in <a>%A_WorkingDir% if an absolute path isn't specified.
OutputFile	(Optional) The name of the file to be kept open for the duration of the loop, which is assumed to be in <a>%A_WorkingDir% if an absolute path isn't specified. Within the loop's body, use the <a>FileAppend command with only one parameter (the text to be written) to append to this special file. Appending to a file in this manner performs better than using <a>FileAppend in its 2-parameter mode because the file does not need to be closed and re-opened for each operation. Remember to include a newline ('`n') after the text, if desired. To append in binary mode rather than text mode, prepend an asterisk to the filename. This causes each newline character ('`n') to be written as a single linefeed (LF) rather than the Windows standard of CR+LF. Example: *C:\My Unix File.txt.
FutureUse	This parameter should be always left blank (omitted).

Remarks

A file-read loop is useful when you want to operate each line contained in a text file, one at a time. It performs better than using FileReadLine because: 1) the file can be kept open for the entire operation; and 2) the file does not have to be re-scanned each time to find the requested line number.

The built-in variable **A_LoopReadLine** exists within any file-read loop. It contains the contents of the current line in the file. If an inner file-read loop is enclosed by an outer file-read loop, the innermost loop's file-line will take precedence.

Lines up to 65,534 characters long can be read. If the length of a line exceeds this, its remaining characters will be read during the next loop iteration.

StringSplit or a parsing loop is often used inside a file-read loop to parse the contents of each line retrieved from *InputFile*. For example, if *InputFile*'s lines are each a series of tab-delimited fields, those fields can individually retrieved as in this example:

```
Loop, read, Database Export.txt
{
    Loop, parse, A_LoopReadLine, %A_Tab%
    {
        MsgBox, Field number %a_index% is %A_LoopField%.
    }
}
```

See Loop for information about Blocks, Break, Continue, and the **A_Index** variable (which exists in every type of loop).

Related

[FileReadLine](#), [FileAppend](#), [Loop](#), [Break](#), [Continue](#), [Blocks](#), [FileSetAttrib](#), [FileSetTime](#)

Examples

```
; Example #1: Only those lines of the 1st file that contain  
; the word FAMILY will be written to the 2nd file. Uncomment  
; this line to overwrite rather than append to any existing file:  
;FileDelete, %A_Home%/Documents/Family Addresses.txt  
  
Loop, read, %A_Home%/Documents/Address List.txt, %A_Home%/Documents/Family Addresses.txt  
{  
    IfInString, A_LoopReadLine, family, FileAppend, %A_LoopReadLine%`n  
}
```

```
; Example #2: A working script that attempts to extract all FTP and HTTP  
; URLs from a text or HTML file:  
FileSelectFile, SourceFile, 3,, Pick a text or HTML file to analyze.  
if SourceFile =  
    return ; This will exit in this case.  
  
SplitPath, SourceFile,, SourceFilePath,, SourceFileNoExt  
DestFile = %SourceFilePath%\%SourceFileNoExt% Extracted Links.txt  
  
IfExist, %DestFile%  
{  
    MsgBox, 4,, Overwrite the existing links file? Press No to append to it.`n`nFILE: %DestFile%  
    IfMsgBox, Yes  
        FileDelete, %DestFile%  
}  
  
LinkCount = 0  
Loop, read, %SourceFile%, %DestFile%  
{  
    URLSearchString = %A_LoopReadLine%  
    Gosub, URLSearch  
}  
MsgBox %LinkCount% links were found and written to "%DestFile%".  
return  
  
URLSearch:  
; It's done this particular way because some URLs have other URLs embedded inside them:
```

```
StringGetPos, URLStart1, URLSearchString, http://
StringGetPos, URLStart2, URLSearchString, ftp://
StringGetPos, URLStart3, URLSearchString, www.

; Find the left-most starting position:
URLStart = %URLStart1% ; Set starting default.
Loop
{
    ; It helps performance (at least in a script with many variables) to resolve
    ; "URLStart%A_Index%" only once:
    StringTrimLeft, ArrayElement, URLStart%A_Index%, 0
    if ArrayElement = ; End of the array has been reached.
        break
    if ArrayElement = -1 ; This element is disqualified.
        continue
    if URLStart = -1
        URLStart = %ArrayElement%
    else ; URLStart has a valid position in it, so compare it with ArrayElement.
    {
        if ArrayElement <> -1
            if ArrayElement < %URLStart%
                URLStart = %ArrayElement%
    }
}

if URLStart = -1 ; No URLs exist in URLSearchString.
    return

; Otherwise, extract this URL:
StringTrimLeft, URL, URLSearchString, %URLStart% ; Omit the beginning/irrelevant part.
Loop, parse, URL, %A_Tab%%A_Space%> ; Find the first space, tab, or angle (if any).
{
    URL = %A_LoopField%
    break ; i.e. perform only one loop iteration to fetch the first "field".
}
; If the above loop had zero iterations because there were no ending characters found,
; leave the contents of the URL var untouched.

; If the URL ends in a double quote, remove it. For now, StringReplace is used, but
; note that it seems that double quotes can legitimately exist inside URLs, so this
; might damage them:
StringReplace, URLCleansed, URL, "", All
FileAppend, %URLCleansed%`n
LinkCount += 1

; See if there are any other URLs in this line:
StringLen, CharactersToOmit, URL
CharactersToOmit += %URLStart%
StringTrimLeft, URLSearchString, URLSearchString, %CharactersToOmit%
```

```
Gosub, URLSearch ; Recursive call to self.  
return
```

SetWorkingDir [v1.0.11+]

Changes the script's current working directory.

SetWorkingDir, DirName

Parameters

DirName

The name of the new working directory, which is assumed to be a subfolder of the current %A_WorkingDir% if an absolute path isn't specified.

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

Remarks

The script's working directory is the default directory that is used to access files and folders when an absolute path has not been specified. In the following example, the file *My Filename.txt* is assumed to be in %A_WorkingDir%: [FileAppend](#), A Line of Text, My Filename.txt

A script's initial working directory is determined by how it was launched. For example, if it was run via shortcut -- such as on the Start Menu -- its working directory is determined by the "Start in" field within the shortcut's properties.

Once changed, the new working directory is instantly and globally in effect throughout the script. All interrupted, paused, and newly launched [threads](#) are affected, including [Timers](#).

Related

[%A_WorkingDir%](#), [FileSelectFolder](#)

Example

SplitPath

Separates a file name into its name, directory, extension, and drive.

```
SplitPath, InputVar [, OutFileName, OutDir, OutExtension, OutNameNoExt, OutDrive]
```

Parameters

InputVar	Name of the variable containing the file name to be analyzed.
OutFileName	Name of the variable in which to store the file name without its path. The file's extension is included.
OutDir	Name of the variable in which to store the directory of the file, including drive letter or share name (if present). The final backslash is not included even if the file is located in a drive's root directory.
OutExtension	Name of the variable in which to store the file's extension (e.g. TXT, DOC, or EXE). The dot is not included.
OutNameNoExt	Name of the variable in which to store the file name without its path, dot and extension.
OutDrive	Name of the variable in which to store the drive letter or server name of the file. If the file is on a local or mapped drive, the variable will be set to the drive letter followed by a colon. If the file is on a network path (UNC), the variable will be set to the share name, e.g. \\Workstation01

Remarks

Any of the output variables may be omitted if the corresponding information is not needed.

If *InputVar* contains a filename with no path or just a relative path (e.g. Address List.txt), *OutDrive* will be made blank but all the other output variables will be set correctly. Similarly, if there is no path present, *OutDir* will be made blank; and if there is a path but no file name present, *OutFileName* and *OutNameNoExt* will be made blank.

Actual files and directories in the file system are not checked by this command. It simply analyzes the string given in *InputVar*.

Wildcards (*) and other characters illegal in filenames are treated the same as legal characters, with the exception of colon, backslash *slash*, and period (dot), which are processed according to their nature in delimiting the drive letter, directory, and extension of the file.

Related

Example

```
FullName = %A_Home%/My Documents/Address List.txt  
  
; To fetch only the bare filename:  
SplitPath, FullName, name  
  
; Or, to fetch all info:  
SplitPath, FullName, name, dir, ext, name_no_ext, drive  
  
; The above will set the variables as follows:  
; name = Address List.txt  
; dir = /home/your-user-name/My Documents  
; ext = txt  
; name_no_ext = Address List  
; drive = C:
```

#Include / #IncludeAgain

Causes the script to behave as though the specified file's contents are present.

```
#Include FileName  
#IncludeAgain FileName
```

Parameters

FileName	The name of the file to be included, which is assumed to be in %A_WorkingDir% if an absolute path isn't specified. It must not contain double quotes, variable references (e.g. %ProgramFiles%), wildcards, or escape sequences (which are not needed since all characters are treated literally).
----------	--

Remarks

#Include ensures that *FileName* is included only once, even if multiple inclusions are encountered for it. By contrast, #IncludeAgain allows multiple inclusions of the same file, while being the same as #Include in all other respects.
Lines displayed in the main window via [ListLines](#) or the menu View->Lines are always numbered according to their physical order within their own files. In other words, including a new file will not change the line numbering of the main script file.
As with other # directives, this one cannot be executed conditionally. In other words, this example would not work:

```
if x = 1  
    #Include SomeFile.ahk ; This is invalid.
```

Related

None

Example

```
#Include C:\My Documents\Scripts\Utility Subroutines.ahk ; Don't enclose in double quotes.
```

#{} (block)

Denotes a block. Blocks are typically used with [Else](#), [Loop](#), and [IF](#)-commands.

```
{  
zero or more commands  
}
```

Remarks

A block is used to bind two or more commands together. It can also be used to change which IF an [ELSE](#) belongs to, as in this example where the block forces the [ELSE](#) to belong to the first IF rather than the second:

```
if var1 = 1  
{  
    if var2 = abc  
        sleep, 1  
}  
else  
    return
```

Although blocks can be used anywhere, currently they are only meaningful when used with [Else](#), [Loop](#), or an IF-type command such as [IfEqual](#) or [IfWinExist](#).

If an [IF](#), [ELSE](#), or [Loop](#) has only a single command, that command need not be enclosed in a block. However, there may be cases where doing so enhances the readability or maintainability of the script.

Related

[Loop](#), [Else](#), [If](#)

Example

```
if x = 1
{
    MsgBox, test1
    Sleep, 5
}
else
    MsgBox, test2
```

Break

Exits (terminates) a [loop](#). Valid only inside a [loop](#).

Break

Parameters

None.

Remarks

The break command applies to the innermost loop in which it is enclosed. The use of break and [continue](#) are encouraged over [goto](#) since they usually make scripts more readable and maintainable.

Related

Example

```
loop
{
    ...
    if var > 25
        break
    ...
    if var <= 5
        continue
}
```

Continue

Skips the rest of the current [loop](#) iteration and begins a new one. Valid only inside a [loop](#).

Continue

Parameters

None.

Remarks

The continue command applies to the innermost loop in which it is enclosed. The use of [break](#) and [continue](#) are encouraged over [goto](#) since they usually make scripts more readable and maintainable.

Related

[Loop](#), [Break](#), [Blocks](#)

Example

```
; This example displays 5 MsgBoxes, one for each number between 6 and 10.  
; Note that in the first 20 iterations of the Loop, the "continue" command  
; causes the loop to start over before it reaches the MsgBox line.  
Loop, 10  
{  
    if A_Index <= 5  
        continue  
    MsgBox, %A_Index%  
}
```

Else

Specifies the command(s) to perform if an IF-command evaluates to FALSE.

Else

Remarks

Every use of this command must "belong to" (be associated with) an IF-command above it. An ELSE always belongs to the nearest unclaimed IF-command above it unless a [block](#) is used to change that behavior. An ELSE can be followed immediately by any other single command on the same line. This is most often used for "else if" ladders (see examples).

Related

See [Blocks](#). Also, every IF-command can use ELSE, including [IfWinActive](#), [IfWinExist](#), [IfMsgBox](#), [IfInString](#), [IfBetween](#), [IfIn](#), and the [IF command](#) itself.

Examples

```
IfWinExist, Untitled - Notepad  
{  
    WinActivate  
    Sleep, 1  
}  
else
```

```
{  
    WinActivate, Some Other Window  
    Sleep, 1  
}  
  
if x = 1  
    Gosub, a1  
else if x = 2 ; "else if" style  
    Gosub, a2  
else IfEqual, x, 3 ; alternate style  
{  
    Gosub, a3  
    Sleep, 1  
}  
else Gosub, a4 ; i.e. Any single command can be on the same line with an ELSE.  
  
;Also OK:  
IfEqual, y, 1, Gosub, b1  
else {  
    Sleep, 1  
    Gosub, b2  
}
```

Exit

Exits the [current thread](#) or (if the script is not [persistent](#) contains no hotkeys) the entire script.

Exit [, ExitCode]

Parameters

ExitCode	An integer (i.e. negative, positive, or zero) that is returned to its caller when the script exits. This code is accessible to any program that spawned the script, such as another script (via RunWait) or a batch (.bat) file. If omitted, ExitCode defaults to zero. Zero is traditionally used to indicate success. Note: Windows 95 may be limited in how large ExitCode can be.
----------	---

Remarks

If the script has no hotkeys, isn't [persistent](#), and hasn't requested the Num/Scroll/CapsLock key(s) to be kept AlwaysOn or AlwaysOff, it will terminate immediately when Exit is encountered (except if it has an [OnExit](#) subroutine).

Otherwise, the Exit command terminates the [current thread](#). In other words, the stack of subroutines called directly or indirectly by a [menu](#), [timer](#), or [hotkey](#) subroutine will all be returned from as though a [Return](#) were immediately encountered in each. If used directly inside such a subroutine -- rather than in one of the subroutines called indirectly by it -- Exit is equivalent to [Return](#).

Use [ExitApp](#) to completely terminate a script that is [persistent](#) or contains hotkeys.

Related

[ExitApp](#), [OnExit](#), [Gosub](#), [Return](#), [Threads](#), [#Persistent](#)

Example

```
#z::  
Gosub, Sub2  
MsgBox, This msgbox will never happen because of the EXIT.  
return  
  
Sub2:  
Exit ; Terminate this subroutine as well as the calling subroutine.
```

ExitApp

Terminates the script unconditionally.

ExitApp [, ExitCode]

Parameters

ExitCode	An integer (i.e. negative, positive, or zero) that is returned to its caller when the script exits. This code is accessible to any program that spawned the script, such as another script (via RunWait) or a batch (.bat) file. If omitted, ExitCode defaults to zero. Zero is traditionally used to indicate success. Note: Windows 95 may be limited in how large ExitCode can be.
----------	---

Remarks

The script is immediately terminated unless it has an [OnExit](#) subroutine. This is equivalent to choosing "Exit" from the script's tray menu or main menu.

[Exit](#) and [ExitApp](#) behave identically when the script contains no hotkeys, is not [persistent](#), and does not ask for the Num/Scroll/Capslock key(s) to be kept AlwaysOn or AlwaysOff.

If the script has an [OnExit](#) subroutine, it will be run in response to [ExitApp](#).

Related

[Exit](#), [OnExit](#), [#Persistent](#)

Example

```
#x::ExitApp ; Assign a hotkey to terminate this script.
```

Gosub

Jumps to the specified label and continues execution until [Return](#) is encountered.

Gosub, Label

Parameters

Label	The name of the label or hotkey label to which to jump, which causes the commands beneath <i>Label</i> to be executed until a Return or Exit is encountered. When that happens, the script jumps back to the first command beneath the Gosub and resumes execution there.
-------	---

The name of the label or [hotkey label](#) to which to jump, which causes the commands beneath *Label* to be executed until a [Return](#) or [Exit](#) is encountered. When that happens, the script jumps back to the first command beneath the Gosub and resumes execution there.

Remarks

As with the parameters of almost all other commands, *Label* can be a [variable](#) reference such as `%MyLabel%`, in which case the name stored in the variable is used as the target. However, performance might suffer slightly because the the target label must be "looked up" every time it is encountered rather than only once when the script is first loaded.

Related

[Return](#), [Blocks](#), [Loop](#), [Goto](#)

Example

Gosub, Label1

MsgBox, The Label1 subroutine has returned (it is finished).

return

Label1:

MsgBox, The Label1 subroutine is now running.

return

Goto

Jumps to the specified label and continues execution.

Goto, Label

Parameters

Label	The name of the label or hotkey label to which to jump.
-------	---

Remarks

The use of Goto is discouraged because it generally makes scripts less readable and harder to maintain. Consider using [Else](#), [Blocks](#), [Break](#), and [Continue](#) as substitutes for Goto.

Related

Example

```
Goto, MyLabel  
...  
MyLabel:  
Sleep, 100  
...
```

Loop (normal)

Perform a series of command repeatedly: either the specified number of times or until [break](#) is encountered.

Loop [, Count]

Parameters

Count	How many times (iterations) to perform the loop. If omitted or blank, the Loop is infinite (i.e. until a break or return is encountered). Unlike Repeat...EndRepeat, if Count is 0, zero iterations will be performed rather than an infinite number.
-------	---

Remarks

The loop command is usually followed by a [block](#), which is a collection of statements that form the *body* of the loop. However, a loop with only a single command does not require a block (an if-else compound statement counts as a single command for this purpose).

The simplest use of this command is an infinite loop that uses the [break](#) command somewhere in the loop's *body* to determine when to stop the loop.

The use of [break](#) and [continue](#) inside a loop are encouraged as alternatives to [goto](#), since they generally make a script more understandable and maintainable. To create a "While" loop, make the first statement of the loop's *body* an IF statement that conditionally issues the [break](#) command. To create a "Do...While" loop, use the same technique except put the IF statement as the last statement in the loop's *body*.

The built-in variable **A_Index** contains the number of the current loop iteration. For example, the first time the script executes the loop's *body*, this variable will contain the number 1. For the second time, it will contain 2, and so on. If an inner loop is enclosed by an outer loop, the inner loop takes precedence. The value will be **0** whenever the variable is referenced outside of a loop. This variable works inside all types of loops, including [file-loops](#) and [registry-loops](#).

Specialized loops:

Loops can be used to automatically retrieve files, folders, or registry items (one at a time). See [file-loop](#) and [registry-loop](#) for details. In addition, [file-read loops](#) can operate on the entire contents of a file, one line at a time, and [parsing loops](#) can operate on the delimited substrings contained within a string, one field at a time.

Related

[File loop](#), [Registry loop](#), [File-read loop](#), [Parsing loop](#), [Break](#), [Continue](#), [Blocks](#)

Examples

```
Loop, 3
{
    MsgBox, Iteration number is %A_Index%. ; A_Index will be 1, 2, then 3
    Sleep, 100
}

Loop
{
    if a_index > 25
        break ; Terminate the loop
    if a_index < 20
        continue ; Skip the below and start a new iteration
    MsgBox, a_index = %a_index% ; This will display only the numbers 20 through 25
}
```

#Loop (parse a string)

Retrieves substrings (fields) from a string, one by one.

Loop, Parse, InputVar [, Delimiters, OmitChars, FutureUse]

Parameters

se	This parameter must be the word PARSE, and unlike other loop types, it must not be a variable reference that resolves to the word PARSE.
utVar	The name of the variable whose contents will be analyzed. Do not enclose the name in percent signs unless you want the <i>contents</i> of the variable to be used as the name.
imiters	<p>If this parameter is blank or omitted, each character of <i>InputVar</i> will be treated as a separate substring.</p> <p>In v1.0.09+, if this parameter is CSV, InputVar will be parsed in standard comma separated value format. Here is an example of a CSV line produced by MS Excel: "first field",SecondField,"the word ""special"" is quoted literally",,"last field, has literal comma"</p> <p>Otherwise, <i>Delimiters</i> contains one or more characters (case sensitive) used to determine where the boundaries between substrings occur in <i>InputVar</i>. Delimiter characters are not considered to be part of the substrings themselves. In addition, if there is nothing between a pair of delimiters within <i>InputVar</i>, the corresponding substring will be empty. For example: ` (an escaped comma) would divide the string based on every occurrence of a comma. Similarly, %A_Tab%%A_Space% would start a new substring every time a space or tab is encountered in <i>InputVar</i>. To use a string as a delimiter rather than a character, first use StringReplace to replace all occurrences of the string with a single character that is never used literally in the text, e.g. one of these special characters: ¢¤¥¦§©ª«®µ¶. Consider this example, which uses the string
 as a delimiter: <pre>StringReplace, NewHTML, HTMLString,
, ¢, All ; Replace each
 with a cent symbol. Loop, parse, NewHTML, ¢ ; Parse the string based on the cent symbol.</pre> <pre>{ ... }</pre> </p>
itChars	<p>An optional list of characters (case sensitive) to exclude from the beginning and end of each substring. For example, if <i>OmitChars</i> is %A_Space%%A_Tab%, spaces and tabs will be removed from the beginning and end (but not the middle) of every retrieved substring.</p> <p>If <i>Delimiters</i> is blank, <i>OmitChars</i> indicates which characters should be excluded from consideration (the loop will not see them).</p>
ureUse	This parameter should be always left blank (omitted).

marks

String parsing loops are useful when you want to operate on each field contained in a string, one at a time. Parsing loops use less memory than `StringSplit` (since `StringSplit` creates a permanent array) and in most cases they are easier to use.

The built-in variable **A LoopField** exists within any parsing loop. It contains the contents of the current substring (field) from *InputVar*. If an inner parsing loop is enclosed by an outer parsing loop, the innermost loop's field will take precedence.

There is no restriction on the size of *InputVar* or its fields. In addition, if *InputVar*'s contents change during the execution of the loop, the loop will not "see" the changes because it is operating on a temporary copy of the original content.

Arrange the fields in a different order prior to parsing, use the [Sort](#) command.

[Learn](#) for information about `Blocks`, `Break`, `Continue`, and the `A_Index` variable (which exists in every type of loop).

Related

[StringSplit](#), [Loop](#), [Break](#), [Continue](#), [Blocks](#), [Sort](#), [FileSetAttrib](#), [FileSetTime](#)

Examples

```
; Example #1:  
Colors = red,green,blue  
Loop, parse, Colors, `,  
{  
    MsgBox, Color number %a_index% is %A_LoopField%.  
}
```

```
; Example #2: This one is useful whenever the clipboard contains  
; files (such as those copied from an open Explorer window).  
; The program automatically converts such files to their file names.  
  
Loop, parse, clipboard, `n, `r  
{  
    MsgBox, 4, , File number %a_index% is %A_LoopField%.`n`nContinue?  
    IfMsgBox, No, break  
}
```

```
; Example #3: Parse a comma separated value (CSV) file:  
Loop, read, C:\Database Export.csv  
{  
    LineNumber = %A_Index%  
    Loop, parse, A_LoopReadLine, CSV  
    {  
        MsgBox, 4, , Field %LineNumber%-%A_Index% is:`n%A_LoopField%`n`nContinue?  
        IfMsgBox, NO, return  
    }  
}
```

Retrieves the contents of the specified registry subkey, one item at a time.

#OnExit [v1.0.13+]

Specifies a subroutine to run when the script exits.

OnExit [, Label, FutureUse]

Parameters

Label	If omitted, the script is returned to its normal exit behavior. Otherwise, this parameter is the label name whose contents will be executed (as a new thread) when the script exits by any means.
-------	---

FutureUse	The parameter should always be omitted.
-----------	---

Remarks

Since the specified subroutine is called instead of terminating the script, **that subroutine must use the ExitApp command if termination is desired**. Alternatively (as in the Examples section below), the OnExit subroutine might display a MsgBox to prompt the user for confirmation — and only if the user presses YES will the script be terminated.

The OnExit subroutine is called when the script exits by any means (except when it is killed by something like "End Task"). It is also called whenever the #SingleInstance and Reload commands ask a previous instance to terminate.

The OnExit thread does not obey #MaxThreads (it will always launch when needed). In addition, while it is running, it cannot be interrupted by hotkeys, custom menu items, or timed subroutines. However, it will be interrupted (and the script will terminate) if the user chooses Exit from the tray menu or main menu, or the script is asked to terminate as a result of Reload or #SingleInstance. Because of this, the OnExit subroutine should be designed to finish quickly unless the user is aware of what it is doing.

If the OnExit thread encounters a failure condition such as a runtime error, the script will terminate. This prevents a flawed OnExit subroutine from making a script impossible to terminate.

If the OnExit subroutine was launched due to an Exit or ExitApp command that specified an exit code, that code is ignored and no longer available. A new exit code can be specified by the OnExit subroutine if/when it calls ExitApp.

Whenever the OnExit subroutine is called by an exit attempt, it starts off fresh with the default values for settings such as SetKeyDelay. These defaults can be changed in the auto execute section.

The built-in variable **A_ExitReason** is blank unless the OnExit subroutine is currently running or has been called at least once by a prior exit attempt. If not blank, it is one of the following words:

Logoff

The user is logging off.

Shutdown	The system is being shut down or restarted, such as by the Shutdown command.
Close	The script was sent a WM_CLOSE or WM_QUIT message, had a critical error, or is being closed in some other way. Although all of these are unusual, WM_CLOSE might be caused by WinClose having been used on the script's main window. To prevent this, dismiss the main window with Send, !{F4}.
Error	A runtime error occurred in a script that has no hotkeys and that is not persistent. An example of a runtime error is Run/RunWait being unable to launch the specified item.
Menu	The user selected Exit from the main window's menu or from the standard tray menu.
Exit	The Exit or ExitApp command was used (includes custom menu items).
Reload	The script is being reloaded via the Reload command or menu item.
Single	The script is being replaced by a new instance of itself as a result of #SingleInstance.

Related

[ExitApp](#), [Shutdown](#), [#Persistent](#), [Threads](#), [Gosub](#), [Return](#), [Menu](#)

Example

```
#Persistent, For demonstration purposes.
OnExit, ExitSub
return

ExitSub:
if A_ExitReason not in Logoff,Shutdown, Avoid spaces around comma.
+
    MsgBox, 4, , Are you sure you want to exit?
    IfMsgBox, No, return
+
ExitApp
```

Pause

Pauses the script's current thread.

Pause [, On|Off|Toggle]

Parameters

On|Off|Toggle

On: Pauses the [current thread](#).

Off: Unpauses ~~the most recently~~ [the underlying](#) paused [thread](#).

Toggle (default): Changes to the opposite of its previous state (On or Off).

Remarks

Unlike [Suspend](#) -- which disables hotkeys -- pause will freeze the [current thread](#) (and as a result, all interrupted threads beneath it will lie dormant).

New hotkeys can still be launched while the script is paused, but when their [threads](#) finish, the underlying interrupted thread will still be paused. In other words, each thread can be paused independently of the others.

~~The color of the tray icon changes from green to red~~ [The icon changes to a loading symbol](#) whenever the script's [current thread](#) is in a paused state.

This command mirrors the function of the "Pause Script" tray menu item.

~~A script is always halted (though not officially paused) while its tray menu or main window's menu is displayed.~~

Related

[Suspend](#), [ExitApp](#), [Threads](#)

Example

Pause::Pause ; Assign the toggle-pause function to the "pause" key.

#p::Pause ; Or assign it to Win+p or some other hotkey.

Return

Returns from a subroutine to which execution had previously jumped via [Gosub](#), [Hotkey](#) activation, [GroupActivate](#), or other means.

[Return](#)

Parameters

None

Remarks

Due to its hotkey nature, if there is no caller to which to return, Return will do an [Exit](#).

Related

[Gosub](#), [Exit](#), [ExitApp](#), [GroupActivate](#)

Example

```
#Z::  
MsgBox, The Win-Z hotkey was pressed.
```

```
Gosub, WinZ  
return
```

```
WinZ:  
Sleep, 1000  
return
```

#SetBatchLines

~~Determines how fast a script will run (affects CPU utilization).~~

~~SetBatchLines, 20ms~~
~~SetBatchLines, LineCount~~

Parameters

20ms	(The 20ms is just an example). If the value ends in ms, it indicates how often the script should sleep (each sleep is 10 ms long). In the following example, the script will sleep for 10ms every time it has run for 20ms. SetBatchLines, 20ms
LineCount	The number of script lines to execute prior to sleeping for 10ms. The value can be as high as 9223372036854775807, or use a value of -1 to never sleep (i.e. run at maximum speed).

Remarks

If this command is not used:

- Default to SetBatchLines 10ms except in versions prior to v1.0.16, which use 10 (lines) instead.

The "ms" method is recommended for scripts whenever speed and cooperation are important. For example, on most systems a setting of 10ms will prevent the script from using any more than 50% of an idle CPU's time. This allows scripts to run quickly while still maintaining a high level of cooperation with CPU sensitive tasks such as games and video capture/playback.

The built-in variable **A_BatchLines** contains the current setting.

The speed of a script can also be impacted by the following commands, depending on the nature of the script: SetWinDelay, SetControlDelay, SetKeyDelay, SetMouseDelay, and SetDefaultMouseSpeed.

Every newly launched hotkey, custom menu item, or timed subroutine starts off fresh with the default setting for this command. That default may be changed by using this command in the auto-execute section (top part) of the script.

Related

[SetWinDelay](#), [SetControlDelay](#), [SetKeyDelay](#), [SetMouseDelay](#), [SetDefaultMouseSpeed](#)

Example

`SetBatchLines, 10ms`

`SetBatchLines, 1000`

SetTimer

Causes a subroutine to be launched automatically and repeatedly at a specified time interval.

`SetTimer, Label [, Period|On|Off, Priority]`

Parameters

Label	The name of the label or hotkey label to which to jump, which causes the commands beneath <i>Label</i> to be executed until a Return or Exit is encountered. As with the parameters of almost all other commands, <i>Label</i> can be a variable reference such as %MyLabel%, in which case the name stored in the variable is used as the target.
Period On Off	On: Re-enables a previously disabled timer at its former <i>period</i> . If the timer doesn't exist, it is created (with a default period of 250). Off: Disables an existing timer. Period: Creates or updates a timer using this parameter as the number of milliseconds that must pass since the last time the <i>Label</i> subroutine was started. When this amount of time has passed, <i>Label</i> will be run again (unless it is still running from the last time). The timer will be automatically enabled. To prevent this, call the command a second time immediately afterward, specifying OFF for this parameter. If blank and: 1) the timer does not exist: it will be created with a period of 250. 2) the timer already exists: it will be enabled and reset at its former <i>period</i> unless a <i>Priority</i> is specified.
Priority	In v1.0.16+, this optional parameter is an integer between -2147483648 and 2147483647 to indicate this timer's thread priority. If omitted, 0 will be used. See Threads for details. To change the priority of an existing timer without affecting it in any other way, leave the parameter before this one blank.

Remarks

Timers are useful because they run asynchronously, meaning that they will run at the specified frequency (interval) even when the script is waiting for a window, displaying a dialog, or busy with another task. Examples of their many uses include taking some action when the user becomes idle (as reflected by [%A_TimeIdle%](#)) or closing unwanted windows the moment they appear.

Although timers may give the illusion that the script is performing more than one task simultaneously, this is not the case. Instead, timed subroutines are treated just like other threads: they can interrupt or be interrupted by another thread, such as a [hotkey subroutine](#). See [Threads](#) for details.

Whenever a timer is enabled, re-enabled, or updated with a new *period*, it will not run right away; its time *period* must expire first. If you wish the timer's first execution to be immediate, use the [Gosub](#) command immediately after the SetTimer command.

Similarly, if SetTimer is used on an existing timer and parameter #2 is a number or the word ON, the timer's internal "time of last run" will be reset to the current time; in other words, the entirety of its period must elapse before it will run again. An easy way to reset a timer is to omit all parameters except the first. Example: [SetTimer](#), *TimerName*

~~Currently, timers cannot run much more often than every 10ms on Windows XP/2000/NT and about 55ms on Windows 9x. Specifying a Period less than this will usually result in an actual interval of 10 or 55 (but this behavior may change in future versions so shouldn't be relied upon).~~

Timers with very short *periods* might not be able to run as often as expected if the current setting of [SetBatchLines](#) restricts the speed of the script too much. Similarly, if the total number of lines to be executed in all timed subroutines is large, you may need to alter [SetBatchLines](#) to allow them to run as often as specified.

In addition, a timer might not be able to run as often as specified under the following conditions:

1. Other applications are putting a heavy load on the CPU.
2. The timer subroutine itself takes longer than its own *period* to run, or there are too many other competing timers (altering [SetBatchLines](#) may help).
3. The timer has been interrupted by another [thread](#), namely another timed subroutine, [hotkey subroutine](#), or [custom menu item](#). If this happens and the interrupting thread takes a long time to finish, the interrupted timer will be effectively disabled for the duration. However, any other timers will continue to run by interrupting the [thread](#) that interrupted the first timer.

Because timers operate by temporarily interrupting the script's current activity, their subroutines should be kept short (so that they finish quickly) whenever a long interruption would be undesirable.

Timers that stay in effect for the duration of a script should usually be created in the [auto-execute section](#). By contrast, a temporary timer might often be disabled by its own subroutine (see the examples).

Whenever a timed subroutine is run, it starts off fresh with the default values for settings such as [SetKeyDelay](#). These defaults can be changed in the [auto-execute section](#).

Although timers will operate when the script is [suspended](#), they will not run whenever [any thread](#) is [paused](#). In addition, they do not operate when the user is navigating through the tray icon menu or the main window's menu.

If [hotkey](#) response time is critical (such as in games) and the script contains any timers whose subroutines take longer than about 5 ms to execute, use the following command to avoid any chance of a 15 ms delay. Such a delay would otherwise happen if a hotkey is pressed at the exact moment a timer thread is in its period of uninterruptibility: [Thread, interrupt, 0](#) ; Make all threads always-interruptible.

If a timer is disabled while its subroutine is currently running, that subroutine will continue until it completes.

The [KeyHistory](#) feature shows how many timers exist and how many are currently enabled.

To keep a non-hotkey script running -- such as one that contains only timers -- use [#Persistent](#).

Related

[Gosub](#), [Return](#), [Threads](#), [Thread](#), [Menu](#), [#Persistent](#)

Examples

; Example #1: Close unwanted windows whenever they appear:

```
SetTimer, CloseMailWarnings, 250
return
```

CloseMailWarnings:

```
WinClose, Microsoft Outlook, A timeout occurred while communicating
```

```
WinClose, Microsoft Outlook, A connection to the server could not be established
return
```

; Example #2: Wait for a certain window to appear and then alert the user:

```
SetTimer, Alert1, 500
return
```

Alert1:

```
IfWinNotExist, Video Conversion, Process Complete
```

```
        return
; Otherwise:
SetTimer, Alert1, Off ; i.e. the timer turns itself off here.
SplashTextOn, , , The video conversion is finished.
Sleep, 3000
SplashTextOff
return
```

```
; Example #3: Detection of single, double, and triple-presses of a hotkey. This
; allows a hotkey to perform a different operation depending on how many times
; you press it:
#c::
if winc_presses > 0 ; SetTimer already started, so we log the keypress instead.
{
    winc_presses += 1
    return
}
; Otherwise, this is the first press of a new series. Set count to 1 and start
; the timer:
winc_presses = 1
SetTimer, KeyWinC, 400 ; Wait for more presses within a 400 millisecond window.
return

KeyWinC:
SetTimer, KeyWinC, off
if winc_presses = 1 ; The key was pressed once.
{
    Run, m:\ ; Open a folder.
}
else if winc_presses = 2 ; The key was pressed twice.
{
    Run, m:\multimedia ; Open a different folder.
}
else if winc_presses > 2
{
    MsgBox, Three or more clicks detected.
}
; Regardless of which action above was triggered, reset the count to
; prepare for the next series of presses:
winc_presses = 0
return
```

#Sleep

Waits the specified amount of time before continuing.

Sleep, Delay

Parameters

Delay

The amount of time to pause (in milliseconds) between 0 and 2147483647 (24 days).

Remarks

Due to the granularity of the OS's time keeping system, *Delay* might be rounded up to the nearest multiple of 10. For example, a delay between 1 and 10 (inclusive) is equivalent to 10 on Windows XP (and probably NT & 2k).

The actual delay time may wind up being longer than what was requested if the CPU is under load.

A delay of 0 will yield the remainder of the script's current timeslice to any other processes that need it, in which case an actual delay between 0 and 20ms (or more) may occur. If there are no needy processes, there will be no delay at all. However, a *Delay* of 0 should always wind up being shorter on most OSes than any longer *Delay* would have been.

While sleeping, new [threads](#) can be launched via [hotkey](#), [custom menu item](#), or [timer](#).

Related

[SetKeyDelay](#), [SetMouseDelay](#), [SetControlDelay](#), [SetWinDelay](#), [SetBatchLines](#)

Example

Sleep, 1000 ; 1 second

Suspend

Disables all or selected [hotkeys](#).

Suspend [, Mode]

Parameters

Mode

On: Suspends all hotkeys except those explained the the Remarks section.

Off: Re-enables all hotkeys.

Toggle (default): Changes to the opposite of its previous state (On or Off).

~~Permit: Does nothing except mark the current subroutine as being exempt from suspension.~~

Remarks

Any hotkey subroutine whose first line is Suspend -- ~~except "Suspend, on"~~ ([in v1.0.07+](#)) -- will be exempt from suspension. In other words, the hotkey will remain enabled even while suspension is ON. This allows suspension to be turned off via such a hotkey.

Suspending a script's hotkeys does not stop the script's already-running [threads](#) (if any); use [Pause](#) to do that.

When a script's hotkeys are suspended, ~~its tray icon changes to the letter S. The icon changes to a keyboard symbol.~~

Related

[Pause](#), [ExitApp](#)

Example

`^!s::Suspend ; Assign the toggle-suspend function to a hotkey.`

GUI [v1.0.19+]

Creates and manages windows and controls. Such windows can be used as data entry forms or custom user interfaces.

GUI, sub-command [, Param2, Param3, Param4]

Table of Contents

- [Menu](#): Adds or removes a menu bar.
- [Color](#): Sets the background color for the window and/or its controls.
- [Font](#): Sets the typeface, size, style, and text color for subsequently created controls.
- [Options](#): Sets various options for the appearance and behavior of the window.
- [Flash](#): Blinks the window and its taskbar button.
- [Show](#): Displays the window.
- [Submit](#): Saves the user's input and optionally hides the window.
- [Cancel](#): Hides the window.
- [Destroy](#): Deletes the window.
- [Default](#): Changes the current thread's default GUI window number.
- [Add](#): Creates a control. Click one of the following types for details:
 - [Text, Edit, Hotkey, Picture](#)
 - [GroupBox, Button, Checkbox, Radio](#)
 - [DropDownList, ComboBox, ListBox](#)
 - [Slider, Progress, Tab](#)
- [Positioning and sizing of controls](#)
- [Acting upon user input: Variables and G-Labels](#)
- [Other options and styles](#)
- [Remarks](#)
- [Examples](#)

Menu [, MenuName]

~~Attaches a menu bar to the window. Use the [Menu](#) command to create an ordinary menu for this purpose. To remove a window's current menu bar, omit the [MenuName](#) parameter.~~

~~Once a menu has been used as a menu bar, it should not be used as a popup menu or a submenu. This is because menu bars internally require a different format (note: this restriction applies only to the menu bar itself, not its submenus). If you need to work around this, create one menu to use as the menu bar and another identical menu to use as a popup menu or submenu.~~

~~The use of certain destructive menu sub-commands such as Delete and DeleteAll against a menu that is currently being used as a menu bar will not succeed (a warning dialog is displayed unless the UseErrorLevel setting is in effect). To make such changes:~~
~~1) detach the menu bar (see above); 2) make the changes; 3) reattach the menu bar. This restriction does not apply to the menu bar's submenus.~~

Color [, WindowColor, ControlColor]

Sets the background color of the window and/or its controls. *ControlColor* is applied to all present and future controls in the window (with the exception of a few types that do not support a custom color).

Leave either parameter blank to retain the current color. Otherwise, specify one of the 16 primary [HTML color names](#) or a 6-digit RGB color value (the 0x prefix is optional), or specify the word Default to return either to its default color. Example values: Silver, FFFFFAA, 0xFFFFAA, Default

By default, the window's background color is the system's color for the face of buttons, and the controls' background color is the system's default window color (usually white).

The color of the menu bar and its submenus can be changed as in this example: [Menu](#), MyMenuBar, Color, White

To make the background transparent, use [WinSet TransColor](#). However, if you do this without first having assigned a custom window via "Gui, Color", buttons will also become transparent. To prevent this, first assign a custom color and then make that color transparent as in this example:

Gui, Color, EEAA99

WinSet, TransColor, EEAA99, My Window Title ; But first turn [DetectHiddenWindows](#) on if needed.

To additionally remove the border and title bar from a window with a transparent background, use the following *after* the window has been made transparent:

Gui, -Caption ; Or use "Gui, 2:-Caption" if it is the second window, etc.

To illustrate the above, there is an example of an on-screen display (OSD) at the bottom of this page.

Font [, Options, FontName]

~~Sets the font typeface, size, style, and/or color for controls created from this point onward.~~

~~Omit both parameters to restore the font to the system's default GUI typeface, size, and color.~~

~~FontName may be the name of any font, such as one from the font table. If FontName is omitted or does not exist on the system, the previous font's typeface will be used (or if none, the system's default GUI typeface). This behavior is useful to make a GUI window have a similar font on multiple systems, even if some of those systems lack the preferred font. For example, by using the following commands in order, Verdana will be given preference over Arial, which in turn is given preference over MS sans serif:~~

~~gui, font,, MS sans serif~~

~~gui, font,, Arial~~

~~gui, font,, Verdana ; Preferred font.~~

~~If the Options parameter is blank, the previous font's attributes will be used. Otherwise, specify one or more of the following option letters as substitutes:~~

~~C: Color name (see color chart) or RGB value — or specify the word Default to return to the system's default color (black on most systems). Example values: cRed, cFFFFAA, cDefault. Note: An individual control can be created with a font color other than the current one by including the C option. Example: "Gui, add, text, cRed, My Text"~~

~~S: Size (in points). Example: s10~~

~~W: Weight (boldness), which is a number between 1 and 1000 (400 is normal and 700 is bold). Example: w600~~

~~The following words are also supported: bold, italic, strike, underline, and norm. Norm returns the font to normal weight/boldness and turns off italic, strike, and underline (but it retains the existing color and size). It is possible to use norm to turn off all attributes and then selectively turn on others. In the following example, the font is set to normal and then to italic: norm italic~~

~~To specify more than one option, include a space between each. Example: cBlue s12 bold~~

~~If a script creates multiple GUI windows (see remarks section), each window remembers its own "current font" for the purpose of creating more controls.~~

~~-~~

+/-Option1 +/-Option2 ...

One or more options may be specified immediately after the GUI command. For performance reasons, it is better to do this before creating the window; that is, before any use of other sub commands such as "Gui Add".

The effect of this command is cumulative; that is, it alters only those settings that are explicitly specified, leaving all the others unchanged.

Specify a plus sign to add the option and a minus sign to remove it. Example:

Gui, -resize +owner ; Change the current/default GUI window.

Gui, 2:-resize +owner ; Change the second GUI window.

Options other than Resize and Owner require version 1.0.21 or greater.

Border: Provides a thin-line border around the window. This is not common.

Caption: Provides a title bar and a thick window border/edge. [Default]

MaximizeBox: Enables the maximize button in the title bar. This is also included as part of Resize below. *MinimizeBox and MaximizeBox can only be used together. They only work before you have run Gui, Show.*

MinimizeBox: Enables the minimize button in the title bar. [Default] *MinimizeBox and MaximizeBox can only be used together. They only work before you have run Gui, Show.*

~~**Owner:** Use +owner to make the window owned by another (but once the window is created, owner has no effect). An owned window has no taskbar button by default, and when visible it is always on top of its owner. It is also automatically destroyed when its owner is destroyed.~~

~~This option must be used after the window's owner is created but before the owned is created (via a command such as "Gui Add"). There are two ways to use +owner as shown in these examples:~~

~~gui, 2:+owner1 ; Make #2 window owned by #1 window;~~

~~gui, +owner ; Make #1 window owned by script's main window.~~

~~On a related note, to temporarily prevent the user from interacting with a window — such as while one of its owned windows is visible — disable the window via "Gui, 1:+0x8000000". To re-enable it later, use "Gui, 1:-0x8000000" followed by "Gui, 1:Show".~~

Resize: Makes the window resizable and enables its maximize button in the title bar. To avoid enabling the maximize button, specify "+Resize -MaximizeButton".

SysMenu: Provides an icon with a drop down menu in the upper left corner of the title bar. [Default]

~~**Theme:** By specifying Theme, all subsequently created controls in the window will have Classic Theme appearance on Windows XP and beyond. To later create additional controls in the window that obey the current theme, turn it back on via +Theme. Note: This option has no effect on operating systems older than Windows XP, nor does it have any effect on XP itself if the Classic Theme is in effect.~~

ToolWindow: Provides a narrower title bar but the window will have no taskbar button.

(Unnamed Style): Specify a plus or minus sign followed immediately by a decimal or hexadecimal style number. You can find a list of Styles with their respective numbers [here](#). Currently, only one code is supported: WS_POPUP (Gui, +0x80000000). This prevents the window from ever grabbing focus. This option only works when you specify it before any other Gui command as the type can only be set at internal window creation time.

(Unnamed ExStyle): Specify a plus or minus sign followed immediately by the letter E and a decimal or hexadecimal extended style number. For example, +E0x40000 would add the WS_EX_APPWINDOW style, which provides a taskbar button for a window that would otherwise lack one. Although the other extended style numbers are not yet documented here (since they are rarely used), they can be discovered on the Internet. You can find a list of ExStyles with their respective numbers [here](#). Currently, only one code is supported: WS_EX_NOACTIVATE (Gui, +E0x8000000). It doesn't seem to work really, try WS_POPUP instead (see "Unnamed Style" above).

Flash [, Off] (v1.0.21+)

Blinks the window's button in the taskbar. This is done by inverting the color of the window's title bar and/or taskbar button (if it has one). The optional word OFF causes the title bar and taskbar button to return to their original colors (but the actual behavior might vary depending on OS version). In the below example, the window will blink five times because each pair of flashes inverts then restores its appearance:

```
loop, 10
```

```
f
```

```
    Gui, flash
```

```
    Sleep, 300
```

```
f
```

```
-
```

Show [, Options, Title]

Makes the window visible, [activates](#) it, and sets its title. If *Title* is omitted, the previous title is retained (or if none, the script's file name is used). Leave *Options* blank to have the window retain its previous size and position. If there is no previous position, the window will be auto-centered in one or both dimensions if the X and/or Y options mentioned below are absent. If there is no previous size, the window will be auto-sized according to the size and positions of the controls it contains.

For *Options*, the following single letters are supported but rarely needed due to auto-sizing and centering mentioned above: **W** (Width), **H** (Height), **X** (X-position), and **Y** (Y-position). Include a number immediately after each option letter. The specified Width and Height are that of the client area, which will often be smaller than the width and height reported by [WinGetPos](#) because the client area excludes the title bar, menu bar, and window edges.

Another option is the word Center, which centers the window in both dimensions. Similarly, the string xCenter or yCenter may be specified to center the window in only one dimension. Examples:

```
Gui, Show, Center
```

```
Gui, Show, xCenter y0
```

In v1.0.24, the word AutoSize resizes the window to accommodate only its currently visible controls. This is useful to resize the window after new controls are added, or existing controls are resized, hidden, or unhidden. Example: Gui, Show, AutoSize Center

Submit [, NoHide]

Saves the contents of each control to its [associated variable](#) (if any) and hides the window unless the word NoHide is present. If the window does not exist -- perhaps due to having been destroyed via "Gui Destroy" -- this command has no effect.

Cancel (the word Hide can also be used)

Hides the window without saving the controls' contents to their associated variables. If the window does not exist -- perhaps due to having been destroyed via "Gui Destroy" -- this command has no effect.

Destroy

Default [v1.0.24+]

Changes the current thread's default GUI window number, which is used whenever a window number is not specified for GuiControl, GuiControlGet, and Gui. In the following example, the default window number is changed to 2: Gui, 2:Default. See Remarks for more information about the default window.

-

Add, ControlType [, Options, Text]

Adds a control to a parent window (first creating the parent window itself, if necessary).

ControlType is one of the following:

- [Text](#), [Edit](#), [Hotkey](#), [Picture](#)
- [GroupBox](#), [Button](#), [Checkbox](#), [Radio](#)
- [DropDownList](#), [ComboBox](#), [ListBox](#)
- [Slider](#), [Progress](#), [Tab](#)

Text: Unmodifiable text with no border; often used to label other controls. In this case, *Text* is the string to display. It may contain linefeeds (`n) to start new lines. If a width (W) is specified in *Options* but no [rows \(R\)](#) or height (H), the text will be word-wrapped as needed, and the control's height will be set automatically.

Edit: An area where free-form text can be typed. In this case, *Text* is the default text to display inside the edit field (linefeed [`n] may be used to start new lines in a multi-line edit control). The control will be multi-line if it has more than one row of text. For example, specifying R3 in *Options* will create a 3-line edit control with the following default properties: a vertical scroll bar, word-wrapping enabled, and the Enter key captured as part of the input rather than triggering the window's default button. Note: To type a literal tab inside a multi-line edit control, the user may press Control-Tab.

The R option is not supported, use W/H instead. However, to make the edit be a multi-line one, specify any R > 1.

When a multi-line edit control is saved to its variable via [Gui Submit](#) or [GuiControlGet](#), any line breaks in the text will be represented as plain linefeeds (`n) rather than the traditional CR+LF (`r`n) used by non-GUI commands such as [ControlGetText](#) and [ControlSetText](#). If the control has word-wrapping enabled, any wrapping that occurs as the user types will not result in linefeed characters (only the Enter keystroke can do that).

Although multi-line edit controls are limited to 64 KB of text on Windows 95/98/Me, they may have as much as 4 GB of text on Windows NT/2k/XP or later. As the user enters text, more memory is allocated as needed.

Edit Options (to remove an option rather than adding it, precede it with a minus sign)

Number: Prevents the user from typing anything other than digits into the field. Note: It is still possible to paste non-digits into it.

Lowercase: The characters typed by the user are automatically converted to lowercase.

Uppercase: The characters typed by the user are automatically converted to uppercase.

Limit: Restricts the user's input to the visible width of the edit field. Alternatively, to limit input to a specific number of characters, include a number immediately afterward. For example, Limit10 would allow no more than 10 characters to be entered.

Password: Hides the user's input (such as for password entry) by substituting masking characters for what the user types. If a non-default masking character is desired, include it immediately after the word Password. For example, Password* would make the masking character an asterisk rather than the black circle (bullet), which is the default on Windows XP and beyond. Note: This option has no effect for multi-line edit controls.

~~ReadOnly~~: Prevents the user from changing the control's contents. However, the text can be scrolled, selected and copied to the clipboard.

~~Multi~~: Makes it possible to have more than one line of text. However, it is usually not necessary to specify this because it will be auto-detected based on height (H), rows (R), or contents (Text).

~~WantReturn~~: Specify ~~WantReturn~~ (that is, a minus sign followed by ~~WantReturn~~) to prevent a multi-line edit control from capturing the Enter keystroke. Pressing Enter will then be the same as pressing the window's default button (if any). In this case, the user may press ~~Control+Enter~~ as a means to start a new line.

~~Tn [v1.0.24+]~~: The letter T may be used to set tab stops inside a multi-line edit control (since tab stops determine the column positions to which literal TAB characters will jump, they can be used to format the text into columns). If the letter T is not used, tab stops are set at every 32 units. If the letter T is used once, tab stops are set at every n units across the entire width of the control. For example, ~~Gui, Add, Edit, vMyEdit r16 t64~~ would double the default distance between tab stops. To have custom tab stops, specify the letter T multiple times as in the following example: ~~Gui, Add, Edit, vMyEdit r16 t8 t16 t32 t64 t128~~. One tab stop is set for each of the absolute column positions in the list, up to a maximum of 50 tab stops. To type a literal tab inside a multi-line edit control, the user may press ~~Control+Tab~~.

Hotkey [v1.0.23+]: A box that looks like a single line edit control but instead accepts a keyboard combination pressed by the user. For example, if the user presses ~~Control+Alt+C~~ on an English keyboard layout, the box would display "Ctrl + Alt + C".

When the ~~Gui Submit~~ command is used, the control's associated output variable (if any) receives the hotkey modifiers and name, which usually should be compatible with the ~~Hotkey~~ command. Examples: ~~^!C, +!!Home, + ^Down, ^Numpad1, !NumpadEnd~~. If there is no hotkey in the control, the output variable is made blank. Note: Some keys are displayed the same even though they are retrieved as different names. For example, both ~~^Numpad7~~ and ~~^NumpadHome~~ might be displayed as ~~Ctrl + Num 7~~.

By default, the control starts off with no hotkey specified. To instead have a default, specify its modifiers and name as the ~~Text~~ parameter. Example: ~~"Gui, Add, Hotkey, vMyHotkey, ^!p"~~. The only modifiers supported are ~~^~~ (Control), ~~!~~ (Alt), and ~~+~~ (Shift). See the key list for available key names.

To restrict the types of hotkeys the user may enter, include the word ~~Limit~~ followed by the sum of one or more of the following numbers:

~~1: Prevent unmodified keys~~

~~2: Prevent Shift-only keys~~

~~4: Prevent Control-only keys~~

~~8: Prevent Alt-only keys~~

~~16: Prevent Shift-Control keys~~

~~32: Prevent Shift-Alt keys~~

~~64: This value is not supported (it will not behave correctly)~~

~~128: Prevent Shift-Control-Alt keys~~

For example, ~~Limit1~~ would prevent unmodified hotkeys such as letters and numbers from being entered, and ~~Limit15~~ would require at least two modifier keys. If the user types a forbidden modifier combination, the ~~Control+Alt~~ combination is visibly and automatically substituted.

The ~~Hotkey~~ control has limited capabilities. For example, it does not support mouse/joystick hotkeys or the Windows key (LWin and RWin). One way to work around this is to provide one or more checkboxes as a means for the user to enable extra modifiers such as the Windows key.

Picture (or Pic): An area containing an image (see last paragraph for supported file types). The ~~Text~~ parameter is the filename of the image, which is assumed to be in [%A_WorkingDir%](#) if an absolute path isn't specified. You can also specify a standard icon name by prefixing it with "icon:". Example: [Gui, Add, Picture, , icon:appointment-new](#). To retain the image's actual width and/or height, omit the W and/or H options. Otherwise, the image is scaled to the specified width and/or height. To shrink or enlarge the image while preserving its aspect ratio, specify -1 for one of the dimensions and a positive number for the other. For example, specifying "w200 h-1" would make the image 200 pixels wide and cause its height to be set automatically. If the picture cannot be loaded or displayed (e.g. file not found), the control is left empty and its height and width are set to zero.

To use a picture as a background for other controls, the picture should normally be added prior to those controls. However, if those controls are input-capable and the picture has a [g-label](#), create the picture after the other controls and include [+0x4000000](#) (which is [WS_CLIPSIBLINGS](#)) in the picture's [Options](#). This trick also allows a picture to be the background behind a Tab control.

Icons and cursors: In v1.0.24, icons (.ico), cursors (.cur), and animated cursors (.ani) are supported on all operating systems. In addition, icons can be loaded from EXE and DLL files that contain them. To use an icon other than the first one in the file, include in Options the word Icon followed by the number of the icon. In the following example, the second icon would be used: Gui, Add, Picture, Icon2, C:\My Application.exe. Note: Specifying an icon number causes a different icon loading method to be used, which might prevent a .ani file from being animated. It also prevents AltSubmit (see next paragraph) from taking effect.

Specifying the word AltSubmit in Options tells the program to use Microsoft's GDIPlus.dll to load the image, which might result in a different appearance for certain types of images. For example, it would load an icon containing a transparent background as a transparent bitmap, which allows the BackgroundTrans option to take effect for such an icon. If GDIPlus is not available (see next paragraph), AltSubmit is ignored and the image is loaded using the normal method.

All operating systems support GIF, JPG, and BMP images. With AutoHotkey v1.0.22+ running on Windows XP or later, additional image formats such as PNG, TIF, ICO, Exif, WMF, and EMF are supported. Operating systems older than XP can be given support by copying Microsoft's free GDI+ DLL into the AutoHotkey.exe folder (but in the case of a compiled script, copy the DLL into the script's folder). To download the DLL, search for the following phrase at www.microsoft.com/gdi/redistributable.

Due to an OS bug, animated cursors scaled to a size greater than 90x90 on Windows 95/98/Me might crash the script.

GroupBox: A rectangular border/frame, often used around other controls to indicate they are related. In this case, Text is the title of the box, which is displayed at its upper-left edge.

Button: A pushbutton, which can be pressed to trigger an action. In this case, Text is the name of the button (shown on the button itself), which may include linefeeds (`n) to start new lines. Include the word Default in Options to make it the default button. The default button's action is automatically triggered whenever the user presses ENTER, except when the keyboard focus is on a different button or a multi-line edit control having the "WantReturn" style. To later change the default button to another button, follow this example, which makes the OK button become the default: GuiControl, +default, OK. To later change the window to have no default button, follow this example: GuiControl, -default, OK

Checkbox: A small box that can be checked or unchecked to represent On/Off, Yes/No, etc. Specify the word Check3 in Options to enable a third state that displays a gray checkmark instead of a black one (the gray state indicates that the checkbox is neither checked nor unchecked). Specify the word Checked or CheckedGray in Options to have the checkbox start off with a black or gray checkmark, respectively. The Text parameter is a label displayed next to the box, typically used as a prompt or description of what the checkbox does. It may include linefeeds (`n) to start new lines. If a width (W) is specified in Options but no rows (R) or height (H), Text will be word-wrapped as needed, and the control's height will be set automatically. The variable associated with a checkbox (if any) receives the number 1 for checked, 0 for unchecked, and -1 for gray/indeterminate.

Radio: Radio button. A small empty circle that can be checked (on) or unchecked (off). These controls usually appear in radio groups, each of which contains two or more radio buttons. When the user clicks a radio button to turn it on, any others in its radio group are turned off automatically (the user may also navigate inside a group with the arrow keys). A radio group is created automatically around all consecutively added radio buttons. To start a new group, specify the word Group in the Options of the first button of the new group, or simply add a non-radio control since that automatically starts a new group.

Specify the word Checked in Options to have the button start off in the "on" state. For the Text parameter, specify the label to display to the right of the radio button. This label is typically used as a prompt or description, and it may include linefeeds (`n) to start new lines. If a width (W) is specified in Options but no rows (R) or height (H), Text will be word-wrapped as needed, and the control's height will be set automatically.

The variable associated with a radio button (if any) receives the number 1 for "on" and 0 for "off". However in v1.0.20+, if only one button in a radio group has a variable, that variable will instead receive the number of the currently selected button: 1 is the first radio button (according to original creation order), 2 is the second, and so on. If there is no button selected, 0 is stored.

DropDownList (or **DDL**): A list of choices that is displayed in response to pressing a small button. In this case, Text is a pipe-delimited list of choices such as Choice1|Choice2|Choice3. To have one of the items pre-selected when the window first appears, include two pipe characters after it. Alternatively, include in Options the word Choose followed immediately by the number to pre-select. For example, Choose5 would pre-select the fifth item. To change the choice or add/remove entries from the list after the control has been created, use GuiControl.

Specify either the word Uppercase or Lowercase in Options to automatically convert all items in the list to upper or lower case. Specify the word Sort to automatically sort the contents of the list alphabetically (this also affects any items added later via GuiControl).

When the [Gui Submit](#) command is used, the control's [associated output variable](#) (if any) receives the text of the currently selected item. However, if the control has the [AltSubmit](#) property, the output variable will receive the item's position number instead (the first item is 1, the second is 2, etc.).

ComboBox: Same as [DropDownList](#) but also permits free form text to be entered as an alternative to picking an item from the list. In addition to allowing all the same options as [DropDownList](#) above, the word [Limit](#) may be included in [Options](#) to restrict the user's input to the visible width of the [ComboBox](#)'s edit field. Also, the word [Simple](#) may be specified to make the [ComboBox](#) behave as though it is an [Edit](#) field with a [ListBox](#) beneath it.

When the [Gui Submit](#) command is used, the control's associated output variable (if any) receives the text of the currently selected item. However, if the control has the [AltSubmit](#) property, the output variable will receive the item's position number instead (the first item is 1, the second is 2, etc.). If either case, if there is no selected item, the output variable will be set to the contents of the [ComboBox](#)'s edit field.

ListBox: A relatively tall box containing a list of choices that can be selected. In this case, [Text](#) is is a pipe delimited list of choices such as [Choice1|Choice2|Choice3](#). To have list item(s) pre-selected when the window first appears, include two pipe characters after each (the multi-select option described below is required if more than one item is to be pre-selected). Alternatively, include in [Options](#) the word [Choose](#) followed immediately by a single item number to pre-select. For example, [Choose5](#) would pre-select the fifth item. To change the choice or add/remove entries from the list after the control has been created, use [GuiControl](#).

Specify the word [ReadOnly](#) in [Options](#) to prevent items from being visibly highlighted when they are selected (but [Gui Submit](#) will still store the selected item). Specify the word [Sort](#) to automatically sort the contents of the list alphabetically (this also affects any items added later via [GuiControl](#)).

When the [Gui Submit](#) command is used, the control's associated output variable (if any) receives the text of the currently selected item. However, if the control has the [AltSubmit](#) property, the output variable instead receives the item's position number (the first item is 1, the second is 2, etc.).

Multi-select ListBox: Specify the word [Multi](#) in [Options](#) to allow more than one item to be selected simultaneously via shift-click and control-click (to avoid the need for shift/control-click, specify the number [8](#) instead of the word [Multi](#)). In this case, [Gui Submit](#) stores a pipe delimited list of item strings in the control's output variable. However, if the [AltSubmit](#) option is in effect, [Gui Submit](#) stores a pipe delimited list of item numbers instead. For example, [1|2|3](#) would indicate that the first three items are selected. To extract the individual items from the string, use a parsing loop such as this example:

```
Loop, parse, MyListBox,
```

```
{
```

```
    MsgBox Selection number %A_Index% is %A_LoopField%.
```

```
}
```

Slider [v1.0.23+]: A sliding bar that the user can move along a vertical or horizontal track. The standard volume control in the taskbar's tray is an example of a slider.

The user may slide the control by the following means: 1) dragging the bar with the mouse; 2) clicking away from the bar with the mouse; 3) turning the mouse wheel while the control has focus; or 3) pressing the following keys while the control has focus: Arrow keys, Page up, Page down, Home, and End.

Specify the starting position number of the slider as the [Text](#) parameter. If [Text](#) is omitted, the slider starts off at 0 or the number in the allowable range that is closest to 0.

When the [Gui Submit](#) command is used, the control's associated output variable (if any) receives the current numeric position of the slider. The position is also stored in the output variable whenever the control's [g](#) label is launched.

If there is a [g](#) label, by default it will be launched only when the user has stopped moving the slider (such as by releasing the mouse button after having dragging it). However, if a slider has the word [AltSubmit](#) in its [Options](#), the [g](#) label is launched for all slider events and the built in variable [A_GuiControlEvent](#) will contain one of the following digits or strings:

- 0: The user pressed the Left arrow or Up arrow key.
- 1: The user pressed the Right arrow or Down arrow key.
- 2: The user pressed the Page up key.
- 3: The user pressed the Page down key.

- 4: The user moved the slider via the mouse wheel, or finished a drag and drop to a new position.
 - 5: The user is currently dragging the slider via the mouse; that is, the mouse button is currently down.
 - 6: The user pressed the Home key to send the slider to the left or top side.
 - 7: The user pressed the End key to send the slider to the right or bottom side.
- Normal: The user has finished moving the slider, either via the mouse or the keyboard. Note: With the exception of mouse wheel movement (#4), the g_label is launched again for the "normal" event even though it was already launched for one of the digit events above.

Slider Options

Vertical: Makes the control slide up and down rather than left and right.

Invert [v1.0.24+]: Reverses the control so that the lower value is considered to be on the right/bottom rather than the left/top. This is typically used to make a vertical slider move in the direction of a traditional volume control. Note: The ToolTip option described below will not obey the inversion and therefore should not be used in this case.

Range: Sets the range to be something other than 0 to 100. After the word Range, specify the minimum, a dash, and maximum. For example, Range1 1000 would allow a number between 1 and 1000 to be selected; Range -50-50 would allow a number between -50 and 50; and Range -10-5 would allow a number between -10 and -5.

Center: The thumb (the bar moved by the user) will be blunt on both ends rather than pointed at one end.

Left: The thumb (the bar moved by the user) will point to the top rather than the bottom. But if the Vertical option is in effect, the thumb will point to the left rather than the right.

NoTicks: No tickmarks will be displayed.

TickCount: Provides tickmarks at the specified interval. After the word TickInterval, specify the interval at which to display additional tickmarks (if the interval is omitted, it is assumed to be one). For example, TickInterval10 would display a tickmark once every 10 positions.

ToolTip: Creates a tooltip that reports the numeric position of the slider as the user is dragging it. To have the tooltip appear in a non-default position, specify one of the following instead: ToolTipLeft or ToolTipRight (for horizontal sliders); ToolTipTop or ToolTipBottom (for vertical sliders). Windows 95 and NT4 require Internet Explorer 3.0 or later to support this option.

Line: Specifies the number of positions to move when the user presses one of the arrow keys. After the word Line, specify number of positions to move. Example: Line2

Page: Specifies the number of positions to move when the user presses the Page-up or Page-down key. After the word Page, specify number of positions to move. Example: Page10

Thick: Specifies the length of the thumb (the bar moved by the user). After the word Thick, specify the thickness in pixels (e.g. Thick30). To go beyond a certain thickness on Windows XP or later, it is probably necessary to either specify the Center option or remove the theme from the control (which can be done by specifying "Gui -theme" prior to creating it and "Gui +theme" afterward).

Buddy1 and Buddy2: Specifies up to two existing controls to automatically reposition at the ends of the slider. Buddy1 is displayed at the left or top side (depending on whether the Vertical option is present). Buddy2 is displayed at the right or bottom side. After the word Buddy1 or Buddy2, specify the variable name of an existing control. For example, Buddy1MyTopText would assign the control whose variable name is MyTopText. Windows 95 and NT4 require Internet Explorer 3.0 or later to support this option.

The above options can be changed after the control is created via GuiControl.

#Progress [v1.0.23+]: A dual color bar typically used to indicate how much progress has been made toward the completion of an operation.

Specify the starting position number of the bar as the Text parameter. If Text is omitted, the bar starts off at 0 or the number in the allowable range that is closest to 0. To later change the position of the bar, follow these examples, all of which operate upon a progress bar whose associated variable name is MyProgress:

GuiControl,, MyProgress, +20 ; Increase the current position by 20.

GuiControl,, MyProgress, 50 ; Set the current position to 50.

For horizontal Progress Bars, the thickness of the bar is equal to the the control's height. For vertical Progress Bars it is equal to the control's width.

Progress Options

Vertical: Makes the bar rise or fall vertically rather than move along horizontally. Windows 95 and NT4 require Internet Explorer 3.0 or later to support this option.

Color: Changes the bar's color. Specify for *n* one of the 16 primary HTML color names or a 6-digit RGB color value. Examples: cRed, cFFFF33, cDefault. If the C option is never used (or cDefault is specified), the system's default bar color will be used.

BackgroundN: Changes the bar's background color. Specify for *n* one of the 16 primary HTML color names or a 6-digit RGB color value. Examples: BackgroundGreen, BackgroundFFFF33, BackgroundDefault. If the Background option is never used (or BackgroundDefault is specified), the background color will be that of the window or tab control behind it.

Range: Sets the range to be something other than 0 to 100. After the word Range, specify the minimum, a dash, and maximum. For example, Range1-1000 would allow numbers between 1 and 1000, Range-50-50 would allow numbers between -50 and 50, and Range-10-5 would allow numbers between -10 and -5. On Windows 95 and NT4, negative ranges and ranges beyond 65535 will not behave correctly unless Internet Explorer 3.0 or later is installed.

Smooth (minus Smooth): Displays a length of segments rather than a smooth continuous bar. Specifying Smooth is also one of the requirements to show a themed progress bar on Windows XP or later. The other requirement is that the bar not have any custom colors; that is, that the C and Background options be omitted. Windows 95 and NT4 require Internet Explorer 3.0 or later to support this option.

The above options can be changed after the control is created via GuiControl:

Tab [v1.0.22+]: A relatively large control containing multiple pages, each of which contains other controls. From this point forward, these pages are referred to as "tabs".

The Text parameter is a pipe-delimited list of tabs such as General|View|Appearance|Settings. To have one of the tabs pre-selected when the window first appears, include two pipe characters after it. Alternatively, include in Options the word Choose followed immediately by the number to pre-select. For example, Choose5 would pre-select the fifth tab. To change the selected tab, add tabs, or remove tabs after the control has been created, use GuiControl.

After creating a Tab control, subsequently added controls automatically belong to its first tab. This can be changed at any time by following these examples:

Gui, Tab -- Future controls are not part of any tab control.

Gui, Tab, Name -- Future controls are owned by the tab whose name starts with Name (not case sensitive).

Gui, Tab, 3 -- Future controls are owned by the third tab of the current tab control.

Gui, Tab, 3, 2 -- Future controls are owned by the third tab of the second tab control.

It is possible to use the "Gui Tab" command above to assign controls to a tab or tab control that doesn't yet exist (except in the case of the Name method). But in that case, the relative positioning options described below are not supported.

Positioning: When each tab of a Tab control receives its first sub-control, that sub-control will have a special default position under the following conditions: 1) The X and Y coordinates are both omitted, in which case the sub-control is positioned at the upper-left corner of the tab control's interior (with a standard margin); 2) The X+n and/or Y+n positioning options are specified, in which case the sub-control is positioned relative to the upper-left corner of the tab control's interior. For example, specifying "x+10 y+10" would position the control 10 pixels right and 10 pixels down from the upper-left corner.

Sub-controls do not necessarily need to exist within their Tab control's boundaries: they will still be hidden and shown whenever their tab is selected or de-selected. This behavior is especially useful with the "buttons" style described below.

When the Gui Submit command is used, the control's associated output variable (if any) receives the name of the currently selected tab. However, if the control has the AltSubmit property, the output variable will receive the tab's position number instead (the first tab is 1, the second is 2, etc.).

If a tab control has both a g label and an output variable, the output variable will be set to the previously selected tab name (or number in the case of AltSubmit) whenever the user switches to a new tab.

Styles and Options: Specify background (minus followed by the word background) to override the window's custom background color and use the system's default Tab control color. Specify wrap to prevent the tabs from taking up more than a single row (in which case if there are too many tabs to fit, arrow buttons are displayed to allow the user to slide more tabs into view). Specify the word Buttons to have a series of buttons at the top of the control rather than a series of tabs (in this case, there will be no border by default because the display area does not typically contain controls). To have the tabs on the left, right, or bottom side instead of the top, specify in Options the word Left, Right, or Bottom, respectively (but see TCS_VERTICAL for limitations on Left and Right).

~~Keyboard navigation: The user may press Control-PageDown/PageUp to navigate from page to page in a tab control; if the keyboard focus is on a control that does not belong to a Tab control, the window's first Tab control will be navigated. Control-Tab and Control-Shift-Tab may also be used except that they will not work if the currently focused control is a multi-line Edit control.~~

~~Each window may have no more than 255 tab controls. Each tab control may have no more than 256 tabs (pages).~~

~~# Positioning and Layout via SmartGUI Creator~~

~~Although the options described in the next section are suitable for simple layouts, you may find it easier to use Rajat's SmartGUI Creator because it's entirely visual, i.e. "what you see is what you get". SmartGUI Creator is free and can be downloaded from <http://www.autohotkey.com/docs/SmartGUI/>~~

~~Options for Size and Position~~

If some dimensions and/or coordinates are omitted from *Options*, the control will be positioned relative to the previous control and/or sized automatically according to its nature and contents.

The following options are supported:

~~# R: Rows of text (can contain a floating point number such as R2.5). R is often preferable to specifying H (Height). If both the R and H options are present, R will take precedence. For a GroupBox, this setting is the number of controls for which to reserve space inside the box. For DropDownLists, ComboBoxes, and ListBoxes, it is the number of items visible at one time inside the list portion of the control (but on Windows XP, it is often desirable to omit both the R and H options for DropDownList and ComboBox, which makes the popup list automatically take advantage of the available height of the user's desktop). For other control types, R is the number of rows of text that can fit inside the control.~~

W: Width, in pixels. If omitted, the width is calculated automatically for some control types based on their contents. The other controls types have the following default widths:

Tab controls: 30 times the current font size, plus 3 times the X margin.

Vertical Progress Bars: Two times the current font size.

Horizontal Progress Bars, horizontal Sliders, DropDownLists, ComboBoxes, ListBoxes, GroupBoxes, Edits, and Hotkeys: 15 times the current font size (except GroupBoxes, which multiply by 18 to provide room inside for margins).

H: Height, in pixels. If both the H and R options are absent, DropDownLists, ComboBoxes, ListBoxes, and empty multi-line Edit controls default to 3 rows; GroupBoxes default to 2 rows; vertical Sliders and Progress Bars default to 5 rows; horizontal Sliders default to 30 pixels (except if a thickness has been specified); horizontal Progress Bars default to 2 times the current font size; Hotkey controls default to 1 row; and Tab controls default to 10 rows. For the other control types, the height is calculated automatically based on their contents. Note that for DropDownLists and ComboBoxes, H is the combined height of the control's always-visible portion and its list portion (but even if the height is set too low, at least one item will always be visible in the list). Also, for all types of controls, specifying the number of rows via the R option is usually preferable to using H because it prevents a control from showing partial/incomplete rows of text.

~~wp+n, hp+n, wp-n, hp-n (where n is any number) can be used to set the width and/or height of a control equal to the previously added control's width or height, with an optional plus or minus adjustment. For example, wp would set a control's width to that of the previous control, and wp-50 would set it equal to 50 less than that of the previous control. [requires 1.0.23+]~~

X: X-position. For example, specifying "x0 y0" would position the control in the upper left corner of the window's client area, which is the area beneath the title bar and menu bar (if any). If X is omitted but not Y, the control will be positioned to the right of all previously added controls, which can be thought of as starting a new "column".

Y: Y-position. If Y is omitted but not X, the control will be positioned beneath all previously added controls, which can be thought of as starting a new "row".

Omitting either X, Y or both is useful to make a GUI layout automatically adjust to any future changes you might make to the size of controls or font point size. By contrast, specifying an absolute position for every control might require you to manually shift the position of all controls that lie beneath and/or to the right of a control that is being enlarged or reduced.

If both X and Y are omitted, the control will be positioned beneath the previous control using a standard padding distance.

~~# For X and Y, an optional plus sign can be included to position a control relative to the bottom or right edge (respectively) of the control that was previously added. For example, specifying Y+10 would position the control 10 pixels beneath the bottom of the previous control rather than using the standard padding distance. Similarly, specifying X+10 would position the control 10 pixels to the right of the previous control's right edge. Note: minus signs cannot be used because they are reserved for absolute positioning.~~

xp+n, yp+n, xp-n, yp-n (where **n** is any number) can be used to position controls relative to the previous control's upper left corner, which is often useful for enclosing controls in a GroupBox.

xm and **ym** can be used to position a control at the leftmost and topmost margins of the window, respectively (these two may also be followed by a plus/minus sign and a number). A window's margin size is proportional to the font size that was in effect when its first control was added. By specifying **ym** without any x-position at all, the control will be positioned at the top margin but to the right of all previously added controls, which can be thought of as starting a new "column". The converse is also true.

~~# In v1.0.20+, **xs** and **ys** are similar to **xm** and **ym** except that they refer to coordinates that were saved by having previously added a control with the word **Section** in its options (the first control of the window always starts a new section, even if that word isn't specified in its options). By specifying **ys** without any x-position at all, the control will be positioned at the previously saved y-position, but to the right of all controls that have been added since the most recent use of the word **Section**; this can be thought of as starting a new column within the section. Example:~~

```
gui, add, edit, w600 ; Add a fairly wide edit control at the top of the window.  
gui, add, text, section, First Name: ; Save this control's position and start a new section.  
gui, add, text,, Last Name:  
gui, add, edit, ys ; Start a new column within this section.  
gui, add, edit  
gui, show
```

The converse of the above (specifying **xs** but omitting the y-position) is also true.

xs and **ys** may optionally be followed by a plus/minus sign and a number. Also, it is possible to specify both the word **Section** and xs/ys in a control's options, which uses the previous section for itself but establishes a new section for subsequent controls.

Options for Assigning Actions and Variables to Controls

V: Variable. To associate a variable with a control, include the letter V followed immediately by the name of the variable. Example: vMyEdit. This variable will be given the contents of its control whenever the **Gui Submit** command is used. If a control is not input-capable -- such as a Text control or GroupBox -- assigning a variable to it can still be helpful since that variable's name serves as the control's unique identifier for use with **GuiControl** and **GuiControlGet**. Note: **Gui Submit** does not change the contents of variables associated with control types such as Text and GroupBox.

G: Gosub. To launch a subroutine in response to the press of a button, changing of a checkbox/radio, selection of a new choice in a ListBox/ComboBox/DropDownList/Tab control, or the clicking of a Picture or Text control: include the letter G followed immediately by the name of the label to execute. The special label gCancel will perform an implicit "Gui Cancel", but only if no "Cancel" label exists in the script.

The **G** option may be omitted for a button, in which case an automatic label is assumed. For example, if the script contains an OK button and a ButtonOK label, that label will be launched when the button is pressed. However, if the text on the button contains spaces, ampersands, linefeeds (`n) or carriage returns (`r), its automatic label will omit those characters. For example, a button titled "&OK" (that is, containing the letter O underlined as a shortcut key) would have an automatic label of ButtonOK. Similarly, a button titled "Save && Exit" would have an automatic label of ButtonSaveExit. For GUI windows other than the first (see remarks section), the window number is included in front of the button's automatic label, e.g. 2ButtonOK.

You can detect a double-click by checking the contents of the built-in variable [A_GuiControlEvent](#). The following control types support this: Text, Picture, Radio, and ListBox. [requires v1.0.20+]

A label specified by the **G** option cannot contain spaces because spaces are used to separate items in the options list. To work around this, include an additional label above the desired target label as in this example:

```
MyLabel:  
::my hotstring::  
MsgBox Both the above labels will execute this subroutine.  
return
```

~~Common Styles and Other Options~~ [some require v1.0.21+]

~~Note:~~ In the absence of a preceding sign, a plus sign is assumed; for example, Wrap is the same as +Wrap. By contrast, -Wrap would remove the word wrapping property.

~~C:~~ Color of text. Specify the letter C followed immediately by a color name (see color chart) or RGB value (the 0x prefix is optional). Examples: cRed, cFFFFAA, c0xFFFFAA, cDefault

~~Disabled:~~ Makes an input capable control appear in a disabled state (it cannot be activated or modified by the user). Use "GuiControl Enable" to enable it later. Note: To make an Edit control read only, specify the string ReadOnly instead.

~~Hidden:~~ The control is initially invisible. Use "GuiControl Show" to show it later.

~~Left:~~ Left justifies the control's text within its available width (meaningful only for buttons).

~~Right:~~ Right justifies the control's text within its available width. For checkboxes and radio buttons, this also puts the box itself on the right side of the control rather than the left.

~~Center:~~ Centers the control's text within its available width.

~~#TabStop:~~ Use Tabstop (i.e. minus TabStop) to have an input capable control skipped over when the user presses the TAB key to navigate.

~~#Section:~~ Starts a new section and saves this control's position for later use with the xs and ys positioning options described above.

~~#AltSubmit:~~ Uses alternate submit method. For DropDownList, ComboBox, and ListBox this causes the Gui Submit command to store the position of the selected item rather than its text. If no item is selected, a ComboBox will still store the text in its edit field; similarly, a DropDownList or ListBox will still make its output variable blank. Note: AltSubmit also affects the behavior of GuiControlGet when it is used against a control that has this property.

~~Wrap:~~ Enables word wrapping of the control's contents within its available width. Since nearly all control types start off with word wrapping enabled, precede Wrap with a minus sign to disable word wrapping.

~~VScroll:~~ Provides a vertical scroll bar if appropriate for this type of control.

~~HScroll:~~ Provides a horizontal scroll bar if appropriate for this type of control. The rest of this paragraph applies to ListBox only. The horizontal scrolling width defaults to 3 times the width of the ListBox. To specify a different scrolling width, include a number immediately after the word HScroll. For example, HScroll500 would allow 500 pixels of scrolling inside the ListBox. However, if the specified scrolling width is smaller than the width of the ListBox, no scroll bar will be shown. The advantage of this is that it makes it possible for the horizontal scroll bar to be added later via "GuiControl, +HScroll500, MyScrollBar", which is otherwise impossible.

~~Uncommon Styles and Options~~ [requires v1.0.21+]

~~#BackgroundTrans~~ [v1.0.23+]: Uses a transparent background, which allows any control that lies behind a Text, Picture, or GroupBox control to show through. For example, a transparent Text control displayed on top of a Picture control would make the text appear to be part of the picture. Use "GuiControl +background" to remove this option later. See Picture control's AltSubmit section for more information about transparent images.

~~-Background~~ (i.e. minus Background): Uses the standard background color rather than the one set by the "Gui Color" command. This is most often used to make a Tab control have its standard color rather than the window color. Use "GuiControl +background" to remove this option later.

~~Border:~~ Provides a thin line border around the control. Most controls do not need this because they already have a type specific border.

~~(Unnamed Style):~~ Specify a plus or minus sign followed immediately by a decimal or hexadecimal style number. If the sign is omitted, a plus sign is assumed.

~~(Unnamed ExStyle):~~ Specify a plus or minus sign followed immediately by the letter E and a decimal or hexadecimal extended style number. If the sign is omitted, a plus sign is assumed. For example, E0x200 would add the WS_EX_CLIENTEDGE style, which provides a border with a sunken edge that might be appropriate for pictures and other controls. Although the other extended style numbers are not yet documented here (since they are rarely used), they can be discovered on the Internet.

Context Sensitive Options

See the description of each control type above for a list of options that apply only to it.

-

Remarks

The following labels will be automatically associated with a GUI window if they exist in the script:

GuiEscape: Launched when the user presses Escape. If this label is absent, pressing Escape has no effect.

GuiClose: Launched when the window is closed by any of the following: pressing its X button in the title bar, selecting "Close" from its system menu, or closing it with WinClose. If this label is absent, closing the window simply hides it, which is the same effect as "Gui Cancel".

GuiSize [v1.0.24+]: Launched when the window is resized, minimized, maximized, or restored. The variables A_GuiWidth and A_GuiHeight contain the new width and height of the window's client area, which is the area excluding title bar, menu bar, and borders. In addition, ErrorLevel will contain one of the following digits:

0: The window has been restored, or resized normally such as by dragging its edges.

1: The window has been minimized.

2: The window has been maximized.

GuiDropFiles [v1.0.24+]: Launched whenever files/folders are dropped onto the window as part of a drag-and-drop operation (but if the label is already running, drop events are ignored). The subroutine may check the A_GuiControl variable to find out which control the files were dropped onto (blank if none). It may also check A_GuiControlEvents to find out the names of the files that were dropped (each filename except the last is terminated by a linefeed [`\n`]). To extract the individual files, use a parsing loop as shown below:

```
Loop, parse, A_GuiControlEvents, `n
+
    MsgBox, 4,, File number %A_Index% is: `n%A_LoopField%. `n`nContinue?
    IfMsgBox, No, Break
+
;
; To extract only the first file, follow this example.
Loop, parse, A_GuiControlEvents, `n
+
    FirstFile = %A_LoopField%
    Break
+
;
; To process the files in alphabetical order, follow this example.
FileList = %A_GuiControlEvents%
Sort, FileList
Loop, parse, FileList, `n
+
    MsgBox, 4,, File number %A_Index% is: `n%A_LoopField%. `n`nContinue?
    IfMsgBox, No, Break
+
```

To temporarily disable drag and drop for a window, remove the WS_EX_ACCEPTFILES style via "Gui E0x10". To re enable it later, use "Gui +E0x10".

General remarks about GUI events

To have multiple events perform the same subroutine, specify their labels consecutively above the subroutine. For example:

GuiEscape:

GuiClose:

ButtonCancel:

Gui, Cancel ; All of the above labels will execute this subroutine.

return

For windows other than the first (see below), the window's number is used as a prefix, e.g. 2GuiEscape and 2GuiClose.

~~A script may have up to 10 GUI windows simultaneously. To operate upon a window number other than the default (see below), include a number followed by a colon in front of the sub-command as in these examples:~~

~~Gui, 2:Add, Text,, Text for about box.~~

~~Gui, 2>Show~~

~~A GUI thread is defined as any thread launched as a result of a GUI action. GUI actions include selecting an item from a GUI window's menu bar, or triggering one of its g-labels (such as by pressing a button).~~

~~The default window number for a GUI thread is that of the window that launched the thread. Non-GUI threads use 1 as their default.~~

~~Similarly, whenever a GUI thread is launched, that thread's last found window starts off as the GUI window itself. This allows commands for windows and controls -- such as WinMove, WinHide, WinSet, WinSetTitle, and ControlGetFocus -- to omit WinTitle and WinText when operating upon the GUI window itself.~~

~~Clicking on a control while its subroutine is already running from a prior click will have no effect.~~

~~The built-in variables A_Gui and A_GuiControl contain the window number and Control ID that launched the current thread. See their links for details.~~

~~All GUI threads start off fresh with the default values for settings such as SetKeyDelay. These defaults can be changed in the auto-execute section.~~

~~A GUI window may be navigated via the TAB key, which moves keyboard focus to the next input-capable control (controls lacking the TabStop style are skipped). The order of navigation is determined by the order in which the controls were originally added. When the window is shown for the first time, the first input-capable control that has the TabStop style will have keyboard focus.~~

~~Certain controls may contain an ampersand (&) to create a keyboard shortcut, which is displayed in the control's text as an underlined character. A user activates the shortcut by holding down the ALT key then typing the corresponding character. For buttons, checkboxes, and radio buttons, pressing the shortcut is the same as clicking the control. For GroupBoxes and Text controls, pressing the shortcut causes keyboard focus to jump to the first input-capable tabstop control that was created after it. However, if more than one control has the same shortcut key, pressing the shortcut will alternate keyboard focus among all controls with the same shortcut.~~

~~To display a literal ampersand inside the control types mentioned above, specify two consecutive ampersands as in this example: Save && Exit~~

~~When the first control is added to a window, the window acquires a default margin on all sides proportional to the size of the currently selected font (0.75 times font height for top & bottom, and 1.25 times font height for left & right). This margin is respected during the auto-positioning of any control that lacks an explicit X or Y coordinate. The margin is also used by the first use of the "Gui Show" command to calculate the window's size (if no explicit size was given).~~

~~For its icon, a GUI window uses the tray icon that was in effect at the time the window was created. Thus, to have a different icon, change the tray icon before creating the window. Example: Menu, Tray, Icon, MyIcon.ico~~

~~Checkboxes, Radio buttons, and GroupBoxes for which a non-default text color was specified will take on Classic Theme appearance on Windows XP and beyond. If this is undesirable, avoid using a non-default text color for these control types.~~

~~Use GuiControl and GuiControlGet to operate upon individual controls in a GUI window.~~

~~Any script that uses the GUI command anywhere is automatically persistent. In v1.0.20+, it is also single instance unless the #SingleInstance directive has been used to override that.~~

Related

[GuiControl](#), [GuiControlGet](#), [Menu](#), [Control](#), [ControlGet](#), [SplashImage](#), [MsgBox](#), [FileSelectFile](#), [FileSelectFolder](#)

Examples

```
; Example #1. Simple input box.
Gui, add, text,, First name.
Gui, add, text,, Last name.
Gui, add, edit, vFirstName ym , The ym option starts a new column of controls.
Gui, add, edit, vLastName
Gui, add, button, default, OK , The label ButtonOK (if it exists) will be run when the button is pressed.
Gui, show,, Simple Input Example
return , End of auto execute section. The script is idle until the user does something.

GuiClose:
ButtonOK:
Gui, submit , Save the input from the user to each control's associated variable.
MsgBox You entered "%FirstName% %LastName%".
ExitApp
```

```
; Example #2. Tab control.
Gui, add, tab,, First Tab|Second Tab|Third Tab
Gui, add, checkbox, vMyCheckbox, Sample checkbox
Gui, tab, 2
Gui, add, radio, vMyRadio, Sample radio1
Gui, add, radio,, Sample radio2
Gui, tab, 3
Gui, add, edit, vMyEdit r5 , r5 means 5 rows tall.
Gui, tab , i.e. subsequently added controls will not belong to the tab control.
Gui, add, button, default xm, OK , xm puts it at the bottom left corner.
Gui, show
return

ButtonOK:
GuiClose:
GuiEscape:
Gui, submit
MsgBox You entered: `n%MyCheckbox%`n%MyRadio%`n%MyEdit%
ExitApp
```

~~Example #3. ListBox containing files in a directory.~~

~~Gui, Add, Text,, Pick a file to launch from the list below.
To cancel, press ESCAPE or close this window.~~

~~Gui, Add, ListBox, vMyListBox gMyListBox w640 r10~~

~~Gui, Add, Button, Default, OK~~

~~Loop, C:*.* , Change this folder and wildcard pattern to suit your preferences.~~

~~+ GuiControl, MyListBox, %A_LoopFileFullPath%~~

~~+ Gui, Show~~

~~return~~

MyListBox:

~~If A_GuiControlEvent <> DoubleClick~~

~~return~~

~~; Otherwise, the user double clicked a list item, so treat that the same as pressing OK.~~

~~; So fall through to the next label.~~

ButtonOK:

~~GuiControlGet, MyListBox , Retrieve the ListBox's current selection.~~

~~MsgBox, 4,, Would you like to launch the file or document below?
%MyListBox%~~

~~If MsgBox, No~~

~~return~~

~~; Otherwise, try to launch it.~~

~~Run, %MyListBox%, UseErrorLevel~~

~~If ErrorLevel = ERROR~~

~~MsgBox Could not launch the specified file. Perhaps it is not associated with anything.~~

~~return~~

GuiClose:

GuiEscape:

ExitApp

~~Example #4: On screen display (OSD).~~

~~CustomColor = EEA99 , Can be any RGB color (it will be made transparent below).~~

Gui, color, %CustomColor%

Gui, font, s24

~~Gui, add, text, vMyText cLime, XXXXX YYYYY , XX & YY serve to auto size the window.~~

Gui, show, x0 y400

~~WinWait, A ; Set the "last found" window for use with the next two commands.~~

~~; Make all pixels of this color transparent and make the text itself translucent (150).~~

WinSet, TransColor, %CustomColor% 150

WinSet, AlwaysOnTop, On

~~; Remove the borders. Due to a quirk in Windows, this must be done after transparency.~~

Gui, Caption

SetTimer, UpdateOSD, 500

return

```
UpdateOSD:  
MouseGetPos, MouseX, MouseY  
GuiControl,, MyText, X%MouseX%, Y%MouseY%  
return
```

```
Example #5: Simple image viewer. This script requires v1.0.24.
```

```
Gui, !Resize  
Gui, Add, Button, default, &Load New Image  
Gui, Add, Radio, y+15 x+10 vRadio checked, Load &actual size  
Gui, Add, Radio, y+15 x+10, Load to &fit screen  
Gui, Add, Pic, xm vPic  
Gui, Show  
return
```

```
ButtonLoadNewImage:
```

```
FileSelectFile, file,,, Select an image., Images (*.gif, *.jpg, *.bmp, *.png, *.tif, *.ico, *.cur, *.ani, *.exe, *.dll)
```

```
if file
```

```
    return
```

```
Gui, Submit, NoHide , Save the values of the radio buttons.
```

```
if Radio = 1 , Display image at its actual size.
```

```
+
```

```
    width = 0
```

```
    height = 0
```

```
+
```

```
else , Second radio is selected. Resize the image to fit the screen.
```

```
+
```

```
    width = %A_ScreenWidth%
```

```
    width = 20 ; Allow room for borders and margins inside.
```

```
    Height = 1 , Keep aspect ratio seems best.
```

```
+
```

```
GuiControl,, Pic, *w%width% *h%height% %file%
```

```
Gui, Show, xCenter y0 AutoSize, %file% ; Resize the window to match the picture size.
```

```
return
```

```
GuiClose:
```

```
ExitApp
```

```
# GuiControl
```

Makes a variety of changes to a control in a GUI window.

GuiControl, Sub-command, ControlID [, Param3]

Parameters

Sub-command	See list below.
ControlID	If the target control has an associated variable, specify the variable's name as the <i>ControlID</i> (this method takes precedence over the ones described next). For this reason, it is usually best to assign a variable to any control that will later be accessed via GuiControl or GuiControlGet, even if that control is not an input-capable type (such as GroupBox or Text). <i>Otherwise, ControlID can be either ClassNN (the classname and instance number of the control) or the name/text of the control, both of which can be determined via Window Spy. When using name/text, the matching behavior is determined by SetTitleMatchMode. Note: a picture control's file name (as it was specified at the time the control was created) may be used as its ControlID.</i>
Param3	This parameter is omitted except where noted in the list of sub-commands below.

ErrorLevel

[ErrorLevel](#) is set to 1 if the specified window/control does not exist or some other problem prevented the command from working. Otherwise, it is set to 0.

Sub-commands

(Blank): Leave *Sub-command* blank to put new contents into the control via *Param3*. All control types are self-explanatory except the following:

Picture: *Param3* should be the filename of the new image to load (see [Gui](#) for supported file types). ~~In v1.0.24+, one or more of the following options may be specified immediately in front of the filename: *wN (width N), *hN (height N), and *IconN (icon number N). In the following example, the second icon is loaded with a width of 100 and an automatic height to "keep aspect ratio": GuiControl,,MyPic,*icon2 *w100 *h-1 C:\My Application.exe. Specify *w0 *h0 to use the image's actual width and height. If *w and *h are omitted, the image will be scaled to fit the current size of the control. Note: Use only one space or tab between the final option and the filename itself; any other spaces and tabs are treated as part of the filename.~~

Edit: Any linefeeds (`n) in *Param3* that lack a preceding carriage return (`r) are automatically translated to CR+LF (`r`n) to make them display properly. However, this is usually transparent because the "Gui Submit" and "GuiControlGet OutputVar" commands will automatically undo this translation by replacing CR+LF with LF (`n).

Hotkey: *Param3* can be blank to clear the control, or a set of modifiers with a key name. Examples: ^lc, ^NumPad1, +Home. The only modifiers supported are ^ (Control), ! (Alt), and + (Shift). See the key list for available key names.

Checkbox: *Param3* can be 0 to uncheck the button, 1 to check it, or -1 to give it a gray checkmark. Otherwise, *Param3* is assumed to be the control's new caption/text. See [Text](#) below for how to override this behavior.

Radio: Same as Checkbox above. However, if the radio button is being checked (turned on) and it is a member of a multi radio group, the other radio buttons in its group will be automatically unchecked.

Slider/Progress: *Param3* should be the new position of the slider/bar. If a *Param3*'s first character is a plus sign, the number will be assumed to be an offset from the current position. For example, +10 would increase the position by 10 and -10 (plus minus ten) would decrease it by 10.

~~Tab/DropDownList/ComboBox/ListBox: Param3 should contain a pipe delimited list of entries to be appended at the end of the control's list. To replace (overwrite) the list instead, include a pipe as the first character (e.g. |Red|Green|Blue). To have one of the entries pre-selected, include two consecutive pipe characters after it (e.g. Red|Green||Blue). The rest of this paragraph applies only to Tab controls. A tab's sub-controls stay associated with their original tab number; that is, they are never associated with their tab's actual display name. For this reason, renaming or deleting a tab will not change the tab number to which the sub-controls belong. For example, if there are three tabs "Red|Green|Blue" and the second tab is deleted by means of "GuiControl, MyTab, |Red|Blue", the sub-controls originally associated with Green will now be associated with Blue.~~

~~**Text** [v1.0.22+]: Behaves the same as the above except for radio buttons and checkboxes, in which case Param3 is treated as the new text/caption even if it is -1, 0, or 1.~~

~~**Move**: Moves and/or resizes the control. Specify one or more of the following option letters in Param3: X (the x-coordinate relative to the GUI window's client area, which is the area not including title bar, menu bar, and borders), Y (the y-coordinate), W (Width), H (Height). Example: x10 y20 w200 h100~~

~~**Focus**: Sets keyboard focus to the control.~~

~~**Enable or Disable**: Enables or disables (grays out) the control. For Tab controls, this will also enable or disable all of the tab's sub-controls. However, any sub-control explicitly disabled via "GuiControl Disable" will remember that setting and thus remain disabled even after its Tab control is re-enabled.~~

~~**Show or Hide**: Shows or hides the control. For Tab controls, this will also show or hide all of the tab's sub-controls.~~

~~**Choose, ControlID, N**: Sets the selection in a ListBox, DropDownList, ComboBox, or Tab control to be the Nth entry. N should be 1 for the first entry, 2 for the second, etc (if N is not an integer, the ChooseString method described below will be used instead). Unlike Control Choose, this sub-command will not trigger any g-label associated with the control unless N is preceded by a pipe character. Example: GuiControl, Choose, myListbox, |3.~~

~~To additionally cause a finishing event to occur (a double click in the case of ListBox), include two leading pipes instead of one (this is not supported for Tab controls).~~

~~**ChooseString, ControlID, String**: Sets the selection (choice) in a ListBox, DropDownList, ComboBox, or Tab control to be the entry whose leading part matches String. The search is not case sensitive. For example, if a the control contains the item "UNIX Text", specifying the word unix (lowercase) would be enough to select it. The pipe and double-pipe prefix are also supported (see "Choose" above details).~~

~~**Font** [v1.0.23+]: Changes the control's font to the typeface, size, color, and style currently in effect for its window. That font can be changed beforehand by means such as this example: Gui, Font, s18 cRed Bold, Verdana~~

~~**+/-Option1 +/- Option2 ...** [v1.0.21+]: Add or remove various options and styles. All of the options described in Gui Add are recognized. In the following example, the AltSubmit option is enabled but control's g-label is removed:~~

~~GuiControl, +AltSubmit -g, myListbox~~

~~In the next example, the OK button is made the new default button:~~

~~GuiControl, +Default, OK~~

~~Although styles and extended styles are also recognized, many of them cannot be applied or removed after a control has been created. Attempts to do so will silently have no effect, and in most such cases, ErrorLevel will be set to 0 (indicating "no error") rather than 1.~~

Remarks

To operate upon a window number other than the default (see below), include a number followed by a colon in front of the sub-command as in these examples:

GuiControl, 2:Show, MyButton

GuiControl, 2:, myListbox, Item1|Item2

A GUI [thread](#) is defined as any thread launched as a result of a GUI action. GUI actions include selecting an item from a GUI window's menu bar, or triggering one of its g-labels (such as by pressing a button).

The default window number for a GUI thread is that of the window that launched the thread. Non-GUI threads use 1 as their default.

Related

[Gui](#), [GuiControlGet](#), [Control](#)

Examples

```
GuiControl,, MyListBox, |Red|Green|Blue ; Replace the current list with a new list.  
GuiControl,, MyEdit, New text line 1.`nNew text line 2.  
GuiControl,, MyRadio2, 1 ; Turn on this radio button and turn off all the others in its group.  
GuiControl, Move, OK, x100 y200 ; Move the OK button to a new location.  
GuiControl, Focus, LastName ; Set keyboard focus to the control whose variable or text is "LastName".
```

#GuiControlGet [v1.0.20+]

Retrieves various types of information about a control in a GUI window.

```
GuiControlGet, OutputVar [, Sub-command, ControlID, Param4]
```

Parameters

OutputVar	The name of the variable in which to store the result. If the command cannot complete (see ErrorLevel below), this variable is made blank.
Sub-command	See list below:
ControlID	If blank or omitted, it defaults to the control whose associated output variable is <i>OutputVar</i> . If the target control has an associated variable, specify the variable's name as the <i>ControlID</i> (this method takes precedence over the ones described next). For this reason, it is usually best to assign a variable to any control that will later be accessed via GuiControl or GuiControlGet, even if that control is not an input-capable type (such as GroupBox or Text). Otherwise, <i>ControlID</i> can be either ClassNN (the classname and instance number of the control) or the name/text of the control, both of which can be determined via Window Spy. When using name/text, the matching behavior is determined by SetTitleMatchMode. Note: a picture control's file name (as it was specified at the time the control was created) may be used as its <i>ControlID</i> .
Param4	This parameter is omitted except where noted in the list of sub commands below.

ErrorLevel

ErrorLevel is set to 1 if the specified window/control does not exist or some other problem prevented the command from working. Otherwise, it is set to 0.

Sub-commands

(Blank): Leave Sub-command blank to retrieve the control's contents. All control types are self-explanatory except the following:

Picture: Retrieves the picture's file name as it was originally specified when the control was created. This name does not change even if a new picture file name is specified.

Edit: Any line breaks in the text will be represented as plain linefeeds (`n) rather than the traditional CR+LF (`r`n) used by non-GUI commands such as ControlGetText and ControlSetText.

Hotkey: Retrieves a blank value if there is no hotkey in the control. Otherwise it retrieves the modifiers and key name. Examples: ^!C, ^Home, +^NumpadHome.

Checkbox/Radio: Retrieves 1 if the control is checked, 0 if it is unchecked, or -1 if it has a gray checkmark. To retrieve the control's text/caption instead, specify the word Text for Param4. Note: Unlike the Gui Submit command, radio buttons are always retrieved individually, regardless of whether they are in a radio group.

Slider/Progress: Retrieves the current position of the slider/bar.

DropDownList/ComboBox/ListBox/Tab control: Retrieves the text of the currently selected item/tab (or its position if the control has the AltSubmit property). For a ComboBox, if there is no selected item, the text in the control's edit field is retrieved instead.

Note: To unconditionally retrieve any control's text/caption rather than its contents, specify the word Text for Param4.

Pos: Retrieves the position and size of the control. The position is relative to the GUI window's client area, which is the area not including title bar, menu bar, and borders. The information is stored in four variables whose names all start with OutputVar.

Example:

ControlGet, MyEdit, Pos

MsgBox The X coordinate is %MyEditX%. The Y coordinate is %MyEditY%. The width is %MyEditW%. The height is %MyEditH%.

Focus: Retrieves the control identifier (ClassName+NN) for the control that currently has keyboard focus. Since the specified GUI window must be active for one of its controls to have focus, OutputVar will be made blank if it is not active. Example:

ControlGet, focused_control, focus

Enabled: Retrieves 1 if the control is enabled or 0 if it is disabled.

Visible: Retrieves 1 if the control is visible or 0 if it is hidden.

Remarks

To operate upon a window number other than the default (see below), include a number followed by a colon in front of the sub-command as in these examples:

GuiControlGet, MyEdit, 2:

GuiControlGet, MyEdit, 2:Pos

GuiControlGet, Outputvar, 2:Focus

A GUI thread is defined as any thread launched as a result of a GUI action. GUI actions include selecting an item from a GUI window's menu bar, or triggering one of its g-labels (such as by pressing a button).

The default window number for a GUI thread is that of the window that launched the thread. Non-GUI threads use 1 as their default.

Related

[Gui](#), [GuiControl](#), [ControlGet](#)

Examples

~~GuiControlGet, MyEdit~~

~~GuiControlGet, CtrlContents,, MyEdit~~; Same as the above except uses a non-default output variable.

~~GuiControlGet, MyCheckbox1~~; Retrieves 1 if it is checked, 0 if it is unchecked.

~~GuiControlGet, MyCheckbox1,,, Text~~; Retrieves the caption/text of the checkbox.

~~GuiControlGet, Pic, Pos, Static4~~; The position/size will be stored in PicX, PicY, PicW, and PicH

#IfMsgBox

Checks which button was pushed by the user during the most recent [MsgBox](#) command.

IfMsgBox, ButtonName

Parameters

ButtonName

Can be any of the following strings to represent which button the user pressed in the most recent [MsgBox](#) command: Yes, No, OK, Cancel, Abort, Ignore, Retry, or Timeout.
The Timeout string is returned if the [MsgBox](#) timed out.

Remarks

None

Related

[MsgBox](#)

Example

```
MsgBox, 4, , Would you like to continue?, 5 ; 5-second timeout.  
If MsgBox, No, Return ; User pressed the "No" button.  
If MsgBox, Timeout, Return ; i.e. Assume "No" if it timed out.  
; Otherwise, continue:  
...
```

#InputBox

Displays an input box to ask the user to enter a string.

```
InputBox, OutputVar [, Title, Prompt, HIDE, Width, Height, X, Y, Font, Timeout, Default]
```

Parameters

OutputVar	The name of the variable in which to store the text entered by the user.
Title	The title of the input box. If blank or omitted, it defaults to the name of the script.
Prompt	The text of the input box, which is usually a message to the user indicating what kind of input is expected.
HIDE	If this parameter is the word HIDE, the user's input will be masked, which is useful for passwords.
Width	If this parameter is blank or omitted, the starting width of the window will be 375.
Height	If this parameter is blank or omitted, the starting height of the window will be 189.
X, Y	The X and Y coordinates of the window (use 0,0 to move it to the upper left corner of the desktop). If either coordinate is blank or omitted, the dialog will be centered in that dimension. Either coordinate can be negative to position the window partially or entirely off the desktop.
Font	Not yet implemented (leave blank). In the future it might accept something like verdana:8
Timeout (v1.0.09)	(optional) Timeout in seconds (can contain a decimal point). If this value exceeds 2147483 (24.8 days), it will be set to 2147483. After the timeout has elapsed, the InputBox window will be automatically closed and ErrorLevel will be set to 2. <i>OutputVar</i> will still be set to what the user entered.

+)	
Default (v1.0.09	A string that will appear in the InputBox's edit field when the dialog first appears. The user can change it by backspacing or other means.
+	

ErrorLevel

See below.

Remarks

The dialog allows the user to enter text and then press OK or CANCEL. The user can resize the dialog window by dragging its borders.

If the user presses the CANCEL button: [ErrorLevel](#) is set to 1 and *OutputVar* to the value entered. This allows the CANCEL button to perform a function other than CANCEL should the script designer wish it.

If the dialog times out, [ErrorLevel](#) will be set to 2.

Related

[Input](#), [MsgBox](#), [FileSelectFile](#), [FileSelectFolder](#), [SplashTextOn](#), [ToolTip](#)

Example

```
InputBox, password, Enter Password, (your input will be hidden), hide
InputBox, UserInput, Please enter a phone number., , 640, 480
if ErrorLevel <> 0
    MsgBox, CANCEL was pressed.
else
    MsgBox, You entered "%UserInput%"
```

MsgBox

Displays a simple message box with one or more buttons and with optional timeout.

MsgBox, Text

MsgBox [, Options, Title, Text, Timeout]

Parameters

Text	If all the parameters are omitted, the MsgBox will display the text "Press OK to continue." Otherwise, this parameter is the text displayed inside the message box to instruct the user what to do, or to present information. Escape sequences can be used to denote special characters. For example, `n indicates a "newline" character, which ends the current line and begins a new one. Thus, using text1`n`ntext2 would create a blank line between text1 and text2.
Options	Indicates the type of message box and the possible button combinations. If blank or omitted, it defaults to 0. See remarks for other allowed values. This parameter will not be recognized if it is contained in a variable reference such as %option%. Instead, use a literal numeric value.
Title	The title of the message box window. If omitted or blank, it defaults to the name of the script (without path).
Timeout	(optional) Timeout in seconds (can contain a decimal point). If this value exceeds 2147483 (24.8 days), it will be set to 2147483. After the timeout has elapsed the message box will be automatically closed and the IfMsgBox command will see the value TIMEOUT.

Remarks

This command has smart comma handling, so it is usually not necessary to [escape](#) commas in the *Text* parameter.

The *Options* parameter can be a combination of the following values.

Function	Value
OK (i.e. just an OK button is displayed)	0
OK/Cancel	1
Abort/Retry/Ignore	2
Yes/No/Cancel	3
Yes/No	4
Retry/Cancel	5
Icon Hand (stop/error)	16

Icon Question	32
Icon Exclamation	48
Icon Asterisk (info)	64
Make 2nd button the default	256
Make 3rd button the default	512
System Modal (always on top)	4096
Task Modal	8192

For example, to specify a SYSTEMMODAL box with the YES/NO buttons the *Options* value would be 4096+4 (4100).

To determine which button the user pressed in the most recent MsgBox, use the [IfMsgBox](#) command.

The text on the buttons can be changed by following [this example](#).

~~The windows shown by this command are a built-in feature of Windows.~~ As such, the X button (close) in the title bar is enabled only when certain buttons are present in the dialog: If there is only an OK button, clicking the X button is the same as pressing OK. Otherwise, the X button is disabled unless there is a Cancel button, in which case clicking the X is the same as pressing Cancel. *Pressing Escape on a window without a X still closes the window as if a fictional Cancel button were pressed.*

~~Pressing Control C while a MsgBox dialog is active will copy its text to the clipboard. This applies to all MsgBoxes, not just those produced by AutoHotkey.~~

In headless mode (when there is no graphical display), MsgBox falls back to Echo.

Related

[IfMsgBox](#), [InputBox](#), [FileSelectFile](#), [FileSelectFolder](#), [ToolTip](#)

Example

```
MsgBox, This is the 1-param method. Commas, do, not, need to be escaped.
MsgBox, 4, , This is the 3-param method, non-escaped commas ok.`n`nContinue?
IfMsgBox, No
    return
MsgBox, 4, , This MsgBox will time out in 5 seconds. Continue?, 5
IfMsgBox, Timeout
    MsgBox, The previous MsgBox timed out.
else IfMsgBox, No
    return
```

#Progress / SplashImage [v1.0.15+]

Creates or modifies a window to display a progress bar or an image.

SplashImage, Off

SplashImage [, ImageFile, Options, SubText, MainText, WinTitle, FontName, FutureUse]

Progress, Off

Progress, Param1 [, SubText, MainText, WinTitle, FontName, FutureUse]

Parameters

ImageFile	The name of the BMP, GIF, or JPG image to display (to display other file formats such as PNG, TIF, and ICO, consider using the Gui command to create a window containing a picture control). <i>ImageFile</i> is assumed to be in %A_WorkingDir% if an absolute path isn't specified. If <i>ImageFile</i> and <i>Options</i> are blank and the window already exists, its image will be unchanged but its text will be updated to reflect any new strings provided in <i>SubText</i> , <i>MainText</i> , and <i>WinTitle</i> . For newly created windows, if <i>ImageFile</i> is blank or there is a problem loading the image, the window will be displayed without the picture.
Param1	If the progress window already exists: If <i>Param1</i> is an integer between 0 and 100, its bar's position (percentage) is changed to that value. If <i>Param1</i> is blank, its bar position will be unchanged but its text will be updated to reflect any new strings provided in <i>SubText</i> , <i>MainText</i> , and <i>WinTitle</i> . In both of these modes, if the window doesn't yet exist, it will be created with the defaults for all options. If the progress window does not exist: A new progress window is created (replacing any old one), and <i>Param1</i> is a string of zero or more options from the list below.
Options	A string of zero or more options from the list below.
SubText	The text to display below the image or bar indicator. Although word wrapping will occur, to begin a new line explicitly, use linefeed (`n). To set an existing window's text to be blank, specify %A_Space%. For the purpose of auto-calculating the window's height, blank lines can be reserved in a way similar to <i>MainText</i> below.
MainText	The text to display above the image or bar indicator (its font is semi bold). Although word wrapping will occur, to begin a new line explicitly, use linefeed (`n). If blank or omitted, no space will be reserved in the window for <i>MainText</i> . To reserve space for single line to be added later, or to set an existing window's text to be blank, specify %A_Space%. To reserve extra lines beyond the first, append one or more linefeeds (`n). Once the height of <i>MainText</i> 's control area has been set, it cannot be changed without recreating the window.
WinTitle	The title of the window. If omitted and the window is being newly created, the title defaults to the name of the script (without path). If the B (borderless) option has been specified, there will be no visible title bar but the window can still be referred to by this title in commands such as <i>WinMove</i> .

FontName	The name of the font to use for both <i>MainText</i> and <i>SubText</i> . The font table lists the fonts included with the various versions of Windows. If unspecified or if the font cannot be found, the system's default GUI font will be used. See the options section below for how to change the size, weight, and color of the font.
FutureUse	This parameter should always be omitted.

Window Size, Position, and Behavior

A: The window will not be always on top.

B: Borderless: The window will have no border and no title bar. To have a border but no title bar, use **B1** for a thin border or **B2** for a dialog-style border.

M: The window will be movable by the user (except if borderless). To additionally make the window resizable (by means of dragging its borders), specify **M1**. To additionally include a system menu and a set of minimize/maximize/close buttons in the title bar, specify **M2**.

T: The window will be given a button in the task bar and it will be unowned. Normally, there is no button because the window is owned by the script's main window.

Hn: Specify for **n** the height of the window's client area (which is the area not including title bar and borders). If unspecified, the height will be calculated automatically based on the height of the image/bar and text in the window.

Wn: Specify for **n** the width of the window's client area. If unspecified, the width will be calculated automatically for **SplashImage** (if there is an image). Otherwise, it will default to 300.

Xn: Specify for **n** the x coordinate of the window's upper left corner. If unspecified, the window will be horizontally centered on the screen.

Yn: Specify for **n** the y coordinate of the window's upper left corner. If unspecified, the window will be vertically centered on the screen.

Layout of Objects in the Window

Cxy: Centered: If this option is absent, both *SubText* and *MainText* will be centered inside the window. Specify 0 for *x* to make *SubText* left justified. Specify a 1 to keep it centered. The same is true for *y* except that it applies to *MainText* (*y* can be omitted).
Example: **c10**

ZHn: Height of object: For Progress windows, specify for **n** the thickness of the progress bar (default 20). For **SplashImage** windows, specify for **n** the height to which to scale image. In v1.0.16+, 1 can be used to make the height proportional to the width specified in **ZWn** (i.e. "keep aspect ratio"). If unspecified, the actual image height will be used. In either case, a value of 0 can be specified to omit the object entirely, which allows the window to be used to display only text in custom fonts, sizes, and colors.

ZWn: Width of object (for **SplashImage** windows only). Specify for **n** the width to which to scale the image. In v1.0.16+, 1 can be used to make the width proportional to the height specified in **ZHn** (i.e. "keep aspect ratio"). If unspecified, the actual image width will be used.

ZXn: Specify for **n** the amount of margin space to leave on the left/right sides of the window. The default is 10 except for **SplashImage** windows with no text, which have a default of 0.

ZYn: Specify for **n** the amount of vertical blank space to leave at the top and bottom of the window and between each control. The default is 5 except for **SplashImage** windows with no text, which have a default of 0.

Note: To create a vertical progress bar or to have more layout flexibility, use the Gui command such as this example:

Gui, Add, Progress, Vertical vMyProgress

Gui, Show

return

... later...

~~GuiControl, MyProgress, +10, Move the bar upward by 10 percent. Omit the + to set an absolute position.~~

-

Font Size and Weight

~~FMn: Specify for n the font size for MainText. Default 0, which causes 10 to be used on most systems. This default is not affected by the system's selected font size in Control Panel > Display settings.~~

~~FSn: Specify for n the font size for SubText. Default 0, which causes 8 to be used on most systems.~~

~~WMn: Specify for n the font weight of MainText. The weight should be between 1 and 1000. If unspecified, 600 (semi bold) will be used.~~

~~WSn: Specify for n the font weight of SubText. The weight should be between 1 and 1000 (700 is traditionally considered to be "bold"). If unspecified, 400 (normal) will be used.~~

-

Object Colors

~~A color can be one of the names from the list below or a 6-digit hexadecimal RGB value. For example, specifying cw1A00FF would set a window background color with red component 1A, green component 00, and blue component FF.~~

~~Add a space after each color option if there are any more options that follow it. Example: cbRed ct900000 cwBlue~~

~~CBn: Color of progress bar: Specify for n one of the 16 primary HTML color names or a 6-digit RGB color value. If unspecified, the system's default bar color will be used. Specify the word Default to return to the system's default progress bar color.~~

~~CTn: Color of text: Specify for n one of the 16 primary HTML color names or a 6-digit RGB color value. If unspecified, the system's default text color (usually black) will be used. Specify the word Default to return to the system's default text color.~~

~~CWn: Color of window (background): Specify for n one of the 16 primary HTML color names or a 6-digit RGB color value. If unspecified, the system's color for the face of buttons will be used (specify the word Default to later return to this color). To make the background transparent on Windows 2000/XP and beyond, use WinSet TransColor.~~

-

Color names and RGB values

Black = 000000

Silver = C0C0C0

Gray = 808080

White = FFFFFF

Maroon = 800000

Red = FF0000

Purple = 800080

Fuchsia = FF00FF

Green = 008000

Teal = 00FF00

Olive = 808000

Yellow = FFFF00

Navy = 000080

Blue = 0000FF

Teal = 008080

Aqua = 00FFFF

Remarks

If the first parameter is the word **OFF**, the window is destroyed.

Each script can display up to 10 Progress windows and 10 SplashImage windows. Each window has a number assigned to it at the time it is created. If unspecified, that number is 1 (the first). Otherwise, precede the first parameter with the number of the window followed by a colon. For example, the Progress command with 2:Off would turn off the number 2 Progress window, 2:75 would set its bar to 75%, 2: would change one or more of its text fields, and 2:B would create a new borderless Progress window. Similarly, the SplashImage command with 2:Off would turn off the number 2 SplashImage window, 2: would change one or more of its text fields, and 2:C:\My Images\Picture1.jpg would create a new number 2 SplashImage window.

Upon creation, a window that would be larger than the desktop is automatically shrunk to fit.

A naked progress bar can be displayed by specifying no SubText, no MainText, and including the following options: b zx0 zy0. A naked image can be displayed the same way except that only the B option is needed.

On Windows XP+, if there is a non-Classic theme in effect, the interior of a progress bar may appear as a series of segments rather than a smooth continuous bar. To avoid this, specify a bar color explicitly such as cbBlue.

Use decimal (not hexadecimal) numbers for option letters that need them, except where noted.

Commands such as WinSet and WinMove can be used to change the attributes of an existing window without having to recreate it.

Related

[GUI](#), [SplashTextOn](#), [ToolTip](#)

Examples

```
Progress, b w200, My SubText, My MainText, My Title  
Progress, 50, Set the position of the bar to 50%.  
Sleep, 4000  
Progress, Off
```

```
; Create a window just to display some 10 point Courier text.  
Progress, m2 b fs10 zh0, This is the Text.  
This is a 2nd line.,,, Courier New
```

```
; Create a simple SplashImage window.  
SplashImage, C:\My Pictures\Company Logo.gif
```

```
; Create a borderless SplashImage window with some large text beneath the image.  
SplashImage, C:\My Pictures\Company Logo.gif, b fs10, This is our company logo.  
Sleep, 4000  
SplashImage, Off
```

```
; Here is a working example that demonstrates how a Progress window can be  
; overlaid on a SplashImage to make a professional looking Installer screen.  
IfExist, C:\WINDOWS\system32\ntimage.gif, SplashImage, C:\WINDOWS\system32\ntimage.gif, A,,, Installation  
Loop, %windir%\system32\*.+  
+  
Progress, %a_index%, %a_loopsfilename%, Installing..., Draft Installation  
Sleep, 50
```

```
+  
    IfEqual, a_index, 100, break
```

#SplashTextOn / SplashTextOff

Creates a customizable text popup window.

SplashTextOff

SplashTextOn [, Width, Height, Title, Text]

Parameters

Width	The width in pixels of the Window. Default 200.
Height	The height in pixels of the window. Default 0 (i.e. just the title bar will be shown).
Title	The title of the window. Default empty (blank).
Text	The text of the window. Default empty (blank).

Remarks

To have more control over layout and font name/color/size, use the Progress command with the zh0 option, which omits the bar and displays only text. Example: Progress, zh0 fs18, Some 18 point text to display.

Use the SplashTextOff command to remove an existing splash window.

The splash window is "always on top", meaning that it stays above all other normal windows. To change this, use WinSet, AlwaysOnTop, Off, <insert title of splash window>. WinSet can also make the splash window transparent.

WinMove can be used to reposition and resize the SplashText window after it has been displayed with this command.

Unlike Progress, SplashImage, MsgBox, InputBox, FileSelectFile, and FileSelectFolder, only one SplashText window per script is possible.

If SplashTextOn is used while the splash window is already displayed, the window will be recreated with the new parameter values. However, rather than recreating the splash window every time you wish to change its title or text, better performance can be achieved by using the following, especially if the window needs to be changed frequently:

WinSetTitle, <insert title of splash window>, , NewTitle

ControlSetText, Static1, NewText, <insert title of splash window>

Related

[Progress](#), [SplashImage](#), [ToolTip](#), [MsgBox](#), [InputBox](#), [FileSelectFile](#), [FileSelectFolder](#), [WinMove](#), [WinSet](#)

Example

```
SplashTextOn, , , Just a simple title.  
Sleep, 2000  
SplashTextOn, 400, 300, Clipboard, The clipboard contains: `n%clipboard%  
WinMove, Clipboard, , 0, 0, Move the splash window to the top left corner.  
Msgbox, Press OK to dismiss the SplashText  
SplashTextOff
```

ToolTip

Creates an always-on-top window anywhere on the screen.

```
ToolTip [, Text, X, Y, WhichToolTip]
```

Parameters

Text	If blank or omitted, the tooltip will be hidden. Otherwise, this parameter is the text to display in the tooltip. To create a multi-line tooltip, use the linefeed character (`n) in between each line, e.g. Line1`nLine2
X, Y	The X and Y position of the tooltip relative to the active window (use "CoordMode, ToolTip" to change to screen coordinates). If the coordinates are omitted, the tooltip will be shown near the mouse cursor.
WhichToolTip	Omit this parameter if you don't need multiple tooltips to appear simultaneously. Otherwise, this is a number between 1 and 20 to indicate which tooltip window to operate upon. If unspecified, that number is 1 (the first).

Remarks

If the X & Y coordinates would cause the tooltip to run off screen, it is repositioned to be visible.

The tooltip appears until it is cleared by running the command with no parameters, until the script terminates, or sometimes until it is clicked upon.

Related

[CoordMode](#), [TrayTip](#), [Progress](#), [SplashTextOn](#), [MsgBox](#), [InputBox](#), [FileSelectFile](#), [FileSelectFolder](#)

Example

```
ToolTip, Multiline`nTooltip, 100, 150  
;  
; To have a ToolTip disappear after a certain amount of time  
; without having to use Sleep (which stops the current thread):  
#Persistent  
ToolTip, Timed ToolTip`nThis will be displayed for 5 seconds.  
SetTimer, RemoveToolTip, 5000  
return  
  
RemoveToolTip:  
SetTimer, RemoveToolTip, Off  
ToolTip  
return
```

TrayTip

Creates a balloon message window ~~near the tray icon~~ *Shows a popup, typically at the top right corner of the screen. Multiple notifications can be shown at the same time.*

TrayTip [, Title, Text, Seconds, Options]

Parameters

Title	If all parameters are omitted, any TrayTip window <i>all existing TrayTip windows</i> currently displayed will be removed. Otherwise, this parameter is the title of the window, which can be up to 63 characters long (characters beyond this length are not shown) . If <i>Title</i> is blank, the title line will be entirely omitted from the balloon window, making it vertically shorter.
-------	---

Text	The message to display, which appears beneath <i>Title</i> . Only the first 255 characters of <i>Text</i> will be displayed. Carriage return (`r) or linefeed (`n) may be used to create multiple lines of text. For example: Line1`nLine2 If this parameter is omitted or blank, any TrayTip window currently displayed will be removed.
Seconds	The approximate number of seconds to display the window, after which it will be automatically removed by the OS. Specifying a number less than 10 or greater than 30 will usually cause the minimum (10) or maximum (30) display time to be used instead. If blank or omitted, the minimum time will usually be used. <i>If blank, the default OS-wide configured notification duration will be used. If set to 0, it will never expire. If set to -1, the library default will be used which appears to be 10 seconds (?)</i> . <i>This setting is currently unreliable due to the nature of its handling by the OS.</i> To have precise control over how long the TrayTip is displayed, use the Sleep command followed by TrayTip with no parameters, or use SetTimer as illustrated in the Examples section below.
Options	Specify one of the following numbers to have a small icon appear to the left of <i>Title</i> : 1: Info icon 2: Warning icon 3: Error icon If omitted, it defaults to 0, which is no icon.

Remarks

TrayTip windows cannot be shown if the script lacks a tray icon (via #NoTrayIcon or *Menu, tray, NoIcon*). Similarly, if the following REG_DWORD value exists and has been set to 0, [TrayTip](#) will not function:
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced >> EnableBalloonTips

Windows XP usually plays a sound when displaying a balloon tip. This sound can be disabled by adding 16 to the *Options* parameter.

Related

[ToolTip](#), [SetTimer](#), [SplashTextOn](#), [MsgBox](#), [InputBox](#), [FileSelectFile](#), [FileSelectFolder](#)

Examples

```
TrayTip, MyTitle, Multiline`nText, 20, 17
; To have more precise control over the display time without
; having to use Sleep (which stops the current thread):
#Persistent
TrayTip, Timed TrayTip, This will be displayed for 5 seconds.
SetTimer, RemoveTrayTip, 5000
return
```

```
RemoveTrayTip:  
SetTimer, RemoveTrayTip, Off  
TrayTip  
return  
  
; To have a TrayTip permanently displayed, use a timer to refresh it  
; periodically:  
SetTimer, RefreshTrayTip, 1000  
Gosub, RefreshTrayTip ; Call it once to get it started right away.  
return  
  
RefreshTrayTip:  
TrayTip  
TrayTip, Refreshed TrayTip, This is a more permanent TrayTip.  
return
```

##HotkeyInterval

Along with **#MaxHotkeysPerInterval**, specifies the rate of hotkey activations beyond which a warning dialog will be displayed.

#HotkeyInterval Value

Parameters

Value

The length of the interval in milliseconds.

Remarks

Care should be taken not to make the above too lenient because if you ever inadvertently introduce an infinite loop of keystrokes (via a Send command that accidentally triggers other hotkeys), your computer could become unresponsive due to the rapid flood of keyboard events. An oversimplified example of an infinite loop of keystrokes: ^c::Send, ^c

If you want to do something like the above without triggering an infinite loop, add a \$ to the hotkey definition (e.g. \$/#y::). See hotkey prefixes for details.

If this directive is unspecified in the script, it will behave as though set to 2000.

-

Related

#MaxHotkeysPerInterval

-

Example

#HotkeyInterval 2000 ; This is the default value (milliseconds).
#MaxHotkeysPerInterval 50 ; This is the default value.

-

##HotkeyModifierTimeout

Affects the behavior of hotkey modifiers: CTRL, ALT, WIN, and SHIFT.

#HotkeyModifierTimeout Value

-

Parameters

Value

The length of the interval in milliseconds. The value can be -1 so that it never times out, or 0 so that it always times out.

-

Remarks

This directive **need not** be used if the script has the keyboard hook installed (you can see if your script uses the hook via the "View > Key history" menu item in the main window, or via the KeyHistory command). This is because the hook can keep track of which modifier keys (ALT/CTRL/WIN/SOFT) the user is physically holding down and doesn't need to use the timeout.

To illustrate the effect of this directive, consider this example:

`^!a::Send, abedefg`

When the Send command executes, the first thing it does is release the CTRL and ALT keys so that the characters get sent properly. After sending all the keys, the command doesn't know whether it can safely push back down CTRL and ALT (to match whether the user is still holding them down). But if less than the specified number of milliseconds have elapsed, it will assume that the user hasn't had a chance to release the keys yet and it will thus push them back down to match their physical state. Otherwise, the modifier keys will not be pushed back down and the user will have to release and press them again to get them to modify the same or another key.

The timeout should be set to a value less than the amount of time that the user typically holds down a hotkey's modifiers before releasing them. Otherwise, the modifiers may be restored to the down position (get stuck down) even when the user isn't physically holding them down.

You can ensure that the keyboard hook is installed, thus making the use of this directive unnecessary, by adding the line `#InstallKeyboardHook` anywhere in the script (the hook is not currently supported on Win9x). Alternatively, reducing `SetKeyDelay` to 0 or -1 should help because it makes `SEND` run more quickly.

If this directive is unspecified in a script, it behaves as though set to 50.

-

Related

[GetKeyState](#)

-

Example

`#HotkeyModifierTimeout 200`

-

#Hotstring [v1.0.16+]

Changing `hotstring` options or ending characters.

```
#Hotstring EndChars NewChars  
#Hotstring NewOptions
```

Parameters

EndChars NewChars

Specify the word `EndChars` followed a single space and then the new ending characters. For example:
`#Hotstring EndChars -(){}';"/,.?!`n `t'r`

Since the new ending characters are in effect globally for the entire script -- regardless of where the `EndChars` command appears -- there is no need to specify `EndChars` more than once.

~~The maximum number of ending characters is 100. Characters beyond this length are ignored.~~

To make tab an ending character, include `'t` in the list. To make space an ending character, include it between two other characters in the list (or at the beginning if the list contains less than 2 other characters).

	<p><i>Return key is identified by \r here</i></p>
NewOptions	<p>Specify new options as described in the Hotstrings section. Example: #Hotstring r s k0 c0</p> <p>Unlike <i>EndChars</i> above, the #Hotstring directive is positional when used this way. In other words, entire sections of hotstrings can have different default options as in this example:</p> <pre>::btw::by the way #Hotstring r c ; All the below hotstrings will use "send raw" and will be case sensitive by default. ::al::airline ::CEO::Chief Executive Officer #Hotstring c0 ; Make all hotstrings below this point case insensitive.</pre>

Related

[Hotstrings](#)

##MaxHotkeysPerInterval

Along with #HotkeyInterval, specifies the rate of hotkey activations beyond which a warning dialog will be displayed.

#MaxHotkeysPerInterval Value

-

Parameters

Value

The maximum number of hotkeys that can be pressed in the interval specified by #HotkeyInterval without triggering a warning dialog.

-

Remarks

Care should be taken not to make the above too lenient because if you ever inadvertently introduce an infinite loop of keystrokes (via a Send command that accidentally triggers other hotkeys), your computer could become unresponsive due to the rapid flood of keyboard events. An oversimplified example of an infinite loop of keystrokes: ^c::Send, ^e

If you want to do something like the above without triggering an infinite loop, add a \$ to the hotkey definition (e.g. \$/#y::). See hotkey prefixes for details.

If this directive is unspecified in the script, it will behave as though set to 50.

-

Related

#HotkeyInterval

-

Example

```
#HotkeyInterval 2000 ; This is the default value (milliseconds).  
#MaxHotkeysPerInterval 50 ; This is the default value.
```

-

##MaxThreads

Sets the maximum number of simultaneous threads:

```
#MaxThreads Value
```

-

Parameters

Value	The maximum total number of threads that can exist simultaneously (limit 20).
-------	---

-

Remarks

This setting is global, meaning that it needs to be specified only once (anywhere in the script) to affect the behavior of the entire script.

Although a value of 1 is allowed, it is not recommended because it would prevent new hotkeys from launching whenever the script is displaying a MsgBox or other dialog.

Any hotkey subroutine whose first line is ExitApp, Pause, Edit, Reload, KeyHistory, ListLines, ListVars, or ListHotkeys will always run regardless of this setting.

If this setting is lower than #MaxThreadsPerHotkey, it effectively overrides that setting.

If this directive is unspecified in the script, it will behave as though set to 10.

Related

#MaxThreadsPerHotkey, Threads, #MaxHotkeysPerInterval, #HotkeyInterval, ListHotkeys, #MaxMem

Example

```
#MaxThreads 2
```

##MaxThreadsBuffer [v1.0.08+]

Causes some or all hotkeys to buffer rather than ignore keypresses when their #MaxThreadsPerHotkey limit has been reached.

```
#MaxThreadsBuffer On|Off
```

Parameters

On|Off

On: All hotkey subroutines between here and the next #MaxThreadsBuffer ON directive will buffer rather than ignore presses of their hotkeys whenever their subroutines are at their #MaxThreadsPerHotkey limit.

Off: This is the default behavior. A hotkey press will be ignored whenever that hotkey is already running its maximum number of threads (usually 1, but this can be changed with #MaxThreadsPerHotkey).

Remarks

This directive is rarely used because this type of buffering, which is OFF by default, usually does more harm than good. For example, if you accidentally press a hotkey twice, having this setting ON would cause that hotkey's subroutine to automatically run a second time if its first thread takes less than 1 second to finish (this type of buffer expires after 1 second, by design). Note that AutoHotkey buffers hotkeys in several other ways. It's just that this particular way can be detrimental, thus it is OFF by default.

The main use for this directive is to increase the responsiveness of the keyboard's auto repeat feature. For example, when you hold down a hotkey whose #MaxThreadsPerHotkey setting is 1 (the default), incoming keypresses are ignored if that hotkey subroutine is already running. Thus, when the subroutine finishes, it must wait for the next auto repeat keypress to come in, which might take 50ms or more due to being caught "in between" keystrokes of the auto repeat cycle. This 50ms delay can be avoided by enabling this directive for any hotkey that needs the best possible response time while it is being auto repeated.

As with all # directives, this one should not be positioned in the script as though it were a command (i.e. it is not necessary to have it contained within a subroutine). Instead, position it immediately before or after the first hotkey label you wish to have affected by it.

Related

~~#MaxThreads, #MaxThreadsPerHotkey, Threads, Hotkey, #MaxHotkeysPerInterval, #HotkeyInterval, ListHotkeys~~

Example

```
#MaxThreadsBuffer on
#x..MsgBox, This hotkey will use this type of buffering.
#y..MsgBox, And this one too.
#MaxThreadsBuffer off
#z..MsgBox, But not this one.
```

#MaxThreadsPerHotkey

Sets the maximum number of simultaneous [threads](#) per hotkey.

#MaxThreadsPerHotkey Value

Parameters

Value	The maximum number of threads that can be launched for a given hotkey subroutine (limit 20).
-------	--

Remarks

This setting is used to control how many "instances" of a given [hotkey](#) subroutine are allowed to exist simultaneously. For example, if a hotkey has a max of 1 and it is pressed again while its subroutine is already running, the press will be ignored. This is helpful to prevent accidental double-presses. However, if you wish these keypresses to be buffered rather than ignored -- perhaps to increase the responsiveness of the keyboard's auto-repeat feature -- use [#MaxThreadsBuffer](#).

Unlike [#MaxThreads](#), this setting is **not** global. Instead, position it before the first hotkey label you wish to have affected by it, which will result in all subsequent hotkeys using that value until another instance of this directive is encountered.

Any [hotkey](#) subroutine whose first line is [ExitApp](#), [Pause](#), [Edit](#), [Reload](#), [KeyHistory](#), [ListLines](#), [ListVars](#), or [ListHotkeys](#) will always run regardless of this setting.

The setting of [#MaxThreads](#) -- if lower than this setting -- takes precedence.

If this directive is unspecified in the script, it will behave as though set to 1.

Related

[#MaxThreads](#), [#MaxThreadsBuffer](#), [Threads](#), [Hotkey](#), [#MaxHotkeysPerInterval](#), [#HotkeyInterval](#), [ListHotkeys](#)

Example

```
#MaxThreadsPerHotkey 3
```

##UseHook

Forces the use of the hook to implement all or some hotkeys.

```
#UseHook [On|Off]
```

Hotkey [v1.0.11+]

Creates, modifies, enables, or disables a hotkey while the script is running.

```
Hotkey, KeyName [, Label, Options]
```

Parameters

KeyName	Name of the hotkey's activation key, including any modifier symbols . For example, specify #c for the Win+C hotkey.
---------	---

	<p>If <i>KeyName</i> already exists as a hotkey, that hotkey will be updated with the values of the other parameters below. For this purpose, <i>KeyName</i> is not case sensitive.</p> <p><i>KeyName</i> can also be the name of a hotkey label (i.e. a double-colon label), which will cause that hotkey to be updated with the values of the other parameters below.</p> <p>When specifying an existing hotkey, the order of the modifier symbols must match the order used when the hotkey was created. For example, <code>^!c</code> is not the same as <code>!^c</code> for matching purposes.</p>
Label	<p>The label name whose contents will be executed (as a new thread) when the hotkey is pressed. Both normal labels and hotkey (double-colon) labels can be used.</p> <p>This parameter can be left blank if <i>KeyName</i> already exists as a hotkey, in which case its label will not be changed. This is done to change the hotkey's <i>Options</i> without changing its label.</p> <p>If the label is specified but the hotkey is disabled from a previous use of this command, the hotkey will still be disabled afterward. To turn it on, use "Hotkey, <i>KeyName</i>, On" immediately afterward.</p> <p>This parameter can also be one of the following special values:</p> <ul style="list-style-type: none"> On: The hotkey is re-enabled. Off: The hotkey is disabled. Toggle [v1.0.12+]: The hotkey is set to the opposite state (enabled or disabled). AltTab (and others): These are special Alt-Tab hotkey actions, which are described here.
Options	<p>A string of zero or more of the following letters with optional spaces in between. Example: <code>B0 T5</code></p> <p>B or B0: Specify the letter B to buffer the hotkey as described in <code>#MaxThreadsBuffer</code>. Specify B0 (B with the number 0) to disable this type of buffering.</p> <p>Pn: Specify the letter P followed by the hotkey's thread priority. If the P option is omitted when creating a hotkey, 0 will be used. Requires v1.0.16+</p> <p>Tn: Specify the letter T followed by n the number of threads to allow for this hotkey as described in <code>#MaxThreadsPerHotkey</code>. Example: <code>T5</code></p> <p>If either or both of the B and T option letters are omitted and the hotkey already exists, those options will not be changed. But if the hotkey does not yet exist — that is, it's about to be created by this command — the options will default to those most recently in effect. For example, the instance of <code>#MaxThreadsBuffer</code> that occurs closest to the bottom of the script will be used. If <code>#MaxThreadsBuffer</code> does not appear in the script, its default setting (OFF in this case) will be used.</p>

Remarks

Creating hotkeys via [double-colon labels](#) performs better than using this command because the hotkeys can all be enabled as a batch (rather than one by one) when the script starts. Therefore, it is best to use this command to create only those hotkeys whose key names are not known until after the script has started running. One such case is when a script's hotkeys for various actions are configurable via an [INI file](#).

A given label can be the target of more than one hotkey. If it is known that a label was called by a hotkey, you can determine which hotkey by using [%A_ThisHotkey%](#).

If the script is [suspended](#), newly added/enabled hotkeys will also be suspended until the suspension is turned off (unless they're exempt as described in the [Suspend](#) section).

The [keyboard](#) and/or [mouse](#) hooks will be installed or removed if justified by the changes made by this command.

Once a script has at least one hotkey, it becomes persistent, meaning that [ExitApp](#) rather than [Exit](#) should be used to terminate it. In v1.0.16+, hotkey scripts are also automatically [#SingleInstance](#) unless [#SingleInstance Off](#) has been specified.

Related

[Hotkey](#) [Symbols](#), [#MaxThreadsBuffer](#), [#MaxThreadsPerHotkey](#), [Suspend](#), [Threads](#), [Thread](#), [Gosub](#), [Return](#), [Menu](#), [SetTimer](#)

Examples

Hotkey, ^!z, MyLabel

Hotkey, RCtrl & RShift, AltTab ; Makes RCtrl & RShift operate like Alt-Tab.

Hotkey, #c, On

Hotkey, \$+#c, Off

Hotkey, ^!a, , T5 ; Change the hotkey to allow 5 threads.

#ListHotkeys

Displays the hotkeys in use by the current script, whether their subroutines are currently running, and whether or not they use the [keyboard](#) or [mouse](#) hook.

ListHotkeys

Parameters

None

Remarks

This command is equivalent to selecting the View->Hotkeys menu item in the main window.

Related

[#InstallKeybdHook](#), [#InstallMouseHook](#), [#UseHook](#), [KeyHistory](#), [ListLines](#), [ListVars](#), [#MaxThreadsPerHotkey](#), [#MaxHotkeysPerInterval](#)

Example
ListHotkeys

Reload

Replaces the currently running instance of the script with a new one.

Reload

Parameters

None

Remarks

This command is useful for scripts that are frequently changed. By assigning a hotkey to this command, you can easily restart the script after saving your changes in an editor.

When the script restarts, it is launched in its original working directory (the one that was in effect when it was first launched). In other words, [SetWorkingDir](#) will not change the working directory that will be used for the new instance.

Related

[Edit](#)

Example

`^!r::Reload ; Assign a hotkey to restart the script.`

`##InstallKeyboardHook`

~~Forces the unconditional installation of the keyboard hook.~~

```
#InstallKeyboardHook
```

~~##InstallMouseHook~~

~~Forces the unconditional installation of the mouse hook.~~

```
#InstallMouseHook
```

BlockInput

Disables or enables the user's ability to interact with the computer via keyboard, mouse, and perhaps other input devices.

```
BlockInput, Mode
```

Parameters

Mode	One of the following words: On : The user is prevented from interacting with the computer (mouse and keyboard input has no effect). Off : Input is re-enabled.
------	--

In v1.0.19+, the following additional modes are available (except on Windows 95/98/Me). These modes operate independently of the above two modes. In other words, "BlockInput On" will continue to block input until "BlockInput Off" is used, even if one of the below modes is also in effect.

Send: Input blocking is automatically enabled prior to each use of Send or SendRaw. This prevents any keystrokes physically typed by the user from disrupting the flow of simulated keystrokes. Input blocking is automatically turned back off after the command completes unless it was on beforehand.

Mouse: Input blocking is automatically enabled prior to each use of MouseMove, MouseClick, or MouseClickDrag. This prevents any physical mouse activity by the user from disrupting the simulated mouse events. Input blocking is automatically turned back off after the command completes unless it was on beforehand.

SendAndMouse: A combination of the above two modes.

Default: Turns off both the *Send* and the *Mouse* modes, but does not change the current state of input blocking. For example, if "BlockInput On" is currently in effect, using "BlockInput Default" will not turn it off. This is also true for "BlockInput Send|Mouse|SendAndMouse".

Remarks

Note: In versions prior to 1.0.19, if BlockInput is ON, the ALT keypress cannot be sent by the Send command. In later versions, input blocking is automatically and momentarily disabled whenever an ALT event is sent (then re-enabled afterward).

Note that commands such as [WinMove](#) will still work on Windows 98/Me when BlockInput is enabled. [ControlSend](#) might also work.

Input will be automatically enabled when the script closes.

Related

[Send](#), [MouseMove](#), [MouseClick](#), [MouseClickDrag](#)

Example

```
if A_OSType <> WIN32_WINDOWS ; i.e. it's not Windows 9x.  
    BlockInput, on  
Run, notepad  
WinWaitActive, Untitled - Notepad  
Send, {F5} ; pastes time and date  
BlockInput, off
```

[# ControlSend / ControlSendRaw](#)

Sends simulated keystrokes to a window or control. *This does not work on some windows. Probably not fixable. Doing 'WinActivate' and 'Send' instead should always work regardless of the program.*

ControlSend [, *Control*, *Keys*, *WinTitle*, *WinText*, *ExcludeTitle*, *ExcludeText*]

Note: ControlSendRaw uses the same parameters.

Parameters

Control	Can be either ClassNN (the classname and instance number of the control) or the name/text of the control, both of which can be determined via Window Spy. When using name/text, the matching behavior is determined by SetTitleMatchMode. If this parameter is blank or omitted, the target window's topmost control will be used. In v1.0.08+, if this parameter is ahk_parent, the keystrokes will be sent directly to the control's parent window (see Automating Winamp for an example).
Keys	The sequence of keys to send (see the Send command for details). To send a literal comma, escape it (` ,). The rate at which characters are sent is determined by SetKeyDelay . Unlike the Send command, mouse clicks cannot be sent by ControlSend.
WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the next 3 parameters are omitted, the Last Found Window will be used. If this is the letter A and the next 3 parameters are omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number, specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

ErrorLevel

[ErrorLevel](#) is set to 1 if there was a problem or 0 otherwise.

Remarks

ControlSendRaw (a command available in v1.0.16+) sends the keystrokes in the Keys parameter exactly as they appear rather than translating {Enter} to an ENTER keystroke, ^c to Control-C, etc.

If the *Control* parameter is omitted, this command will attempt to send directly to the target window by sending to its topmost control (which is often the correct one) or the window itself if there are no controls. This is useful if a window does not appear to have any controls at all, or just for the convenience of not having to worry about which control to send to.

By default, modifier keystrokes (Control, Alt, Shift, and Win) are sent as they normally would be by the Send command. This allows command prompt and other console windows to properly detect uppercase letters, control characters, etc. It may also improve reliability in other ways.

[BlockInput](#) should be avoided when using ControlSend against a console window such as command prompt. This is because it might prevent capitalization and modifier keys such as Control from working properly.

However, in some cases these modifier events may interfere with the active window, especially if the user is actively typing during a ControlSend or if the Alt key is being sent (since Alt activates the active window's menu bar). In v1.0.21+, this can be avoided by explicitly sending modifier up and down events as in this example:

ControlSend, Edit1, {Alt down}f{Alt up}, Untitled - Notepad

The value of [SetKeyDelay](#) determines the speed at which keys are sent.

If the target control is an Edit control (or something similar), the following are usually more reliable and faster than ControlSend:

[Control](#), EditPaste, This text will be inserted at the caret position., ControlName, WinTitle

[ControlSetText](#), ControlName, This text will entirely replace any current text., WinTitle

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[SetKeyDelay](#), [Control](#), [ControlGet](#), [ControlGetText](#), [ControlMove](#), [ControlGetPos](#), [ControlClick](#), [ControlSetText](#), [ControlFocus](#), [Send](#), [Automating Winamp](#)

Examples

ControlSend, Edit1, This is a line of text in the notepad window., Untitled

[SetTitleMatchMode](#), 2

ControlSend, , abc, cmd.exe ; Send directly to a command prompt window.

GetKeyState

Checks if a keyboard key or mouse/joystick button is down or up. Also retrieves joystick status.

GetKeyState, OutputVar, KeyName [, Mode]

Parameters

OutputVar

The name of the variable in which to store the retrieved key state, which is either D for down or U for up. The variable will be empty (blank) if the state of the key could not be determined.

~~The items below apply only to joysticks:~~

~~1) For joystick axes such as JoyX [v1.0.10+], OutputVar will be set to a floating point number between 0 and 100 to indicate the joystick's position as a percentage of that axis's range of motion. The format of the number can be changed via SetFormat. This test script can be used to analyze your joystick(s).~~

	<p>2) When KeyName is JoyPOV [v1.0.10+], the retrieved value will be between 0 and 35900. The following approximate POV values are used by many joysticks:</p> <ul style="list-style-type: none"> -1: no angle to report 0: forward POV 9000 (i.e. 90 degrees): right POV 27000 (i.e. 270 degrees): left POV 18000 (i.e. 180 degrees): backward POV
KeyName	<p>This can be just about any single character from the keyboard or one of the key names from the key list, such as a mouse/joystick button. Examples: B, 5, LWin, RControl, Alt, Enter, Escape, LButton, MButton, Joy1. In v1.0.22+, an explicit virtual key code such as vkFF may also be specified. This is useful in the rare case where a key has no name and produces no visible character when pressed. Its virtual key code can be determined by following the steps at the bottom fo the key list page.</p>
Mode	<p>If omitted, the mode will default to that which retrieves the logical state of the key. This is the state that the OS and the active window believe the key to be in, but is not necessarily the same as the physical state. Alternatively, one of these letters may be specified:</p> <p>P – Retrieve the physical state (i.e. whether the user is physically holding it down)</p> <p>T – Retrieve the toggle state (only valid for keys that can be toggled such as Capslock, Numlock, and Scrolllock). A retrieved value of D means the key is "on", while U means it's "off". Retrieving the toggle state might be less reliable on Windows 9x. For example, immediately after the key is pressed, its new state might not be retrieved correctly until after a the display of a window that's associated with the script, such as a MsgBox.</p> <p>This parameter is ignored when retrieving joystick status.</p>

Remarks

Keys are only detected as pressed down when you did press them **after** the script started. In other words, **GetKeyState** does not work as expected at the very beginning of a script.

To wait for a key or mouse/joystick button to achieve a new state, it is usually easier to use [KeyWait](#) instead of a **GetKeyState** loop.

The physical state of a key or mouse button will usually be the same as the logical state unless the keyboard and/or mouse hooks are installed, in which case it will accurately reflect whether or not the user is physically holding down the key or button. You can determine if your script is using the hooks via the **KeyHistory** command or menu item. You can force either or both of the hooks to be installed by adding the **#InstallKeybdHook** and **#InstallMouseHook** directives to the script.

The following script demonstrates how to use **GetKeyState** to convert a joystick into a two-button mouse by remapping its buttons and axis control: [JoystickMouse Script](#)

Related

[KeyWait](#), [Key List](#), [KeyHistory](#), [#InstallKeybdHook](#), [#InstallMouseHook](#)

Examples

; Basic examples:

GetKeyState, state, Shift

GetKeyState, state, CapsLock, T ; D if CapsLock is ON or U otherwise.

```
GetKeyState, state, RButton ; Right mouse button.  
GetKeyState, state, Joy2 ; The second button of the first joystick.  
  
; Remapping example:  
; In this case, the mouse button is kept held down while NumpadAdd  
; is down, which effectively transforms NumpadAdd into a mouse button.  
; This method can also be used to repeat an action while the user is  
; holding down a key or button:  
*NumpadAdd::  
MouseClick, left,,, 1, 0, D ; Hold down the left mouse button.  
Loop  
{  
    Sleep, 10  
    GetKeyState, state, NumpadAdd, P  
    if state = U ; The key has been released, so break out of the loop.  
        break  
    ; ... insert here any other actions you want repeated.  
}  
MouseClick, left,,, 1, 0, U ; Release the mouse button.  
return  
  
; Example: Make joystick button behavior depend on joystick axis position.  
joy2::  
GetKeyState, joyx, JoyX  
if joyx > 75  
    MsgBox Action #1 (button pressed while joystick was pushed to the right)  
else if joyx < 25  
    MsgBox Action #2 (button pressed while joystick was pushed to the left)  
else  
    MsgBox Action #3 (button pressed while joystick was centered horizontally)  
return  
  
; See the Joystick-To-Mouse script for a more elaborate example.
```

#KeyHistory

Displays script info and a history of the 20 most recent keystrokes and mouse clicks.

KeyHistory

Parameters

None

Remarks

This command is equivalent to selecting the "View > Key history" menu item in the main window.

Although the 40 most recent keyboard and mouse events are displayed, because each keystroke or mouse click consists of a down event and an up event, only 20 "complete" events are shown.

This feature is intended to help debug scripts and hotkeys. It can also be used to detect the scan code of a non-standard keyboard key using the steps described at the bottom of the key list page (knowing the scan code allows such a key to be made into a hotkey).

If the script does not have the keyboard hook installed, the KeyHistory window will display only the keyboard events generated by the script itself (i.e. not the user's). If the script does not have the mouse hook installed, mouse button events will not be shown.

The KeyHistory command also reports other information about the script such as whether it is currently using either of the hooks. You can also force the hooks to be installed by adding either or both of the following lines to the script:

```
#InstallKeybdHook  
#InstallMouseHook
```

Related

[#InstallKeybdHook](#), [#InstallMouseHook](#), [ListHotkeys](#), [ListLines](#), [ListVars](#), [GetKeyState](#), [KeyWait](#)

Examples

```
KeyHistory ; Display the history info in a window.
```

KeyWait

Waits for a key or mouse/joystick button to be released or pressed down.

KeyWait, KeyName [, Options]

Parameters

KeyName	This can be just about any single character from the keyboard or one of the key names from the key list , such as a mouse/joystick button. Joystick attributes other than buttons are not supported. An explicit virtual key code such as vkFF may also be specified. This is useful in the rare case where a key has no name and produces no visible character when pressed. Its virtual key code can be determined by following the steps at the bottom fo the key list page.
Options	If this parameter is blank, the command will wait indefinitely for the specified key or mouse/joystick button to be physically released by the user. However, if the keyboard hook is not installed and KeyName is a keyboard key released artificially by means such as the Send command, the key will be seen as having been physically released. The same is true for mouse buttons when the mouse hook is not installed. Options: A string of one or more of the following letters (in any order, with optional spaces in between): D: Wait for the key to be pushed down. L: Check the logical state of the key, which is the state that the OS and the active window believe the key to be in (not necessarily the same as the physical state). This parameter is ignored for joystick buttons. T: Timeout (e.g. T3). The number of seconds to wait before timing out and setting ErrorLevel to 1. This value can be a floating point number such as 2.5, but it should not be a hexadecimal value such as 0x03.

ErrorLevel

[ErrorLevel](#) is set to 1 if the command timed out or 0 otherwise.

Remarks

*Keys are only detected as pressed down when you did press them **after** the script started. In other words, KeyWait does not work as expected at the very beginning of a script, unless you are using the D option.*

The physical state of a key or mouse button will usually be the same as the logical state unless the keyboard and/or mouse hooks are installed, in which case it will accurately reflect whether or not the user is physically holding down the key. You can determine if your script is using the hooks via the [KeyHistory](#) command or menu item. You can force either or both of the hooks to be installed by adding the `#InstallKeybdHook` and `#InstallMouseHook` directives to the script.

If the key or button achieves the specified state, the command will not wait for the timeout (if any) to expire. Instead, it will immediately set [ErrorLevel](#) to 0 and the script will continue executing.

While the command is in a waiting state, new [threads](#) can be launched via [hotkey](#), [custom menu item](#), or [timer](#).

Related

[GetKeyState](#), [Key List](#), [KeyHistory](#), `#InstallKeybdHook`, `#InstallMouseHook`, [ClipWait](#), [WinWait](#)

Examples

```
; Example #1: Basic usage:  
KeyWait, a ; Wait for the A key to be released.  
KeyWait, LButton, D ; Wait for the left mouse button to be pressed down.  
KeyWait, Joy1, D T3 ; Wait up to 3 seconds for the first joystick button to be pressed down.  
KeyWait, LAlt, L ; Wait for the left-alt key to be logically released.
```

```
; Example #2: A simple hotkey:  
~Capslock::  
KeyWait, Capslock ; Wait for user to physically release it.  
MsgBox You pressed and released the Capslock key.  
return
```

```
; Example #3: Remapping a key or mouse button:  
; The left mouse button is kept held down while NumpadAdd is down,  
; which effectively transforms NumpadAdd into the left mouse button.  
*NumpadAdd::  
MouseClick, left,,, 1, 0, D ; Hold down the left mouse button.  
KeyWait, NumpadAdd ; Wait for the key to be released.  
MouseClick, left,,, 1, 0, U ; Release the mouse button.  
return
```

```
; Example #4: Detects when a key has been double-pressed (similar to double-click).  
; KeyWait is used to stop the keyboard's auto-repeat feature from creating an unwanted  
; double-press when you hold down the RControl key to modify another key. It does this by  
; keeping the hotkey's thread running, which blocks the auto-repeats by relying upon  
; #MaxThreadsPerHotkey being at its default setting of 1.  
; Note: There is a more elaborate script to distinguish between single, double, and  
; triple-presses at the bottom of the SetTimer page.  
~RControl::  
if A_PriorHotkey <> ~RControl  
{  
    KeyWait, RControl  
    return  
}  
if A_TimeSincePriorHotkey > 400 ; Too much time between presses, so this isn't a double-press.  
{  
    KeyWait, RControl  
    return  
}
```

```
MsgBox You double-pressed the right control key.  
return
```

Input

Waits for the user to type a string.

Input [, OutputVar, Options, EndKeys, MatchList]

Parameters

OutputVar	The name of the variable in which to store the text entered by the user (by default, artificial input is also captured). If this and the other parameters are omitted, any input in progress in another thread is terminated and its ErrorLevel is set to the word NewInput . By contrast, the ErrorLevel of the current command will be set to 0 if it terminated a prior input or 1 if there was no prior input to terminate. Non-character keys such as PageUp and Escape are not stored. Whitespace characters such as TAB are stored literally. ENTER is stored as linefeed (`n) in v1.0.16+ as carriage return (`r)
Options	<u>A string of zero or more of the following letters (in any order, with optional spaces in between):</u> B: Backspace is ignored. Normally, pressing backspace during an input will remove the most recently pressed character from the end of the string. Note: If the input text is visible (such as in an editor) and the arrow keys or other means are used to navigate within it, backspace will still remove the last character rather than the one behind the caret (insert position). C: Case sensitive. Normally, <i>MatchList</i> is not case sensitive. I: Ignore input generated by AutoHotkey itself, such as the Send command. L: Length limit (e.g. L5). The maximum allowed length of the input. When the text reaches this length, the input will be terminated and ErrorLevel will be set to the word Max unless the text matches one of the <i>MatchList</i> phrases, in which case ErrorLevel is set to the word Match . If unspecified, the length limit is 16383, which is also the absolute maximum. M [v1.0.21+]: Modified keystrokes such as Control A through Control Z are recognized and transcribed if they correspond to real ASCII characters. Consider this example, which recognizes Control C: Transform, CtrlC, Chr, 3 ; Store the character for Ctrl C in the CtrlC var. Input, OutputVar, L1 M if OutputVar = %CtrlC% MsgBox, You pressed Control C. ExitApp

~~Note: The characters Ctrl A through Ctrl Z correspond to "Chr 1" through "Chr 26". Also, the M option might cause some keyboard shortcuts such as Ctrl LeftArrow to misbehave while an Input is in progress.~~

T: Timeout (e.g. T3). The number of seconds to wait before terminating the input and setting [ErrorLevel](#) to the word Timeout. If the input times out, [OutputVar](#) will be set to whatever text the user had time to enter. This value can be a floating point number such as 2.5.

V: Visible. Normally, the user's input is hidden from the system (suppressed). Use this option to have the user's keystrokes sent to the active window.

*****: Wildcard (find anywhere). Normally, what the user types must exactly match one of the *MatchList* phrases for a match to occur. By contrast, this option finds a match more often because it searches the entire length of the input text.

A list of zero or more keys, any one of which terminates the input when pressed (the *EndKey* itself is not written to [OutputVar](#)). When an input is terminated this way, [ErrorLevel](#) is set to the word EndKey followed by a colon and the name of the *EndKey*, e.g. EndKey:: or EndKey:Escape.

The *EndKey* list uses a format similar to the [Send](#) command. For example, specifying {Enter}.{Esc} would cause either ENTER, period ., or ESCAPE to terminate the input.

To use Control, Alt, or Shift as end-keys, specify the left and/or right version of the key, not the neutral version. For example, specify {LControl}{RControl} rather than {Control}.

~~Although modified keys such as Control-C (^c) are not supported, certain keys that require the shift key to be held down — namely punctuation marks such as ?:@&{— are supported in v1.0.14+.~~

~~An explicit virtual key code such as {vkFF} may also be specified. This is useful in the rare case where a key has no name and produces no visible character when pressed. Its virtual key code can be determined by following the steps at the bottom fo the key list page.~~

A comma-separated list of key phrases, any of which will cause the input to be terminated (in which case [ErrorLevel](#) will be set to the word Match). **Any spaces or tabs around the delimiting commas are significant**, meaning that they are part of the match string. For example, if *MatchList* is "ABC , XYZ ", the user must type a space after ABC or before XYZ to cause a match.

Two consecutive commas results in a single literal comma. For example, the following would produce a single literal comma at the end of string: "string1,,,string2". Similarly, the following list contains only a single item with a literal comma inside it: "single,,item".

Because the items in *MatchList* are not treated as individual parameters, the list can be contained entirely within a variable. ~~In fact, all or part of it must be contained in a variable if its length exceeds 16383 since that is the maximum length of any script line. For example, MatchList might consist of %List1%,%List2%,%List3% — where each of the sublists contains a large list of match phrases.~~

ErrorLevel

1 or 0	Whenever the command is used without parameters, ErrorLevel is set to 0 if it successfully terminates a prior input, or 1 if there is no input in progress.
NewInput	The input was interrupted by another thread that used the Input command.
Max	The input reached the maximum allowed length and it does not match any of the items in <i>MatchList</i> .
Timeout	The input timed out.
Match	The input matches one of the items in <i>MatchList</i> .
EndKey:name	One of the <i>EndKeys</i> was pressed to terminate the input. In this case, ErrorLevel contains the word EndKey followed by a colon and the name of the terminating key without braces, e.g. EndKey:Enter, EndKey:Escape, etc.

Remarks

If this command is used while an input is already in progress in another [thread](#), that input will be terminated and its [ErrorLevel](#) will be set to the word `NewInput`. After that (if parameters were given), the new input will commence.

Hotkeys are still in effect while the input is in progress; in other words, pressing a hotkey will have its normal effect. Similarly, [custom menu items](#) will still function and [timed subroutines](#) will still run.

~~When a script first uses this command, the keyboard hook will be installed automatically if it isn't already present. Once that happens, the script becomes persistent, meaning that `ExitApp` should be used to terminate it.~~

Although not as flexible, [hotstrings](#) are generally easier to use than the Input command.

Related

[Hotstrings](#), [InputBox](#), [#InstallKeybdHook](#), [Threads](#), [StringCaseSense](#)

Example

```
; This is a working hotkey example. Since the hotkey has the tilde (~)
; prefix, its own keystroke will pass through to the active window
; (except on Win9x). Thus, if you type [btw (or one of the other match
; phrases) in any editor, the script will automatically perform an
; action of your choice (such as replacing the typed text):

~[:::
Input, UserInput, V T5 L4 C, {enter}.{esc}{tab}, btw, otoh, fl, ahk, ca
if ErrorLevel = Max
{
    MsgBox, You entered "%UserInput%", which is the maximum length of text.
    return
}
if ErrorLevel = Timeout
{
    MsgBox, You entered "%UserInput%" at which time the input timed out.
    return
}
if ErrorLevel = NewInput
    return
IfInString, ErrorLevel, EndKey:
{
    MsgBox, You entered "%UserInput%" and terminated the input with %ErrorLevel%.
    return
}
; Otherwise, a match was found.
SetKeyDelay, -1 ; Most editors can handle the fastest speed.
if UserInput = btw
    Send, {backspace 4}by the way
```

```
else if UserInput = otoh
    Send, {backspace 5}on the other hand
else if UserInput = fl
    Send, {backspace 3}Florida
else if UserInput = ca
    Send, {backspace 3}California
else if UserInput = ahk
    Run, www.autohotkey.com
return
```

Send / SendRaw

Sends simulated keystrokes to the active window.

Send, Keys
SendRaw, Keys

Parameters

Keys

The sequence of keys to send. The rate at which they are sent is determined by [SetKeyDelay](#).

Remarks

Please note that due to a bug in AHK_X11, certain special characters cannot be sent with this command natively. You will have to add a + in front of them to simulate a SHIFT key press. Example: Instead of `Send, Yes{!}` you will have to do `Send, Yes+{!}` or just use [SendRaw](#) instead.

[SendRaw](#) (available in v1.0.16+) sends the keystrokes in the *Keys* parameter exactly as they appear rather than translating {Enter} to an ENTER keystroke, ^c to Control-C, etc.

The "Send" command syntax is similar to that of ScriptIt and the Visual Basic "SendKeys" command. Characters are sent as written with the exception of the following characters:

'!

Sends an ALT keystroke, therefore *Send, This is text!a* would send the keys "This is text" and then press "ALT+a".

N.B. Some programs are very choosy about capital letters and ALT keys, i.e. !A is different than !a. The first says ALT+SHIFT+A, the second is ALT+a. If in doubt, use lowercase!

'+'

Sends a SHIFT keystroke, therefore *Send*, *He!llo!* would send the text "Hello". *Send*, *!+a* would send "ALT+SHIFT+a".

'^'

Sends a CONTROL keystroke, therefore *Send*, *^!a* would send "CTRL+ALT+a".

N.B. Some programs are very choosy about capital letters and CTRL keys, i.e. *^A* is different than *^a*. The first says CTRL+SHIFT+A, the second is CTRL+a. If in doubt, use lowercase!

'#'

Sends a WIN keystroke, therefore *Send*, *#a* would hold down the Windows key and then press the letter "a".

Certain special keys can be sent and should be enclosed in braces:

N.B. Windows does not allow the simulation of the "CTRL-ALT-DEL" combination!

Blind mode

The Blind mode can be enabled with `{Blind}`, which gives the script more control by disabling a number of things that are normally done automatically to make things work as expected. `{Blind}` must be the first item in the string to enable the Blind mode. It has the following effects:

- The Blind mode avoids releasing the modifier keys (Alt, Ctrl, Shift, and Win) if they started out in the down position. For example, the hotkey `+s::Send {Blind}abc` would send ABC rather than abc because the user is holding down Shift.
- Modifier keys are restored differently to allow a Send to turn off a hotkey's modifiers even if the user is still physically holding them down. For example, `^space::Send {Ctrl up}` automatically pushes Ctrl back down if the user is still physically holding Ctrl, whereas `^space::Send {Blind}{Ctrl up}` allows Ctrl to be logically up even though it is physically down.
- `SetStoreCapsLockMode` is ignored; that is, the state of CapsLock is not changed.
- ~~Menu masking is disabled. That is, Send omits the extra keystrokes that would otherwise be sent in order to prevent: 1) Start Menu appearance during Win keystrokes (LWin/RWin); 2) menu bar activation during Alt keystrokes. However, the Blind mode does not prevent masking performed by the keyboard hook following activation of a hook hotkey.~~
- Send does not wait for Win to be released even if the text contains an = keystroke. This would normally be done to prevent Send from triggering the system "lock workstation" hotkey (`win+L`). See Hotkeys for details.

The Blind mode is used internally when [remapping a key](#). For example, the remapping `a::b` would produce: 1) "b" when you type "a"; 2) uppercase "B" when you type uppercase "A"; and 3) Ctrl+B when you type Ctrl+A.

`{Blind}` is not supported by `SendRaw` or `ControlSendRaw`; use `{Blind} (Raw)` instead.

~~The Blind mode is not completely supported by `SendPlay`, especially when dealing with the modifier keys (Ctrl, Alt, Shift, and Win).~~

Send Command	Resulting Keypress
{F1} - {F24}	Function keys. For example: {F12} is the F12 key.
{!}	!
{#}	#

{+}	+
{^}	^
{()}	{
{}}	}
{ENTER}	ENTER key on the main keyboard
{ESCAPE} or {ESC}	ESCAPE
{SPACE}	SPACE (this is only needed for spaces that appear either at the beginning or the end of the string to be sent -- ones in the middle can be literal spaces)
{TAB}	TAB
{BACKSPACE} or {BS}	Backspace
{DELETE} or {DEL}	Delete
{INSERT} or {INS}	Insert
{UP}	Cursor up key on main keyboard
{DOWN}	Cursor down key on main keyboard
{LEFT}	Cursor left key on main keyboard
{RIGHT}	Cursor right key on main keyboard
{HOME}	Home key on main keyboard
{END}	End key on main keyboard
{PGUP}	Page up key on main keyboard
{PGDN}	Page down key on main keyboard
{CapsLock}	CapsLock (using SetCapsLockState is more reliable on NT/2k/XP)
{ScrollLock}	ScrollLock
{NumLock}	NumLock
{CONTROL} or {CTRL}	CONTROL (sends the neutral virtual key but the left scan code)
{LCONTROL} or {LCTRL}	Left CONTROL key (same as CONTROL for Win9x, but on NT/2k/XP it sends the left virtual key rather than the neutral one)
{RCONTROL} or {RCTRL}	Right CONTROL key

{CONTROLDOWN} or {CtrlDown}	Holds the CONTROL key down until {CTRLUP} is sent. XP/2000/NT: To hold down the left or right key instead, use {RCtrl Down} and {RCtrl Up}.
{ALT}	ALT (sends the neutral virtual key but the left scan code)
{LALT}	Left ALT key (same as ALT for Win9x, but on NT/2k/XP it sends the left virtual key rather than the neutral one)
{RALT}	Right ALT key
{ALTDOWN}	Holds the ALT key down until {ALTUP} is sent. XP/2000/NT: To hold down the left or right key instead, use {RAlt Down} and {RAlt Up}.
{SHIFT}	SHIFT (sends the neutral virtual key but the left scan code)
{LSHIFT}	Left SHIFT key (same as SHIFT for Win9x, but on NT/2k/XP it sends the left virtual key rather than the neutral one)
{RSHIFT}	Right SHIFT key
{SHIFTDOWN}	Holds the SHIFT key down until {SHIFTUP} is sent. XP/2000/NT: To hold down the left or right key instead, use {RShift Down} and {RShift Up}.
{LWIN}	Left Windows key
{RWIN}	Right Windows key
{LWINDOW}	Holds the left Windows key down until {LWINUP} is sent
{RWINDOW}	Holds the right Windows key down until {RWINUP} is sent
{APPSKEY}	Windows App key (invokes the right-click or context menu)
{SLEEP}	Computer SLEEP key.
{ASC nnnnn}	# Sends an ALT+nnnnn keypad combination, which can be used to generate special characters that don't exist on the keyboard. To generate ASCII characters, specify a number between 1 and 255. To generate ANSI characters (standard in most languages), specify a number between 128 and 255, but precede it with a leading zero, e.g. {Asc 0133}. In v1.0.13+, to generate Unicode characters, specify a number between 256 and 65535 (without a leading zero). You can also retrieve or store Unicode text on the clipboard via " Transform Unicode ".
{vkXX} or {vkXXscYYY}	# Sends a keystroke that has virtual key XX and scan code YYY. If the scYYY portion is omitted, XX's default scan code will be sent. The values for XX and YYY are hexadecimal and can sometimes be determined from the main window's "View->Key history" menu item. Requires v1.0.14+. Example: Send {vkFFsc159}
{Numpad0} - {Numpad9}	Numpad digit keys (used when Numlock is ON). Example: {Numpad5} is the digit 5.

{NumpadDot}	Numpad Period (used when Numlock is ON)
{NumpadEnter}	Enter key on keypad
{NumpadMult}	Numpad Multiply
{NumpadDiv}	Numpad Divide
{NumpadAdd}	Numpad Add
{NumpadSub}	Numpad Subtract
{NumpadDel}	Delete key on keypad (this key and the following Numpad keys are used when Numlock is OFF)
{NumpadIns}	Insert key on keypad
{NumpadClear}	Clear key on keypad (usually the '5' key).
{NumpadUp}	Cursor up key on keypad
{NumpadDown}	Cursor down key on keypad
{NumpadLeft}	Cursor left key on keypad
{NumpadRight}	Cursor right key on keypad
{NumpadHome}	Home key on keypad
{NumpadEnd}	End key on keypad
{NumpadPgUp}	Page up key on keypad
{NumpadPgDn}	Page down key on keypad
{BROWSER_BACK}	2000/XP Only: Select the browser "back" button
{BROWSER_FORWARD}	2000/XP Only: Select the browser "forward" button
{BROWSER_REFRESH}	2000/XP Only: Select the browser "refresh" button
{BROWSER_STOP}	2000/XP Only: Select the browser "stop" button
{BROWSER_SEARCH}	2000/XP Only: Select the browser "search" button
{BROWSER_FAVORITES}	2000/XP Only: Select the browser "favorites" button
{BROWSER_HOME}	2000/XP Only: Launch the browser and go to the home page
{VOLUME_MUTE}	2000/XP Only: Mute the volume
{VOLUME_DOWN}	2000/XP Only: Reduce the volume

{VOLUME_UP}	2000/XP Only: Increase the volume
{MEDIA_NEXT}	2000/XP Only: Select next track in media player
{MEDIA_PREV}	2000/XP Only: Select previous track in media player
{MEDIA_STOP}	2000/XP Only: Stop media player
{MEDIA_PLAY_PAUSE}	2000/XP Only: Play/pause media player
{LAUNCH_MAIL}	2000/XP Only: Launch the email application
{LAUNCH_MEDIA}	2000/XP Only: Launch media player
{LAUNCH_APP1}	2000/XP Only: Launch user app1
{LAUNCH_APP2}	2000/XP Only: Launch user app2
{PRINTSCREEN}	PRINTSCR
{CTRLBREAK}	Ctrl+break
{PAUSE}	PAUSE
{LButton}, {RButton}, {MButton}, {XButton1}, {XButton2}, {WheelDown}, {WheelUp}	These mouse button events will occur at the mouse cursor's current location. The delay between mouse clicks is determined by SetMouseDelay (not SetKeyDelay). To have control over position and other options, use MouseClick . Requires v1.0.20.

In addition to the 26 English letters, the following letters and symbols are also supported (and possibly others too): €,ƒ,...†‡^‰„šŒŽ‘„“•—~™šœžŸ iƒ£¤¥!§„©¤«¬®—°±²³’µ¶·,¹º»¼½¾¿ÀÁÂÃÅÅÆÇÉÉÈÍÍÐÑÓÓÔÔÖ×ØÙÙÙÙÝÞßàáââääæçééèííðñòóôôö÷øùúûüýþþ

Single keys can also be repeated, e.g.

- Send, {DEL 4} Presses the DEL key 4 times
- Send, {S 30} Sends 30 'S' characters
- Send, +{TAB 4} Presses SHIFT+TAB 4 times

Keys can also be held down or released, e.g.

- Send, {B down}{B up}
- Send, {TAB down}{TAB up}

The [BlockInput](#) command can be used to prevent any keystrokes physically typed by the user from disrupting the flow of simulated keystrokes.

Related

[SetKeyDelay](#), [SetStoreCapslockMode](#), [ControlSend](#), [BlockInput](#), [Hotstrings](#), [WinActivate](#)

Examples

```
Send, Sincerely,{enter}John Smith  
Send, !fs ; Select the File->Save menu (Alt+F followed by S).
```

SetKeyDelay

Sets the delay that will occur after each keystroke sent by [Send](#) and [ControlSend](#).

```
SetKeyDelay [, Delay, PressDuration]
```

Parameters

Delay	Time in milliseconds. Use -1 for no delay at all and 0 for the smallest possible delay. Leave this parameter blank to retain the current <i>Delay</i> .
PressDuration	Certain games and other specialized applications may require a delay inside each keystroke; that is, after the press of the key but before its release. Use -1 for no delay at all (default) and 0 for the smallest possible delay. Omit this parameter to leave the current <i>PressDuration</i> unchanged. <i>Note:</i> <i>PressDuration</i> also produces a delay after any change to the modifier key state (CTRL, ALT, SHIFT, and WIN) needed to support the keys being sent.

Remarks

A short delay (sleep) is automatically and invisibly done after every keystroke sent by [Send](#) or [ControlSend](#). This is done to improve the reliability of scripts because a window sometimes can't keep up with a rapid flood of keystrokes.

~~Due to the granularity of the OS's time keeping system, delays might be rounded up to the nearest multiple of 10. For example, a delay between 1 and 10 (inclusive) is equivalent to 10 on Windows XP (and probably NT & 2k).~~

A delay of 0 internally executes a Sleep(0), which yields the remainder of the script's timeslice to any other process that may need it. If there is none, Sleep(0) will not sleep at all. By contrast, a delay of -1 will never sleep. For better reliability, 0 is recommended as an alternative to -1.

If unset, the default delay is 10. The default *PressDuration* is -1.

The built-in variable **A_KeyDelay** contains the current setting of *Delay*. There is no built-in variable for *PressDuration*.

Related

[Send](#), [ControlSend](#), [SetMouseDelay](#), [SetControlDelay](#), [SetWinDelay](#), [SetBatchLines](#), [MouseClick](#)

Example

`SetKeyDelay, 0`

~~#SetCapsLockState/SetNumLockState/SetScrollLockState~~

~~Sets the state of the Capslock/NumLock/ScrollLock key. Can also force the key to stay on or off.~~

~~`SetCapsLockState, State`
`SetNumLockState, State`
`SetScrollLockState, State`~~

Parameters

~~State~~

~~On: Toggles the key into the "on" state.~~

~~Off: Toggles the key into the "off" state.~~

~~AlwaysOn: Forces the key to be always on (not supported on Windows 9x).~~

~~AlwaysOff: Forces the key to be always off (not supported on Windows 9x).~~

Remarks

~~Keeping a key AlwaysOn or AlwaysOff requires the keyboard hook, which will be automatically installed in such cases.~~

~~These commands are currently not 100% reliable on Windows Me/98/95 due to the limitations of those OSes. As an alternative, a key can be reliably toggled to its opposite state via the Send command. For example: `Send, {Capslock}`~~

-

Related

[SetStoreCapslockMode](#), [GetKeyState](#)

-

Example

[SetNumlockState](#), on
[SetScrollLockState](#), AlwaysOff

#SetStoreCapslockMode

~~Whether to restore the state of CapsLock after a Send.~~

[SetStoreCapslockMode](#), On|Off

-

Parameters

On|Off

~~On: This is the initial setting for all scripts. The CapsLock key will be restored to its former value if Send needed to change it temporarily for its operation.~~

~~Off: The state of the CapsLock key is not restored.~~

-

Remarks

~~This setting is currently not reliable under Windows Me/98/95 due to the limitations of those OSes. This means that the Capslock key will not always be turned off for Send and ControlSend. Even when it is successfully turned off, it might not get turned back on after the keys are sent.~~

~~This command is rarely used because the default behavior is best in most cases.~~

~~Every newly launched hotkey, custom menu item, or timed subroutine starts off fresh with the default setting for this command. That default may be changed by using this command in the auto execute section (top part) of the script.~~

-

Related

~~SetCaps/Num/ScrollLockState, Send, ControlSend~~

-

Example

~~SetStoreCapslockMode, off~~

EnvAdd

Sets a **variable** to the sum of itself plus the given value (can also add or subtract time from a **date-time** value). Synonymous with: `var += value`

`EnvAdd, Var, Value [, TimeUnits]`

`Var += Value [, TimeUnits]`

`Var++`

Parameters

Var	The name of the variable to be operated upon.
Value	Any integer or floating point number.
TimeUnits	If present, this parameter directs the command to add <i>Value</i> to <i>Var</i> , treating <i>Var</i> as a date-time stamp in the YYYYMMDDHH24MISS format and treating <i>Value</i> as the integer or floating point number of units to add (specify a negative number to perform subtraction). <i>TimeUnits</i> can be either Seconds, Minutes, Hours, or Days (or just the first letter of each of these). If <i>Var</i> is an empty variable, the current time will be used in its place. If <i>Var</i> contains an invalid timestamp or a year prior to 1601, or if <i>Value</i> is non-numeric, <i>Var</i> will be made blank to indicate the problem. The built-in variable A_Now contains the current local time in YYYYMMDDHH24MISS format.

Remarks

This command is equivalent to the shorthand style: `Var += Value`

Variables can be increased or decreased by 1 by using `Var++`, `Var--`, `++Var`, or `--Var`.

If either *Var* or *Value* is blank or does not start with a number, it is considered to be 0 for the purpose of the calculation (except when using *TimeUnits*, see above).

If either *Var* or *Value* contains a decimal point, the end result will be a floating point number in the format set by [SetFormat](#).

*For general math operations, you can also resort back to the more modern := operator, which you might know from modern Windows AHK. In this case, you want to omit the percent signs % unless you want to double-deref. This := operator (for "expressions") is much more limited on AHK_X11 though, and also performs poorly. Example usage: my_var := other_var * (16 + 5)*

Related

[EnvSub](#), [EnvMult](#), [EnvDiv](#), [SetFormat](#), [If var is \[not\] type](#), [SetEnv](#), [FileGetTime](#)

Example

```
EnvAdd, MyCount, 2  
MyCount += 2 ; Equivalent to above  
var1 = ; Make it blank so that the below will use the current time instead.  
var1 += 31, days  
MsgBox, %var1% ; The answer will be the date 31 days from now.
```

EnvDiv

Sets a [variable](#) to itself divided by the given value. Synonymous with: var /= value

EnvDiv, Var, Value

Parameters

Var	The name of the variable to be operated upon.
Value	Any integer or floating point number (except zero).

Remarks

This command is equivalent to the shorthand style: Var /= Value

Division by zero will result in an error-message window when the script is loaded (if possible) or during runtime otherwise.

If either *Var* or *Value* is blank or does not start with a number, it is considered to be 0 for the purpose of the calculation.

If either *Var* or *Value* contains a decimal point, the end result will be a floating point number in the format set by [SetFormat](#). Otherwise, the result will be truncated (e.g. 19 divided by 10 will yield 1).

Related

[EnvAdd](#), [EnvSub](#), [EnvMult](#), [SetFormat](#), [If var is \[not\] type](#), [SetEnv](#), bitwise operations (Transform)

Example

`EnvDiv, MyCount, 2`

`MyCount /= 2` ; Equivalent to above

EnvMult

Sets a [variable](#) to itself times the given value. Synonymous with: `var *= value`

`EnvMult, Var, Value`

Parameters

Var	The name of the variable to be operated upon.
Value	Any integer or floating point number.

Remarks

This command is equivalent to the shorthand style: `Var *= Value`

If either *Var* or *Value* is blank or does not start with a number, it is considered to be 0 for the purpose of the calculation.

If either *Var* or *Value* contains a decimal point, the end result will be a floating point number in the format set by [SetFormat](#).

Related

[EnvAdd](#), [EnvSub](#), [EnvDiv](#), [SetFormat](#), [If var is \[not\] type](#), [SetEnv](#), bitwise operations (Transform)

Example

```
EnvMult, MyCount, 2  
MyCount *= 2 ; Equivalent to above
```

EnvSub

Sets a [variable](#) to itself minus the given value (can also compare date-time values). Synonymous with: `var -= value`

```
EnvSub, Var, Value [, TimeUnits]  
Var -= Value [, TimeUnits]  
Var--
```

Parameters

Var	The name of the variable to be operated upon.
Value	Any integer or floating point number.
TimeUnits	If present, this parameter directs the command to subtract <i>Value</i> from <i>Var</i> as though both of them are date-time stamps in the YYYYMMDDHH24MISS format. <i>TimeUnits</i> can be either Seconds, Minutes, Hours, or Days (or just the first letter of each of these). If <i>Value</i> is blank, the current time will be used in its place. Similarly, if <i>Var</i> is an empty variable, the current time will be used in its place. If either <i>Var</i> or <i>Value</i> is an invalid timestamp or contains a year prior to 1601, <i>Var</i> will be made blank to indicate the problem. The built-in variable A_Now contains the current local time in YYYYMMDDHH24MISS format. To determine the elapsed time between two events that are only a few seconds or minutes apart, it is more precise (and often easier) to use A_TickCount as described here .

Remarks

This command is equivalent to the shorthand style: Var -= Value

Variables can be increased or decreased by 1 by using Var++, Var--, ++Var, or --Var.

If either Var or Value is blank or does not start with a number, it is considered to be 0 for the purpose of the calculation (except when using *TimeUnits*; see above).

If either Var or Value contains a decimal point, the end result will be a floating point number in the format set by [SetFormat](#).

Related

[EnvAdd](#), [EnvMult](#), [EnvDiv](#), [SetFormat](#), [If var is \[not\] type](#), [SetEnv](#), [FileGetTime](#)

Example

```
EnvSub, MyCount, 2
```

```
MyCount -= 2 ; Equivalent to above
```

```
var1 = 20050126
```

```
var2 = 20040126
```

```
EnvSub, var1, %var2%, days
```

```
MsgBox, %var1% ; The answer will be 366 since 2004 is a leap year.
```

If/IfEqual/IfNotEqual/IfLess/IfLessOrEqual/IfGreater/IfGreaterOrEqual

Compares a [variable](#) to a value for equality. Synonymous with: if var = value ... if var <> value

IfEqual, var, value (same: if var = value)

IfNotEqual, var, value (same: if var <> value) (*same: if var != value*)

IfGreater, var, value (same: if var > value)

IfGreaterOrEqual, var, value (same: if var >= value)

IfLess, var, value (same: if var < value)

IfLessOrEqual, var, value (same: if var <= value)

Parameters

var	The variable name.
value	A literal string, number, or variable reference (e.g. %var2%). Value can be omitted if you wish to compare <i>var</i> to an empty string (blank).

Remarks

If both *var* and *value* are purely numeric, they will be compared as numbers rather than as strings. Otherwise, they will be compared alphabetically as strings (that is, alphabetical order will determine whether *var* is greater, equal, or less than *value*).

Another command can only appear on the same line as this one if you use the command-name style. In other words, this is valid:

IfEqual, x, 1, Sleep, 1

if x = 1 Sleep 1 ; But this is not.

IfGreater, x, 1, x += 2 ; Nor is this.

IfGreater, x, 1, EnvAdd, x, 2 ; But this is okay.

On a related note, The command "[if var \[not\] between LowerBound and UpperBound](#)" checks whether a variable is between two values. Also, "[if var \[not\] in value1,value2](#)" can be used to check whether a variable's contents exist within a list of values.

Related

[if var in/contains MatchList](#), [if var between](#), [IfInString](#), [Blocks](#), [Else](#)

Example

```
if counter = 1
    Sleep, 1
if MyVar = Test String
    Sleep, 1
if MyVar <> %MyVar2%
{
    Sleep, 1
    Exit
}
else
{
    Sleep, 2
    return
}
```

If var [not] between LowerBound and UpperBound

Checks whether a [variable's](#) contents are numerically or alphabetically between two values (inclusive).

```
if Var between LowerBound and UpperBound  
if Var not between LowerBound and UpperBound
```

Parameters

Var	The variable name whose contents will be checked.
LowerBound	To be within the specified range, <i>Var</i> must be greater than or equal to this string, number, or variable reference.
UpperBound	To be within the specified range, <i>Var</i> must be less than or equal to this string, number, or variable reference.

Remarks

If all three of the parameters are purely numeric, they will be compared as numbers rather than as strings. Otherwise, they will be compared alphabetically as strings (that is, alphabetical order will determine whether *Var* is within the specified range). In that case, "[StringCaseSense On](#)" can be used to make the comparison case sensitive.

Related

[IfEqual/Greater/Less](#), [if var in/contains MatchList](#), [if var is type](#), [IfInString](#), [StringCaseSense](#), [EnvAdd](#), [Blocks](#), [Else](#)

Example

```
if var between 1 and 5  
    MsgBox, %var% is in the range 1 to 5, inclusive.  
  
if var not between 0.0 and 1.0  
    MsgBox %var% is not in the range 0.0 to 1.0, inclusive.  
  
if var between blue and red  
    MsgBox %var% is alphabetically between blue and red.  
  
LowerLimit = 1
```

```
UpperLimit = 10
InputBox, UserInput, Enter a number between %LowerLimit% and %UpperLimit%
if UserInput not between %LowerLimit% and %UpperLimit%
    MsgBox Your input is not within the valid range.
```

#if var is [not] type

Checks whether a variable's contents are numeric, uppercase, etc.

```
if var is type
if var is not type
```

Parameters

var	The variable name.
type	See the remarks below.

Remarks

Supported Types:

integer	True if var is non-empty and contains a purely numeric string without a decimal point. Leading and trailing whitespace is allowed. The string may start with a plus or minus sign.
float	True if var is non-empty and contains a floating point number, that is, a purely numeric string containing a decimal point. Leading and trailing whitespace is allowed. The string may start with a plus sign, minus sign, or decimal point.
number	True if var contains an integer or floating point number, as described above.
digit	True if var is empty or contains only digits, which consist of the characters 0 through 9. Whitespace, plus signs, minus signs, and decimal points are not allowed.
xdigit	Hexadecimal digit: Same as digit except the characters A through F (upper or lower case) are also allowed. Requires v1.0.14+.
alpha	True if var is empty or contains only alphabetic characters. False if there are any digits, whitespace, punctuation, or other non-alphabetic characters anywhere in the string. For example, if var contains a space followed by a letter, it is not considered to be alpha.
upper	True if var is empty or contains only uppercase characters. False if there are any digits, whitespace, punctuation, or other non-uppercase characters anywhere in the string. Requires v1.0.14+.

lower	True if var is empty or contains only lowercase characters. False if there are any digits, whitespace, punctuation, or other non-lowercase characters anywhere in the string. Requires v1.0.14+.
alnum	Same as alpha except that digit characters are also allowed.
space	True if var is empty or contains only whitespace, which consists of the following characters: space (%A_Space%), tab (%A_Tab% or `t), linefeed (`n), return (`r), vertical tab (`v), and formfeed (`f).
time	True if var contains a valid date/time stamp, which can be all or just the leading part of the YYYYMMDDHH24MISS format. For example, a 4-digit string such as 2004 is considered valid. Use StringLen to determine whether additional time components are present. Years less than 1601 are not considered valid because the operating system generally does not support them. The maximum year considered valid is 9999. The word DATE may be used as a substitute for the word TIME, with the same result.

Related

%A_YYYY%, SetFormat, FileGetTime, IfEqual, If var in/contains MatchList, If var between, StringLen, IfInString, StringUpper, EnvAdd, Blocks, Else

Example

```
if var is float
    MsgBox, %var% is a floating point number.
else if var is integer
    MsgBox, %var% is an integer.
if var is time
    MsgBox, %var% is also a valid date/time.
```

Random

Generates a pseudo-random number.

Random, OutputVar [, Min, Max]

Parameters

OutputVar	The name of the variable in which to store the result. The format of stored floating point numbers is determined by SetFormat .
Min	The smallest number that can be generated (can be negative). If omitted, the smallest number will be 0. The lowest allowed value is -2147483648 for integers, but floating point numbers have no restrictions.
Max	The largest number that can be generated. If omitted, the largest number will be 2147483647 (which is also the largest allowed integer value -- but floating point numbers have no restrictions).

Remarks

This command creates a pseudo-randomly generated number, which is a number that simulates a true random number but is really a number based on a complicated formula to make determination/guessing of the next number extremely difficult. If either *Min* or *Max* contains a decimal point, the end result will be a floating point number in the format set by [SetFormat](#). Otherwise, the result will be an integer.

Related

[SetFormat](#)

Example

```
Random, rand, 1, 10  
Random, rand, 0.0, 1.0
```

Comments based on the original source

This function uses the Mersenne Twister random number generator, MT19937, written by Takuji Nishimura and Makoto Matsumoto, Shawn Cokus, Matthe Bellew and Isaku Wada.

The Mersenne Twister is an algorithm for generating random numbers. It was designed with consideration of the flaws in various other generators. The period, $2^{19937}-1$, and the order of equidistribution, 623 dimensions, are far greater. The generator is also fast, it avoids multiplication and division, and it benefits from caches and pipelines. For more information see the inventors' web page at <http://www.math.keio.ac.jp/~matumoto/emt.html>

Copyright (C) 1997 – 2002, Makoto Matsumoto and Takuji Nishimura, All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Do NOT use for CRYPTOGRAPHY without securely hashing several returned values together, otherwise the generator state can be learned after reading 624 consecutive values.

#SetFormat

Sets the format of integers and floating point numbers generated by math operations.

SetFormat, NumberType, Format

Parameters

NumberType	Must be either INTEGER or FLOAT.
	For NumberType INTEGER, this parameter must be H or HEX for hexadecimal, or D for decimal. Hexadecimal numbers all start with the prefix 0x (e.g. 0xFF). For NumberType FLOAT, this parameter is TotalWidth.DecimalPlaces (e.g. 0.6).
Format	TotalWidth is typically 0 to indicate that number should not have any blank or zero padding. If a higher value is used, numbers will be padded with spaces or zeroes (see below) to make them that wide. DecimalPlaces is the number of decimal places to display (rounding will occur). If blank or zero, neither a decimal portion nor a decimal point will be displayed, that is, the number will be stored as an integer rather than a floating point number. Padding: If TotalWidth is high enough to cause padding, spaces will be added on the left side, i.e. numbers will be right-justified. To use left-justification instead, precede TotalWidth with a minus sign. To pad with zeroes instead of spaces, precede TotalWidth with a zero.

Remarks

If this command is not used in a script, integers default to decimal format and floating point numbers default to TotalWidth.DecimalPlaces = 0.6. Every newly launched hotkey or timed subroutine starts off fresh with the default setting for this command. That default may be changed by using this command in the auto execute section (top part) of the script.

You can determine whether a variable contains a numeric value by using "if var is number/integer/float"

A variable containing a number can be forced into a new format (as set by this command) by adding zero to it, e.g. `Var += 0` or `Var += 0.0`

The built-in variable `A_FormatInteger` contains the current integer format (H or D). `A_FormatFloat` contains the current floating point format.

-

Floating Point Format

Since AutoHotkey stores all variables as strings, the stored precision of floating point numbers is determined by `DecimalPlaces`. In other words, once a variable is rounded off, the extra precision is lost and cannot be reclaimed without redoing the calculation using a higher value of `DecimalPlaces`.

Any leading or trailing spaces (i.e. padding) in a floating point value will be lost if that value is explicitly assigned to another variable. Use `AutoTrim` to prevent this.

For the current float format to take effect in a math operation such as addition, at least one of the inputs must contain a decimal point.

A variable containing an integer can be "converted" into a floating point number by adding `0.0` to it, e.g. `Var += 0.0`. However, if the current float format specifies no `DecimalPlaces`, use this instead:

```
if Var is integer  
    Var = %Var%.0
```

-

Hexadecimal Format

Hexadecimal numbers all start with the prefix `0x` (e.g. `0xA9`). They can be used anywhere a numerical value is expected. For example, "Sleep, `0xFF`" is equivalent to "Sleep, 255" regardless of the current integer format set by `SetFormat`.

To convert an integer from decimal to hexadecimal, use this example (the converse can be used to convert in the opposite direction):

```
SetFormat, integer, hex  
VariableContainingAnInteger += 0  
SetFormat, integer, d
```

-

Related

[EnvAdd](#), [EnvSub](#), [EnvMult](#), [EnvDiv](#), [AutoTrim](#), [if var is type](#)

-

Example

```
Var = 11.333333  
SetFormat, float, 0.2  
Var += ; Sets Var to be 12.33  
SetFormat, float, 06.0  
Var += 0 ; Sets Var to be 000012  
SetFormat, integer, hex  
Var += 0 ; Sets Var to be 0xe
```

~~format, integer, d~~

~~inform~~ [v1.0.11+]

...miscellaneous math functions and tasks such as ASCII/Unicode conversion and bitwise operations.

~~form, OutputVar, Cmd, Value1 [, Value2]~~

meters

<code>utVar</code>	The name of the variable in which to store the result of <code>Cmd</code> . <code>SetFormat</code> determines whether integers are stored as hexadecimal or decimal.
<code>, Value1/2</code>	See list below:

Value1, Value2

~~cmd. Value1 and Value2 parameters are dependent upon each other and their usage is described below.~~

`[Clipboard::String] [v1.0.24+]`: Retrieves or stores Unicode text on the clipboard. There are two modes of operation as illustrated in the following examples.

`GetClipboardText()` - Retrieves stored Unicode text on the clipboard. The form `OutputString` retrieves the clipboard's Unicode text as a UTF-8 string.

`String clipboardText = Clipboard.GetText(TextDataFormat.Unicode);` Retrieves the clipboard's Unicode text as a `String`.

In the second example above, a literal LITE-8 string may be optionally used in place of %M-LITE-String#%.

In the example above, a word can be being may be optionally used in place of the `be` being in the botkey such as the following to determine the UTE 8 string that corresponds to a given Unicode string:

hotkey such as the following:

Copy some Unicode text onto the clipboard, then return to this window and press OK to continue.

~~6x Copy some Unicode text on form. ClipLTE Unicode~~

Transform, Clipboard, Unicode

The clipboard now contains the following line that you can paste into your script. When executed, this line will cause the original Unicode string you copied to be placed onto the clipboard: `n`n%Clipboard

~~Windows 95 requires the Microsoft Layer for Unicode to support this command. Windows 98/Me/NT4 or later have built in support. Also, the "Send {ASC nnnnn}" command is an alternative way to produce Unicode characters.~~

Asc, String: Retrieves the ASCII code (a number between 0 and 255) for the first character in *String*. If *String* is blank, *OutputVar* will also be made blank. Example: Transform, OutputVar, Asc, %VarContainingString%

Chr, Value1: Retrieves the single character corresponding to the ASCII code indicated by *Value1*. If *Value1* is not between 0 and 255 inclusive, *OutputVar* will be made blank to indicate the problem. Example: Transform, OutputVar, Chr, 130

Deref, String [v1.0.17+]: Expands variable references and escape sequences contained inside other variables. Any badly formatted variable references will be omitted from the expanded result. The same is true if *OutputVar* is expanded into itself; in other words, any references to *OutputVar* inside *String*'s variables will be omitted from the expansion (note however that *String* itself can be %OutputVar%). In the following example, if var1 contains the string "test" and var2 contains the literal string "%var1%", *OutputVar* will be set to the string "test". Transform, OutputVar, deref, %var2%

HTML, String [v1.0.15+]: Converts *String* into its HTML equivalent by translating characters whose ASCII values are above 127 to their HTML names (e.g. £ becomes £). In addition, the four characters "&<>" are translated to ",<,>. Finally, each linefeed (`n) is translated to
n (i.e.
 followed by a linefeed).

Mod, Dividend, Divisor: Retrieves the remainder of *Dividend* divided by *Divisor*. If *Divisor* is zero, *OutputVar* will be made blank. *Dividend* and *Divisor* can both contain a decimal point. If negative, *Divisor* will be treated as positive for the calculation. In the following example, the result is 2: Transform, OutputVar, mod, 5, 3

Pow, Value1, N: Retrieves *Value1* raised to the *N*th power. Both *Value1* and *N* may contain a decimal point. If *N* is negative, *OutputVar* will be formatted as a floating point number even if *Value1* and *N* are both integers. If *Value1* is negative, *OutputVar* will be made blank, i.e. negatives are not supported.

Exp, N: Retrieves e (which is approximately 2.71828182845905) raised to the *N*th power. *N* may be negative and/or contain a decimal point.

Sqrt, Value1: Retrieves the square root of *Value1*. If *Value1* is negative, *OutputVar* will be made blank.

Log, Value1: Retrieves the logarithm (base 10) of *Value1*. If *Value1* is negative, *OutputVar* will be made blank.

Ln, Value1: Retrieves the natural logarithm (base e) of *Value1*. If *Value1* is negative, *OutputVar* will be made blank.

Round, Value1 [, N]: If *N* is omitted, *OutputVar* will be set to *Value1* rounded to the nearest integer. If *N* is positive number, *Value1* will be rounded to *N* decimal places. If *N* is negative, *Value1* will be rounded by *N* digits to the left of the decimal point. For example, -1 rounds to the ones place, -2 rounds to the tens place, -3 rounds to the hundreds place, etc.

Ceil, Value1: Retrieves *Value1* rounded up to the nearest integer.

Floor, Value1: Retrieves *Value1* rounded down to the nearest integer.

Abs, Value1: Retrieves the absolute value of *Value1*, which is computed by removing the leading minus sign (dash) from *Value1* if it has one.

Sin, Value1: Retrieves the trigonometric sine of *Value1*. *Value1* must be expressed in radians.

Cos, Value1: Retrieves the trigonometric cosine of *Value1*. *Value1* must be expressed in radians.

Tan, Value1: Retrieves the trigonometric tangent of *Value1*. *Value1* must be expressed in radians.

ASin, Value1: Retrieves the arcsine (the number whose sine is *Value1*) in radians. If *Value1* is less than -1 or greater than 1, *OutputVar* will be made blank.

ACos, Value1: Retrieves the arccosine (the number whose cosine is *Value1*) in radians. If *Value1* is less than -1 or greater than 1, *OutputVar* will be made blank.

ATan, Value1: Retrieves the arctangent (the number whose tangent is *Value1*) in radians.

BitNot, Value1: Stores the bit inverted version of *Value1* in *OutputVar*. If *Value1* is between 0 and 4294967295 (0xffffffff), it will be treated as an unsigned 32 bit value. Otherwise, it is treated as a signed 64 bit value. In the following example, the result is 0xffff0f0 (4294963440): Transform, OutputVar, BitNot, 0xf0f

BitAnd, Value1, Value2: Retrieves the result of the bitwise AND of *Value1* and *Value2*. In the following example, the result is 0xff00 (65280): Transform, OutputVar, BitAnd, 0xff0f, 0xffff0

BitOr, Value1, Value2: Retrieves the result of the bitwise OR of *Value1* and *Value2*. In the following example, the result is 0xf0f0 (61680). Transform, OutputVar, BitOr, 0xf000, 0x00f0

BitXOr, Value1, Value2: Retrieves the result of the bitwise EXCLUSIVE OR of *Value1* and *Value2*. In the following example, the result is 0xff00 (65280). Transform, OutputVar, BitXOr, 0xf00f, 0x0f0f

BitShiftLeft, Value1, Value2: Retrieves the result of shifting *Value1* to the left by *Value2* bit positions, which is equivalent to multiplying *Value1* by "2 to the *Value2*th power". In the following example, the result is 8. Transform, OutputVar, BitShiftLeft, 1, 3

BitShiftRight, Value1, Value2: Retrieves the result of shifting *Value1* to the right by *Value2* bit positions, which is equivalent to dividing *Value1* by "2 to the *Value2*th power" (truncating the remainder). In the following example, the result is 2. Transform, OutputVar, BitShiftRight, 17, 3

-

Remarks

If either *Value1* or *Value2* is a floating point number, the following *Cmds* will retrieve a floating point number rather than an integer: Mod, Pow, Round, and Abs. The number of decimal places retrieved is determined by SetFormat.

To convert a radians value to degrees, multiply it by 180/pi (approximately 57.29578). To convert a degrees value to radians, multiply it by pi/180 (approximately 0.01745329252).

The value of pi (approximately 3.141592653589793) is 4 times the arctangent of 1.

-

Related

SetFormat, EnvMult, EnvDiv, StringLower, if var is type

-

Example

Transform, OutputVar, Asc, A , Get the ASCII code of the letter A.

#NoTrayIcon

Disables the showing of a tray icon.

#NoTrayIcon

Parameters

None

Remarks

Specifying this anywhere in a script will prevent the showing of a tray icon for that script when it is launched (even if the script is compiled into an EXE).

If you use this for a script that has hotkeys, you might want to bind a hotkey to the [ExitApp](#) command. Otherwise, there will be no easy way to exit the program (without restarting the computer or killing the process). Example: #x::ExitApp

The tray icon can be made to disappear or reappear at any time during the execution of the script by using the command [menu, tray, Icon](#) or [menu, tray, NoIcon](#). The only drawback of using the [menu, tray, NoIcon](#) at the very top of the script is that the tray icon might be briefly visible when the script is first launched. To avoid that, use [#NoTrayIcon](#) instead.

The built-in variable **A_IconHidden** contains 1 if the tray icon is currently hidden or 0 otherwise.

Related

[Menu](#), [ExitApp](#)

Example

#NoTrayIcon

#SingleInstance

Prevents more than one instance of a script from existing simultaneously.

```
#SingleInstance [force|ignore|off|prompt]
```

Parameters

force|ignore|off|prompt

The word FORCE causes the program to skip the dialog box that is normally shown whenever a running script is launched a second time. Instead, the old instance will be automatically replaced, which is similar in effect to the [Reload](#) command.

The word IGNORE skips the dialog box as above, but leaves the old instance running. In other words, attempts to launch an already-running script are ignored.

The word OFF allows the running of multiple instances of a script. This is the default for scripts that are neither [persistent](#) nor contain [hotkeys](#).

Remarks

Related

[Reload](#)

Example

```
#SingleInstance force  
#SingleInstance ignore  
#SingleInstance off
```

#AutoTrim

~~Determines whether SetEnv and "var = value" statements remove spaces and tabs.~~

~~Currently always OFF.~~

~~AutoTrim, On|Off~~

Parameters

~~On|Off~~

~~On: tabs and spaces at the beginning and end of a string are removed from a variable whenever it is assigned a new value. This is the default.~~

~~Off: tabs and spaces are not removed.~~

Remarks

If this command is not used by a script, the setting defaults to ON.

The built-in variable **A_AutoTrim** contains the current setting (On or Off).

Every newly launched hotkey, custom menu item, or timed subroutine starts off fresh with the default setting for this command. That default may be changed by using this command in the auto-execute section (top part) of the script.

Related

The **%A_Space%** and **%A_Tab%** built-in variables contain a single space and single tab character, respectively.

Example

AutoTrim, off

NewVar1 = %OldVar%; If OldVar contains leading and trailing spaces, NewVar will have them too.

NewVar2 = %A_Space%; With AutoTrim off, a single space can be assigned this way.

CoordMode

Sets coordinate mode for various commands to be either relative or screen.

CoordMode, ToolTip|Pixel|Mouse [, Screen|Relative]

Parameters

Param1	ToolTip: Affects ToolTip . Pixel: Affects PixelGetColor and PixelSearch . Mouse: Affects MouseMove , MouseClick , MouseClickDrag , and MouseGetPos . Caret: Affects the built-in variables A_CaretX and A_CaretY . Menu: Affects the "Menu Show" command when coordinates are specified for it.
Param2	Screen (this is the default when omitted): Coordinates are relative to the desktop (entire screen).

Remarks

If this command isn't used, all commands except those documented otherwise (e.g. [WinMove](#) and [InputBox](#)) use coordinates that are relative to the active window.

Every newly launched [hotkey](#), [custom menu item](#), or [timed](#) subroutine starts off fresh with the default setting for this command. That default may be changed by using this command in the auto-execute section (top part) of the script.

Related

[MouseMove](#), [MouseClick](#), [MouseClickDrag](#), [MouseGetPos](#), [PixelGetColor](#), [PixelSearch](#), [ToolTip](#), [Menu](#)

Example

```
; Place ToolTips at absolute screen coordinates:  
CoordMode, ToolTip, Screen
```

#Echo

Prints to the console (standard out).

Echo, Text

Parameters

<i>Text</i>	<i>Whatever you want to write.</i>
-------------	------------------------------------

Remarks

You might only need this if you intend to write an application that is meant to be used from the command line. There isn't really any benefit to using Echo over RunWait, echo apart from small performance benefit and shorter instruction length.

Example

Echo, Hello World

Edit

Opens the current script for editing in the associated editor.

Parameters

None

Remarks

This command opens the current script for editing (using the associated "edit" verb, or Notepad if no verb), or activates an existing Editor window if it finds one it thinks may have the file open based on its window title. It is equivalent to selecting "Edit This Script" from the tray menu.

This command has no effect when operating from within a compiled script.

On a related note, the Extras folder included with AutoHotkey has some add-ons for editors such as TextPad. In addition, context sensitive help for AutoHotkey commands can be enabled in any editor via [this example](#).

Related

[Reload](#)

Example

Edit ; opens the script for editing.

#Eval

Takes a string argument and runs it as commands.

Eval, Commands

Parameters

Commands

The commands to run. May include line breaks to run multiple. E.g. `Eval, MsgBox 1`nMsgBox 2`

Remarks

Thread variables such as `ErrorLevel` cannot be accessed as the code is executed in its own thread, run immediately.

Example

```
; Run a dynamic command:  
cmd = MsgBox  
Eval, %cmd% Hello
```

##DefineCommand

Registers a new command. Comparable but inferior to "function"s in modern Windows AHK.

#DefineCommand, Name, Label

Parameters

Name	The name of the new command which you can then invoke in subsequent lines.
Label	The name of the label or hotkey label to which to jump when the newly registered command is invoked. Internally this behaves like GoSub , check there for details.

Remarks

You should use this directive if you need to encapsulate logic that you use multiple times in your script, to prevent repeating yourself. Only do this if you can't find another built-in command that already does what you need.

This directive can be used multiple times to register multiple commands. It will only affect code in subsequent lines. This means that you will likely want to put it at the top of your script. The labels, however, can be placed anywhere in your script.

Inside the respective label, you can access the passed parameters using the pseudo-built-in vars A_Param1, A_Param2 and so on. Check the example below.

This feature hasn't been optimized a lot. It may not be super fast and it's possible to shadow variable names with a custom command (not built-in commands though). Generally however, it should work great for most use cases.

Example

```
; Registering a new custom command: GetMaximumValue, OutputVar [, Value1, Value2, ...]
#DefineCommand GetMaximumValue, LblGetMaximumValue

GetMaximumValue, var1, 7, 8, 26, 12, 15
GetMaximumValue, var2, 4, %var1%

; Prints: 26
MsgBox, Maximum: %var2%
Return

LblGetMaximumValue:
max = 0
; Iterating A_Param1, A_Param2 and so on until no more params are found:
Loop
{
    param_index = %A_Index%
    ; We skip the first param as it's the OutputVar
    param_index += 1
    StringTrimLeft, value, A_Param%param_index%, 0
    If value =
        Break
    If value > %max%
        max = %value%
}
```

```
; We have defined OutputVar as our first parameter. This will set the out value:  
%A_Param1% = %max%  
max =  
Return
```

#ListLines

~~Displays the script lines most recently executed.~~

ListLines

- Parameters

~~None~~

- Remarks

~~This command is equivalent to selecting the "View->Lines most recently executed" menu item in the main window.~~

- Related

~~KeyHistory, ListHotkeys, ListVars~~

- Example

~~ListLines~~

#ListVars

Displays the script's variables: their names and current contents.

ListVars

-

Parameters

None

-

Remarks

This command is equivalent to selecting the "View->Variables" menu item in the main window.

-

Related

[KeyHistory](#), [ListHotkeys](#), [ListLines](#)

-

Example

ListVars

-

#EnvGet

Retrieves an environment variable.

EnvGet, *OutputVar*, *EnvVarName*

Menu [v1.0.07+]

Creates, ~~deletes, modifies~~ and displays menus and menu items. Changes the tray icon and its tooltip. ~~Controls whether the main window of a compiled script can be opened.~~

Menu, MenuName, Cmd [, P3, P4, P5, FutureUse]

Parameters

MenuName	This parameter must be the word TRAY since that is the only supported menu. In v1.0.13+, it can be TRAY or the name of any custom menu. A custom menu is automatically created the first time its name is used with the Add command. Example: Menu, MyMenu, Add, Item1 Once created, a custom menu can be displayed with the Show command. It can also be attached as a submenu to one or more other menus via the Add command.
Cmd, P3, P4, P5	These 4 parameters are dependent on each other. See list below for the allowed combinations.
FutureUse	This parameter should always be omitted.

Change the Items in a Menu

Add [, MenuItemName, Label-or-Submenu, Pn]: This is a multipurpose command that adds a menu item, ~~updates one with a new submenu or label, or converts one from a normal item into a submenu (or vice versa)~~. If *MenuItemName* doesn't yet exist, it will be added. ~~Otherwise, MenuItemName is updated with the newly specified Label-or-Submenu.~~

To add a menu separator line, omit both parameters. If only *Label-or-Submenu* is omitted, *MenuItemName* will be used as both the label and the menu item's name. The label subroutine is run as a new **thread** when the user selects the menu item (similar to **Gosub** and **hotkey subroutines**).

~~To have MenuItemName become a submenu — which is a menu item that opens a new menu when selected — specify for Label-or-Submenu a colon followed by the MenuName of an existing custom menu. Example:~~
~~Menu, MySubmenu, add, Item1~~

~~Menu, tray, add, This Menu Item Is A Submenu, :MySubmenu~~

The *Pn* parameter specifies for *n* the menu's **thread priority**, e.g. p1. If omitted when adding a menu item, the priority will be 0. If omitted when ~~updating a menu item, the item's priority will not be changed~~. To change an existing item's priority only, omit the *Label-or-Submenu* parameter. Use a decimal (not hexadecimal) number for *n*.

Delete [, MenuItemName]: Deletes *MenuItemName* from the menu. Standard menu items (see below) cannot be individually deleted. If the *default* menu item is deleted, the effect will be similar to having used the *NoDefault* option. If *MenuItemName* is omitted, the entire *MenuName* menu will be deleted as will any menu items in other menus that use *MenuName* as a submenu.

DeleteAll: Deletes all custom menu items from the menu, leaving the menu empty unless it contains the standard items (see below). Unlike a menu entirely deleted by the *Delete* command (see above), an empty menu still exists and thus any other menus that use it as a submenu will retain those submenus.

Rename, MenuItemName [, NewName] [v1.0.08+]: Changes *MenuItemName* to *NewName* (if *NewName* is blank, *MenuItemName* will be converted into a separator line). *NewName* must not be the same as any existing custom menu item. The menu item's current target label or submenu is unchanged.

Check, MenuItemName: Adds a visible checkmark in the menu next to *MenuItemName* (if there isn't one already).

Uncheck, MenuItemName: Removes the checkmark from *MenuItemName* if there is one.

ToggleCheck, MenuItemName: Adds a checkmark if there wasn't one; otherwise, removes it.

Enable, MenuItemName: Allows the user to once again select *MenuItemName* if was previously disabled (grayed).

Disable, MenuItemName: Changes *MenuItemName* to a gray color to indicate that the user cannot select it.

ToggleEnable, MenuItemName: Disables *MenuItemName* if it was previously enabled; otherwise, enables it.

Default [, MenuItemName]: Changes the menu's default item to be *MenuItemName* and makes that item's font bold (setting a default item in menus other than the tray currently has no other function). When the user double-clicks the tray icon, its default menu item is launched. If there is no default, double-clicking has no effect. If *MenuItemName* is omitted, the effect is the same as having used *NoDefault* below.

NoDefault: For the tray menu: Changes the menu back to having its standard default menu item, which is OPEN for non-compiled scripts and none for compiled scripts (except when the *MainWindow* option is in effect). If the OPEN menu item does not exist due to a previous use of the *NoStandard* command below, there will be no default and thus double-clicking the tray icon will have no effect. For menus other than the tray: Any existing default item is returned to a non-bold font.

Standard: Inserts the standard menu items at the bottom of the menu (if they are not already present). This command can be used with the tray menu or any other menu.

NoStandard: Removes all standard (non-custom) menu items from the tray menu (if they are present).

Change the Icon or ToolTip (*MenuName* must be TRAY)

Icon, FileName [, IconNumber] [v1.0.13+]: Changes the script's icon to one of the ones from *FileName* (a .ico, .dll, or .exe file or an image file such as .png or .gif. .jpg does not seem to work.). If *IconNumber* is omitted, it defaults to 1 (the first icon in the file). Specify an asterisk (*) for *FileName* to restore the script to its default icon. This setting affects the tray icon and the one displayed by *InputBox*, *Progress*, and *GUI* windows. Compiled scripts are also affected even if a custom icon was specified at the time of compiling. Note: Changing the icon will not unhide the tray icon if it was previously hidden by means such as *#NoTrayIcon*; to do that, use "Menu, Tray, Icon" as described below. Also, the built-in variables **A_IconNumber** and **A_IconFile** contain the number and name (with full path) of the current icon (both are blank if the icon is the default).

Icon (with no parameters): Creates the tray icon if it isn't already present. This will override *#NoTrayIcon* if that directive is also present in the script.

NoIcon: Removes the tray icon if it exists. If this command is used at the very top of the script, the tray icon might be briefly visible when the script is launched. To prevent that, use *#NoTrayIcon* instead. The built-in variable **A_IconHidden** contains 1 if the tray icon is currently hidden or 0 otherwise.

Tip [, Text] [v1.0.13+]: Changes the tray icon's tooltip -- which is displayed when the mouse hovers over it -- to be *Text*. To create a multi-line tooltip, use the linefeed character (`n) in between each line, e.g. Line1`nLine2. If *Text* is omitted, the tooltip is restored to its default text. The built-in variable **A_IconTip** contains the current text of the tooltip (blank if the text is at its default).

Misc.

Show [, X, Y] [v1.0.13+]: Displays *MenuName*, allowing the user to select an item with arrow keys, menu shortcuts (underlined letters), or the mouse. Any menu can be shown, including the tray menu but with the exception of GUI menu bars. If both X and Y are omitted, the menu is displayed at the current position of the mouse cursor. If only one of them is omitted, the mouse cursor's position will be used for it. X and Y are relative to the active window. Specify "CoordMode, Menu" beforehand to make them relative to the entire screen. [X and Y require v1.0.23+].

Color, ColorValue [, Single] [v1.0.19+]: Changes the background color of the menu to *ColorValue*, which is one of the 16 primary HTML color names or a 6-digit RGB color value (see color chart). Leave *ColorValue* blank (or specify the word Default) to restore the menu to its default color. If the word Single is not present as the next parameter, any submenus attached to this menu will also be changed in color. This command has no effect on Windows 95/NT.

Click, ClickCount [v1.0.24+]: Specify 1 for *ClickCount* to allow a single click to activate the tray menu's default menu item. Specify 2 for *ClickCount* to return to the default behavior (double-click). Example: Menu, Tray, Click, +

MainWindow [v1.0.09+]: This command affects compiled scripts only. It allows the script's main window to be opened via the tray icon, which is otherwise impossible. It also enables the items in the main window's View menu -- such as "Lines most recently executed" -- allowing users to view the script's source code and other info. *MenuName* must be TRAY.

NoMainWindow [v1.0.09+] (default): This command affects compiled scripts only. It restores the script to its default behavior, that is, it prevents the main window from being opened. Even while this option is in effect, the following commands are still able to show the main window when they are encountered in the script at runtime: ListLines, ListVars, ListHotkeys, and KeyHistory. *MenuName* must be TRAY.

UseErrorLevel [, off] [v1.0.13+]: If this option is never used in the script, it defaults to OFF. The OFF setting displays a dialog and terminates the current thread whenever the Menu command generates an error. Specify *Menu, AnyMenuName, UseErrorLevel* to prevent the dialog and thread termination; instead, ErrorLevel will be set to 1 if there was a problem or 0 otherwise. To turn this option back off, specify OFF for the next parameter. This setting is global, meaning it affects all menus, not just *MenuName*.

Remarks

To underline one of the letters in a menu item's name, precede that letter with an ampersand (&). When the menu is displayed, such menu items can be selected by pressing the corresponding key on the keyboard. To display a literal ampersand, specify two consecutive ampersands as in this example: Save && Exit

The names of menus and menu items can be up to 260 characters long.

When referring to an existing menu or menu item, the name is not case sensitive but you must include any ampersands it contains. Example: &Notepad

Separator lines can be added to the menu by using *Menu, tray, add* (i.e. omit all other parameters). However, separator lines currently cannot be individually deleted. To work around this, use *Menu, tray, DeleteAll* and then re-add your custom menu items.

Custom menu items are always added at the bottom of the menu. For the tray menu: To put your menu items on top of the standard menu items (after adding your own menu items) run *Menu, tray, NoStandard* followed by *Menu, tray, Standard*.

If a menu ever becomes completely empty -- such as by using *Menu, MyMenu, DeleteAll* -- it cannot be shown. If the tray menu becomes empty, right-clicking and double-clicking the tray icon will have no effect.

If a menu item's subroutine is already running and the user selects the same menu item again, a new **thread** will be created to run that same subroutine, interrupting the previous thread.

Whenever a subroutine is launched via a menu item, it starts off fresh with the default values for settings such as *SetKeyDelay*. These defaults can be changed in the **auto-execute section**.

In v1.0.13+, the built-in variable **A_ThisMenuItem** and **A_ThisMenuItemPos** contain the name and position of the custom menu item most recently selected by the user (blank if none). Similarly, **A_ThisMenu** is the name of the menu from which **A_ThisMenuItem** was selected. These variables are useful when building a menu whose contents are not always the same. In such a case, it is usually best to point all such menu items to the same label and have it refer to the above variables to determine what action to take.

To keep a non-hotkey script running -- such as one that contains only custom menus or menu items -- use **#Persistent**.

Related

[GUI](#), [Threads](#), [Thread](#), [Gosub](#), [Return](#), [SetTimer](#), [#Persistent](#)

Examples

```
Menu, tray, add, Item1, MenuHandler
return
MenuHandler:
MsgBox You selected %A_ThisMenuItem% from menu %A_ThisMenu%
return

; This next example is a working script to demonstrate some of the various menu commands.
#Persistent
#SingleInstance
menu, tray, add ; separator
menu, tray, add, TestToggle&Check
menu, tray, add, TestToggleEnable
menu, tray, add, TestDefault
menu, tray, add, TestStandard
menu, tray, add, TestDelete
menu, tray, add, TestDeleteAll
menu, tray, add, TestRename
menu, tray, add, Test
return

;;;;;;;;;;;;;;;

TestToggle&Check:
menu, tray, ToggleCheck, TestToggle&Check
menu, tray, Enable, TestToggleEnable ; Also enables the next test since it can't undo the disabling of itself.
menu, tray, add, TestDelete ; Similar to above.
return

TestToggleEnable:
menu, tray, ToggleEnable, TestToggleEnable
return

TestDefault:
if default = TestDefault
{
    menu, tray, NoDefault
    default =
}
else
{
    menu, tray, Default, TestDefault
```

```
        default = TestDefault
    }
    return

    TestStandard:
    if standard <> n
    {
        menu, tray, NoStandard
        standard = n
    }
    else
    {
        menu, tray, Standard
        standard = y
    }
    return

    TestDelete:
    menu, tray, delete, TestDelete
    return

    TestDeleteAll:
    menu, tray, DeleteAll
    return

    TestRename:
    if NewName <> renamed
    {
        OldName = TestRename
        NewName = renamed
    }
    else
    {
        OldName = renamed
        NewName = TestRename
    }
    menu, tray, rename, %OldName%, %NewName%
    return

    Test:
    MsgBox, You selected "%A_ThisMenuItem%" in menu "%A_ThisMenu%".
    return
```

#PixelGetColor

Retrieves the color of the pixel at the specified x,y screen coordinates.

```
PixelGetColor, OutputVar, X, Y [, RGB]
```

Parameters

OutputVar	The name of the variable in which to store the color ID in hexadecimal blue-green-red (BGR) format. For example, the color purple is defined 0x800080 because it has an 80 intensity for its blue and red components but a 00 intensity for its green component.
X,Y	The X and Y coordinates of the pixel. Coordinates are relative to the active window unless CoordMode was used to change that.
RGB	If this parameter is the word RGB, the color will be retrieved in RGB vs. BGR format. In other words, the red and the blue components are swapped. This is useful for retrieving colors compatible with WinSet , Gui , Progress , and SplashImage .

ErrorLevel

[ErrorLevel](#) is set to 1 if there was a problem or 0 otherwise.

Remarks

The pixel must be visible; in other words, it is not possible to retrieve the pixel color of a window hidden behind another window.

Use Window Spy (available in tray icon menu) or the example below to determine the colors currently on the screen.

Related

[PixelSearch](#), [CoordMode](#), [MouseGetPos](#)

Example

```
^!z:::  
MouseGetPos, MouseX, MouseY  
PixelGetColor, color, %MouseX%, %MouseY%
```

```
MsgBox, The color at the current cursor position is %color%.  
return
```

PixelSearch

Searches a region of the screen for a pixel of the specified color(s).

```
PixelSearch, OutputVarX, OutputVarY, X1, Y1, X2, Y2, ColorID [, Variation, RGB]
```

Parameters

OutputVarX/Y	The names of the variables in which to store the X and Y coordinates of the first pixel that matches <i>ColorID</i> . Either or both of these may be omitted (i.e. if you're just interested in the value of ErrorLevel set by the command).
X1,Y1	The X and Y coordinates of the upper left corner of the rectangle to search. Coordinates are relative to the active window unless CoordMode was used to change that.
X2,Y2	The X and Y coordinates of the lower right corner of the rectangle to search. Coordinates are relative to the active window unless CoordMode was used to change that.
ColorID	The decimal or hexadecimal color ID to search for, in Blue-Green-Red (BGR) format. Color IDs can be determined using Window Spy (accessible from the tray menu) or via PixelGetColor . Example: 0x9d6346
Variation	A number between 0 and 255 (inclusive) to indicate the allowed number of shades of variation in either direction for the intensity of the red, green, and blue components of the color. This can be helpful if the color you're looking for isn't always exactly the same shade. If you specify 255 shades of variation, all colors will match. The default is 0 shades.
RGB	If this parameter is the word RGB, <i>ColorID</i> is assumed to be in RGB vs. BGR format. In other words, the red and the blue components are swapped.

ErrorLevel

[ErrorLevel](#) is set to 0 if the color was found in the specified region, 1 if it was not found, or 2 if there was a problem that prevented the command from conducting the search.

Remarks

The region to be searched must be visible; in other words, it is not possible to search a region of a window hidden behind another window.

If the region to be searched is large and the search is repeated with high frequency, it may consume a lot of CPU time. To alleviate this, keep the size of the area to a minimum.

Related

[PixelGetColor](#), [CoordMode](#), [MouseGetPos](#)

Example

```
PixelSearch, Px, Py, 200, 200, 300, 300, 16777215, 3
if ErrorLevel = 0
    MsgBox, A color within 3 shades of variation was found at X%Px% Y%Py%.
else
    MsgBox, That color was not found in the specified region.
```

SetEnv

Assigns the specified value to a [variable](#). Synonymous with: `var = value`

```
SetEnv, Var, Value
Var = Value
```

Parameters

Var	The name of the variable in which to store <code>Value</code> .
Value	The string or number to store.

Remarks

The name "SetEnv" is misleading and is used only for backward compatibility with AutoIt2: Unlike AutoIt2, AutoHotkey does not store its variables in the environment. This is because performance would be worse and also because the OS limits such variables to 32K (AutoHotkey variables, including the clipboard, are essentially unlimited in size). Use `EnvSet` instead of `SetEnv` to write to an environment variable.

The memory occupied by a large variable can be freed by setting it equal to nothing, e.g. `var =`

It is possible to create an [array](#) with this command and any others that accept an *OutputVar*. This is done by making *OutputVar* contain a reference to another variable, e.g. `array%0i% = 123`. See [Arrays](#) for more info.

Related

[EnvSet](#), [EnvAdd](#), [EnvSub](#), [EnvMult](#), [EnvDiv](#), [If](#), [Arrays](#)

Example

```
Var1 = This is a string.  
Color2 = 450  
Color3 = %Var1%  
Array%0% = %A_TICKCOUNT%
```

#SysGet [v1.0.22+]

~~Retrieves screen resolution, multi-monitor info, dimensions of system objects, and other system properties.~~

~~SysGet, OutputVar, Sub-command [, Param3]~~

~~Parameters~~

OutputVar	The name of the variable in which to store the result.
Sub-command	See list below.
Param3	This parameter is omitted except where noted below.

~~Sub-commands~~

~~MonitorCount~~: Retrieves the total number of monitors. Unlike `SM_CMONITORS` mentioned in the table below, `MonitorCount` includes all monitors, even those not being used as part of the desktop. On Windows 95/NT the count is always 1.

~~MonitorPrimary~~: Retrieves the number of the primary monitor, which will be 1 in a single monitor system. On Windows 95/NT the primary monitor is always 1.

~~Monitor [, N]~~: Retrieves the bounding coordinates of monitor number `N` (if `N` is omitted, the primary monitor is used). The information is stored in four variables whose names all start with `OutputVar`. If `N` is too high or there is a problem retrieving the info, the variables are all made blank. Example:

```

SysGet, Mon2, Monitor, 2
MsgBox, Left: %Mon2Left% - Top: %Mon2Top% - Right: %Mon2Right% - Bottom %Mon2Bottom%.
MonitorWorkArea [, N]: Same as the above except the area is reduced to exclude the area occupied by the taskbar and other registered desktop toolbars.
MonitorName [, N]: The operating system's name for monitor number N (if N is omitted, the primary monitor is used).
(Numeric): Specify for Sub-command one of the numbers from the table below to retrieve the corresponding value. The following example would store the number of mouse buttons in a variable named "MouseButtonCount".
SysGet, MouseButtonCount, 43
-
```

Commonly Used

Number	Description
80	SM_CMONITORS : Number of display monitors on the desktop (not including "non display pseudo monitors"). Windows 95/NT : The retrieved value is always 0.
43	SM_CMOUSEBUTTONS : Number of buttons on mouse (0 if no mouse is installed).
16, 17	SM_CXFULLSCREEN , SM_CYFULLSCREEN : Width and height of the client area for a full-screen window on the primary display monitor, in pixels.
61, 62	SM_CXMAXIMIZED , SM_CYMAXIMIZED : Default dimensions, in pixels, of a maximized top-level window on the primary display monitor.
59, 60	SM_CXMAXTRACK , SM_CYMAXTRACK : Default maximum dimensions of a window that has a caption and sizing borders, in pixels. This metric refers to the entire desktop. The user cannot drag the window frame to a size larger than these dimensions.
28, 29	SM_CXMIN , SM_CYMIN : Minimum width and height of a window, in pixels.
57, 58	SM_CXMINIMIZED , SM_CYMINIMIZED : Dimensions of a minimized window, in pixels.
34, 35	SM_CXMINTRACK , SM_CYMINTRACK : Minimum tracking width and height of a window, in pixels. The user cannot drag the window frame to a size smaller than these dimensions. A window can override these values by processing the WM_GETMINMAXINFO message.
0, 1	SM_CXSCREEN , SM_CYSCREEN : Width and height of the screen of the primary display monitor, in pixels. These are the same as the built-in variables A_ScreenWidth and A_ScreenHeight .
78, 79	SM_CXVIRTUALSCREEN , SM_CYVIRTUALSCREEN : Width and height of the virtual screen, in pixels. The virtual screen is the bounding rectangle of all display monitors. The SM_XVIRTUALSCREEN , SM_YVIRTUALSCREEN metrics are the coordinates of the top-left corner of the virtual screen. Windows NT , Windows 95 : The retrieved value is always 0.
49	SM_MOUSEPRESENT : Nonzero if a mouse is installed; zero otherwise.
75	SM_MOUSEWHEELPRESENT : Nonzero if a mouse with a wheel is installed; zero otherwise. Windows 95 : The retrieved value is always 0.
63	SM_NETWORK : Least significant bit is set if a network is present; otherwise, it is cleared. The other bits are reserved for future use.

8193	SM_REMOTECONTROL: This system metric is used in a Terminal Services environment. Its value is nonzero if the current session is remotely controlled; zero otherwise. Windows 2000/NT, Windows Me/98/95: The retrieved value is always 0.
4096	SM_REMOTESESSION: This system metric is used in a Terminal Services environment. If the calling process is associated with a Terminal Services client session, the return value is nonzero. If the calling process is associated with the Terminal Server console session, the return value is zero. The console session is not necessarily the physical console. Windows NT 4.0 SP3 and earlier, Windows Me/98/95: The retrieved value is always 0.
70	SM_SHOWSOUNDS: Nonzero if the user requires an application to present information visually in situations where it would otherwise present the information only in audible form; zero otherwise.
8192	SM_SHUTTINGDOWN: Nonzero if the current session is shutting down; zero otherwise. Windows 2000/NT, Windows Me/98/95: The retrieved value is always 0.
23	SM_SWAPBUTTON: Nonzero if the meanings of the left and right mouse buttons are swapped; zero otherwise.
76, 77	SM_XVIRTUALSCREEN, SM_YVIRTUALSCREEN: Coordinates for the left side and the top of the virtual screen. The virtual screen is the bounding rectangle of all display monitors. By contrast, the SM_CXVIRTUALSCREEN, SM_CYVIRTUALSCREEN metrics (further above) are the width and height of the virtual screen. Windows NT, Windows 95: The retrieved value is always 0.

Not Commonly Used

Number	Description
56	SM_ARRANGE: Flags specifying how the system arranged minimized windows. See MSDN for more information.
67	SM_CLEANBOOT: Specifies how the system was started: 0 Normal boot 1 Fail-safe boot 2 Fail-safe with network boot
5, 6	SM_CXBORDER, SM_CYBORDER: Width and height of a window border, in pixels. This is equivalent to the SM_CXEDGE value for windows with the 3-D look.
13, 14	SM_CXCURSOR, SM_CYCURSOR: Width and height of a cursor, in pixels. The system cannot create cursors of other sizes.
36, 37	SM_CXDDOUBLECLK, SM_CYDDOUBLECLK: Width and height of the rectangle around the location of a first click in a double-click sequence, in pixels. The second click must occur within this rectangle for the system to consider the two clicks a double-click. (The two clicks must also occur within a specified time.)
68, 69	SM_CXDRAG, SM_CYDRAG: Width and height of a rectangle centered on a drag point to allow for limited movement of the mouse pointer before a drag operation begins. These values are in pixels. It allows the user to click and release the mouse button easily without unintentionally starting a drag operation.
45, 46	SM_CXEDGE, SM_CYEDGE: Dimensions of a 3-D border, in pixels. These are the 3-D counterparts of SM_CXBORDER and SM_CYBORDER.
7, 8	SM_CXFIXEDFRAME, SM_CYFIXEDFRAME (synonymous with SM_CXDLGFRAME, SM_CYDLGFRAME): Thickness of the frame around the perimeter of a window that has a caption but is not sizable, in pixels. SM_CXFIXEDFRAME is the height of the horizontal border and SM_CYFIXEDFRAME is the width of the vertical border.
83, 84	SM_CXFOCUSBORDER, SM_CYFOCUSBORDER: Width (in pixels) of the left and right edges and the height of the top and bottom edges of a control's focus rectangle. Windows 2000/NT, Windows Me/98/95: The retrieved value is always 0.
21, 22	SM_CXHSCROLL, SM_CYHSCROLL: Width of the arrow bitmap on a horizontal scroll bar, in pixels; and height of a horizontal scroll bar, in pixels.

10	SM_CXHTHUMB: Width of the thumb box in a horizontal scroll bar, in pixels.
11, 12	SM_CXICON, SM_CYICON: Default width and height of an icon, in pixels.
38, 39	SM_CXICONSPACING, SM_CYICONSPACING: Dimensions of a grid cell for items in large icon view, in pixels. Each item fits into a rectangle of this size when arranged. These values are always greater than or equal to SM_CXICON and SM_CYICON.
71, 72	SM_CXMENUCHECK, SM_CYMENUCHECK: Dimensions of the default menu check mark bitmap, in pixels.
54, 55	SM_CXMENUSIZE, SM_CYMENUSIZE: Dimensions of menu bar buttons, such as the child window close button used in the multiple document interface, in pixels.
47, 48	SM_CXMINSPACING SM_CYMINSPACING: Dimensions of a grid cell for a minimized window, in pixels. Each minimized window fits into a rectangle this size when arranged. These values are always greater than or equal to SM_CXMINIMIZED and SM_CYMINIMIZED.
29, 30	SM_CXSIZE, SM_CYSIZE: Width and height of a button in a window's caption or title bar, in pixels.
32, 33	SM_CXSIZEFRAME, SM_CYSIZEFRAME: Thickness of the sizing border around the perimeter of a window that can be resized, in pixels. SM_CXSIZEFRAME is the width of the horizontal border, and SM_CYSIZEFRAME is the height of the vertical border. Synonymous with SM_CXFRAME and SM_CYFRAME.
49, 50	SM_CXSMICON, SM_CYSMICON: Recommended dimensions of a small icon, in pixels. Small icons typically appear in window captions and in small icon view.
52, 53	SM_CXSMSIZE SM_CYSMSIZE: Dimensions of small caption buttons, in pixels.
2, 20	SM_CXVSCROLL, SM_CYVSCROLL: Width of a vertical scroll bar, in pixels; and height of the arrow bitmap on a vertical scroll bar, in pixels.
4	SM_CYCAPTION: Height of a caption area, in pixels.
18	SM_CYKANJIWINDOW: For double byte character set versions of the system, this is the height of the Kanji window at the bottom of the screen, in pixels.
15	SM_CYMENU: Height of a single-line menu bar, in pixels.
51	SM_CYSMCAPTION: Height of a small caption, in pixels.
9	SM_CYVTHUMB: Height of the thumb box in a vertical scroll bar, in pixels.
42	SM_DBCSENABLED: Nonzero if User32.dll supports DBCS; zero otherwise. Windows Me/98/95: Nonzero if the double byte character set (DBCS) version of User.exe is installed; zero otherwise.
22	SM_DEBUG: Nonzero if the debug version of User.exe is installed; zero otherwise.
82	SM_IMMENABLED: Nonzero if Input Method Manager/Input Method Editor features are enabled; zero otherwise. Windows NT, Windows Me/98/95: The retrieved value is always 0. SM_IMMENABLED indicates whether the system is ready to use a Unicode-based IME on a Unicode application. To ensure that a language-dependent IME works, check SM_DBCSENABLED and the system ANSI code page. Otherwise the ANSI to Unicode conversion may not be performed correctly, or some components like fonts or registry setting may not be present.

87	SM_MEDIACENTER : Nonzero if the current operating system is the Windows XP, Media Center Edition, zero if not.
40	SM_MENUDROPALIGNMENT : Nonzero if drop-down menus are right-aligned with the corresponding menu-bar item; zero if the menus are left-aligned.
74	SM_MIDEASTENABLED : Nonzero if the system is enabled for Hebrew and Arabic languages, zero if not.
41	SM_PENWINDOWS : Nonzero if the Microsoft Windows for Pen computing extensions are installed; zero otherwise.
44	SM_SECURE : Nonzero if security is present; zero otherwise.
81	SM_SAMEDISPLAYFORMAT : Nonzero if all the display monitors have the same color format, zero otherwise. Note that two displays can have the same bit depth, but different color formats. For example, the red, green, and blue pixels can be encoded with different numbers of bits, or those bits can be located in different places in a pixel's color value. Windows NT, Windows 95 : The retrieved value is always 0.
73	SM_SLOWMACHINE : Nonzero if the computer has a low-end (slow) processor; zero otherwise.
86	SM_TABLETPC : Nonzero if the current operating system is the Windows XP Tablet PC edition, zero if not.

-

Remarks

The built-in variables ~~A_ScreenWidth~~ and ~~A_ScreenHeight~~ contain the dimensions of the primary monitor, in pixels.

-

Related

[WinGet](#)

-

Examples

Example #1:

```
SysGet, MouseButtonCount, 43
SysGet, VirtualScreenWidth, 78
SysGet, VirtualScreenHeight, 79
```

Example #2: This is a working script that displays info about each monitor.

```
SysGet, MonitorCount, MonitorCount
SysGet, MonitorPrimary, MonitorPrimary
MsgBox, Monitor Count: `t%MonitorCount%`nPrimary Monitor: `t%MonitorPrimary%
Loop, %MonitorCount%
+
SysGet, MonitorName, MonitorName, %A_Index%
SysGet, Monitor, Monitor, %A_Index%
SysGet, MonitorWorkArea, MonitorWorkArea, %A_Index%
MsgBox, Monitor: `t%A_Index%`nName: `t%MonitorName%`nLeft: `t%MonitorLeft%`nTop: `t%MonitorTop%`nRight: `t%MonitorRight%`nBottom: `t%MonitorBottom%
```

(MonitorWorkAreaBottom: work)
+

#Thread [v1.0.16+]

Sets the priority or interruptibility of the threads.

Thread, Setting, P2 [, P3]

-

Parameters

See list below.

-

Setting, P2, P3

The *Setting*, *P2*, and *P3* parameters are dependent upon each other and their usage is described below:

Priority, n: Specify for *n* an integer between -2147483648 and 2147483647 to indicate the current thread's new priority. This has no effect on other threads. See Threads for details.

On a related note, the OS's priority level for the entire script can be changed as in this example: *Process, priority,, high*

Interrupt, Duration [, LineCount]: By default, every newly launched thread is uninterruptible for a *Duration* of 15 milliseconds or a *LineCount* of 1000 script lines, whichever comes first. This gives the thread a chance to finish rather than being immediately interrupted by another thread that is waiting to launch (such as a buffered hotkey or a series of timed subroutines that are all due to be run).

If either component is 0, each newly launched thread is immediately interruptible. If either component is -1, the thread cannot be interrupted as a result of that component.

The *Interrupt* setting is global, meaning that all subsequent threads will obey it, even if the setting is changed somewhere other than the auto-execute section. However, interrupted threads are unaffected because their period of uninterruptibility has already expired. Similarly, the current thread is unaffected except if it is uninterruptible at the time the *LineCount* component is changed, in which case the new *LineCount* will be in effect for it.

If a hotkey is pressed or a custom menu item is selected while the current thread is uninterruptible, that event will be buffered. In other words, it will launch when the current thread finishes or becomes interruptible, whichever comes first. The exception to this is when the current thread becomes interruptible before it finishes, and it is of higher priority than the buffered event; in this case the buffered event is unbuffered and discarded.

Regardless of this setting, a thread will become interruptible the moment it displays a *MsgBox*, *InputBox*, *FileSelectFile*, or *FileSelectFolder* dialog.

Either parameter can be left blank to avoid changing it.

-

Remarks

This command should be used sparingly because most scripts perform more consistently with settings close to the defaults.

-

Related

[Threads](#), [Hotkey](#), [Menu](#), [SetTimer](#)

-

Example

~~Thread, priority, 1 ; Make priority of current thread slightly above average.~~

~~Thread, interrupt, 50, 2000~~

~~; Make each newly launched thread immediately interruptible.~~

~~Thread, interrupt, 0~~

URLDownloadToFile

Downloads a file from the Internet.

URLDownloadToFile, URL, Filename

Parameters

URL	URL of the file to download. For example, <code>http://someorg.org</code> might retrieve the welcome page for that organization.
Filename	Name of the file to be created locally, which is assumed to be in <code>%A_WorkingDir%</code> if an absolute path isn't specified. Any existing file will be overwritten by the new file.

ErrorLevel

`ErrorLevel` is set to 1 if there was a problem or 0 otherwise.

Remarks

The download might appear to succeed even when the remote file doesn't exist. This is because many web servers send an error page instead of the missing file. This error page is what will be saved in place of *Filename*.

Due to the nature of this command, there might be some keyboard or mouse lag (reduced responsiveness) during the download if the script executing it has the keyboard and/or mouse hooks installed. This limitation will be resolved in a future release, if possible.

Internet Explorer 3 or greater must be installed for this function to work. Firewalls or the presence of multiple network adapters may cause this function to fail. Also, some websites may block such downloads.

The file is obtained from IE cache whenever possible. You might be able to force a (re)download from the Internet by supplying a fake cgi parameter, for example: "http://.../test.exe?fakeParam=42"

Remember that backslash (\) is used within *Filename*, in contrast to *URL*, which uses forward slash (/).

Related

[FileCopy](#)

Examples

URLDownloadToFile, http://www.someorg.org, /tmp/SomeOrg's Welcome.htm

URLDownloadToFile, http://someorg.org/archive.zip, /tmp/SomeOrg's Archive.zip

ControlClick

Sends a mouse button or mouse wheel event to *Interacts with* a control.

ControlClick, Control-or-Pos [, WinTitle, WinText, WhichButton, ClickCount, Options, ExcludeTitle, ExcludeText]

Parameters

Control-or-Pos

If this parameter is blank or omitted, the target window's topmost control will be used. Otherwise, one of the two modes below will be used.

Mode 1 (Position): In v1.0.24+, this parameter may optionally specify X and Y coordinates relative to the target window's upper left corner. The X coordinate must precede the Y coordinate and there must be at least one space or tab between them. Example: X55 Y33. If there is a control at the specified coordinates, it will be sent the click event at those exact coordinates. If there is no control, the target window itself will be sent the event (which might have no effect depending on the nature of the window). Note: In this mode, the X and Y option letters of the Options parameter are ignored.

	<p>Mode 2 (ClassNN or Text): Specify either ClassNN (the classname and instance number of the control (<i>recommended</i>) or the name/text of the control (<i>can be slow</i>), both of which can be determined via Window Spy. When using name/text, the matching behavior is determined by SetTitleMatchMode.</p> <p>By default, mode 2 takes precedence over mode 1. For example, in the unlikely event that there is a control whose text or ClassNN has the format "Xnnn Ynnn", it would be acted upon by Mode 2. To override this and use mode 1 unconditionally, specify the word Pos in Options as in the following example: ControlClick, x255 y152, WinTitle,,, Pos</p>
WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the next 3 parameters are omitted, the Last Found Window will be used. If this is the letter A and the next 3 parameters are omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number , specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support</i> .
WhichButton	<p>The button to click: LEFT, RIGHT, MIDDLE, or (in v1.0.09+) just the first letter of each of these. If omitted or blank, the LEFT button will be used.</p> <p>WheelUp (or WU) and WheelDown (or WD) are also supported. In this case, ClickCount is the number of notches to turn the wheel.</p> <p>X1 (XButton1, the 4th mouse button) and X2 (XButton2, the 5th mouse button) are also supported.</p>
ClickCount	The number of clicks to send. If omitted or blank, 1 click is sent.
Options	<p>A series of zero or more of the following option letters. Example: d x50 y25</p> <p>D: Press the mouse button down but do not release it (i.e. generate a down event). If both the D and U options are absent, a complete click (down and up) will be sent.</p> <p>U: Release the mouse button (i.e. generate an up event). This option should not be present if the D option is already present (and vice versa).</p> <p>Pos: Specify the word Pos anywhere in Options to unconditionally use the X/Y positioning mode as described in the Control or Pos parameter above.</p> <p>Xn: Specify for n the X position to click at, relative to the control's upper left corner. If unspecified, the click will occur at the horizontal center of the control. [Requires v1.0.16+]</p> <p>Yn: Specify for n the Y position to click at, relative to the control's upper left corner. If unspecified, the click will occur at the vertical center of the control. [Requires v1.0.16+]</p> <p>Use decimal (not hexadecimal) numbers for the X and Y options.</p>
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support</i> .

ErrorLevel

[ErrorLevel](#) is set to 1 if there was a problem or 0 otherwise.

Remarks

This command relies on accessibility support. Please read this section to avoid problems.

Button- or position related options are not supported because control interactions are based on atspi accessible actions, not specific buttons. This command always tries to find the action that best resembles a "Click" and executes it. If you encounter an application where ControlClick doesn't work as expected, this may be a bug - please consider opening an issue at https://github.com/phillip294/ahk_x11.

Not all applications obey a ClickCount higher than 1 for turning the mouse wheel. For those applications, use a Loop to turn the wheel more than one notch as in this example, which turns it 5 notches:

Loop, 5

ControlClick, Control, WinTitle, WinText, WheelUp

To improve reliability, a short delay is done automatically after every use of this command. That delay can be changed via SetControlDelay.

Window titles and text are always case sensitive. Hidden windows are not detected unless DetectHiddenWindows has been turned on.

Related

[SetControlDelay](#), [Control](#), [ControlGet](#), [ControlGetText](#), [ControlMove](#), [ControlGetPos](#), [ControlFocus](#), [ControlSetText](#), [ControlSend](#), [MouseClick](#)

Examples

ControlClick, OK, Some Window Title ; Clicks the OK button

ControlClick, push_button_0_1_1, Some Window Title ; Clicks the OK button based on its ClassNN which is usually faster

ControlClick, x55 y77, WinTitle ; Clicks at a set of coordinates. Note the lack of a comma between X and Y.

MouseClick

Clicks or holds a mouse button, or turns the mouse wheel. Note: The [Click](#) command is generally more flexible and easier to use.

MouseClick, WhichButton [, X, Y, ClickCount, ~~Speed~~, D|U, R]

Parameters

WhichButton	The button to click: LEFT, RIGHT, or MIDDLE, or just the first letter of each of these. WheelUp (or WU) and WheelDown (or WD) are also supported on Windows NT/2000/XP+. In this case, ClickCount is the number of notches to turn the wheel. X1 (XButton1, the 4th mouse button) and X2 (XButton2, the 5th mouse button) are also supported. For example: MouseClick, X1
X, Y	The x/y coordinates to move the mouse to, prior to clicking. Coordinates are relative to the active window unless CoordMode was used to change that. If omitted, the current position is used.
ClickCount	The number of times to click the mouse. If omitted, the button is clicked once.
Speed	The speed to move the mouse in the range 0 (fastest) to 100 (slowest). Note: a speed of 0 will move the mouse instantly. If omitted, the default speed (as set by SetDefaultMouseSpeed or 2 otherwise) will be used.
D U	If this parameter is omitted, each click will consist of a "down" event followed by an "up" event. Alternatively: D = Press the mouse button down but do not release it (i.e. generate a down-event). U = Release the mouse button (i.e. generate an up-event).
R	If this parameter is the letter R, the X and Y coordinates will be treated as offsets from the current mouse position. In other words, the cursor will be moved from its current position by X pixels to the right (left if negative) and Y pixels down (up if negative).

Remarks

To perform a shift-click or control-click, use the [Send](#) command before and after the operation as shown in these examples:

```
; Example #1:  
Send, {Control down}  
MouseClick, left, 55, 233  
Send, {Control up}
```

```
; Example #2:  
Send, {Shift down}  
MouseClick, left, 55, 233  
Send, {Shift up}
```

Some applications such as certain full-screen games may have trouble tracking the mouse if it moves too quickly. The speed parameter or [SetDefaultMouseSpeed](#) can be used to reduce the speed.

~~Not all applications obey a ClickCount higher than 1 for turning the mouse wheel. For those applications, use a Loop to turn the wheel more than one notch as in this example, which turns it 5 notches:~~
~~Loop, 5
 MouseClick, WheelUp~~

The [BlockInput](#) command can be used to prevent any physical mouse activity by the user from disrupting the simulated mouse events produced by the mouse commands.

~~There is an automatic delay after every click down and click up of the mouse. Use SetMouseDelay to change the length of the delay.~~

Related

[CoordMode](#), [SetDefaultMouseSpeed](#), [SetMouseDelay](#), [MouseClickDrag](#), [MouseGetPos](#), [MouseMove](#), [BlockInput](#)

Examples

```
; Double click at the current mouse pos:  
MouseClick, left  
MouseClick, left  
  
; Same as above:  
MouseClick, left, , , 2  
  
; Move to specified coordinates then right-click once:  
MouseClick, right, 200, 300  
  
; Here are two hotkeys that simulate the turning of the mouse wheel:  
#up::MouseClick, WheelUp, , , 2 ; Turn it by two notches.  
#down::MouseClick, WheelDown, , , 2
```

Click

Clicks a mouse button at the specified coordinates. It can also hold down a mouse button, turn the mouse wheel, or move the mouse.

Click [, Options]

Parameters

Options

If blank or omitted, a single left click is performed at the mouse cursor's current position. Otherwise, specify one or more of the following components: *Coords*, *WhichButton*, *ClickCount*, *DownOrUp*, and/or *Relative*. Separate each component from the next with at least one space, tab, and/or comma. The components can appear in any order except *ClickCount*, which must occur somewhere to the right of *Coords*, if present.

Coords: If omitted, the cursor's current position is used. Otherwise, specify the X and Y coordinates to which the mouse cursor is moved prior to clicking. For example, `Click, 100 200` clicks the left mouse button at a specific position. Coordinates are relative to the active window unless [CoordMode](#) was used to change that.

WhichButton: If omitted, it defaults to Left (the left mouse button). Otherwise, specify Left, Right, Middle (or just the first letter of each of these); or X1 (fourth button) or X2 (fifth button). For example, [Click](#), [Right](#) clicks the right mouse button at the mouse cursor's current position. Unlike [MouseClick](#), the left and right buttons behave consistently across all systems, even if the user has swapped the buttons via the system's control panel.

WhichButton can also be WheelUp or WU to turn the wheel upward (away from you), or WheelDown or WD to turn the wheel downward (toward you). [v1.0.48+]: WheelLeft (or WL) or WheelRight (or WR) may also be specified (but they have no effect on older operating systems older than Windows Vista). **ClickCount** is the number of notches to turn the wheel. However, some applications do not obey a **ClickCount** value higher than 1 for the mouse wheel. For them, use the Click command multiple times by means such as [Loop](#).

ClickCount: If omitted, it defaults to 1. Otherwise, specify the number of times to click the mouse button or turn the mouse wheel. For example, [Click](#), [2](#) performs a double-click at the mouse cursor's current position. If **Coords** is specified, **ClickCount** must appear after it. Specify zero (0) to move the mouse without clicking; for example, [Click](#), [100 200 0](#).

DownOrUp: If omitted, each click consists of a down-event followed by an up-event. Otherwise, specify the word Down (or the letter D) to press the mouse button down without releasing it. Later, use the word Up (or the letter U) to release the mouse button. For example, [Click](#), [Down](#) presses down the left mouse button and holds it.

Relative: If omitted, the X and Y coordinates will be used for absolute positioning. Otherwise, specify the word Rel or Relative to treat the coordinates as offsets from the current mouse position. In other words, the cursor will be moved from its current position by X pixels to the right (left if negative) and Y pixels down (up if negative).

Remarks

The Click command is generally preferred over MouseClick because it automatically compensates if the user has swapped the left and right mouse buttons via the system's control panel.

The Click command uses the sending method set by SendMode. To override this mode for a particular click, use a specific Send command in combination with {Click}, as in this example: [SendEvent](#) ([Click](#) 100 200).

To perform a shift-click or control-click, [clicking via Send](#) is generally easiest. For example:

```
Send +{Click 100 200} ; Shift+LeftClick  
Send ^{Click 100 200 Right} ; Control+RightClick
```

Unlike [Send](#), the Click command does not automatically release the modifier keys (Ctrl, Alt, Shift, and Win). For example, if Ctrl is currently down, Click would produce a control-click but Send {Click} would produce a normal click.

The SendPlay mode is able to successfully generate mouse events in a broader variety of games than the other modes. In addition, some applications and games may have trouble tracking the mouse if it moves too quickly, in which case SetDefaultMouseSpeed can be used to reduce the speed (but only in SendEvent mode).

The [BlockInput](#) command can be used to prevent any physical mouse activity by the user from disrupting the simulated mouse events produced by the mouse commands. However, this is generally not needed for the [SendInput](#) and [SendPlay](#) modes because they automatically postpone the user's physical mouse activity until afterward.

There is an automatic delay after every click-down and click-up of the mouse (except for [SendInput mode](#) and for turning the mouse wheel). Use [SetMouseDelay](#) to change the length of the delay.

Related

[Send {Click}](#), [SendMode](#), [CoordMode](#), [SetDefaultMouseSpeed](#), [SetMouseDelay](#), [MouseClick](#), [MouseClickDrag](#), [MouseMove](#), [ControlClick](#), [BlockInput](#)

Examples

Clicks the left mouse button at the mouse cursor's current position.

```
Click
```

Clicks the left mouse button at a specific position.

```
Click, 100 200
```

Moves the mouse cursor to a specific position without clicking.

```
Click, 100 200 0
```

Clicks the right mouse button at a specific position.

```
Click, 100 200 Right
```

Performs a double-click at the mouse cursor's current position.

```
Click, 2
```

Presses down the left mouse button and holds it.

```
Click, Down
```

Releases the right mouse button.

```
Click, Up Right
```

MouseClickDrag

Clicks and holds the specified mouse button, moves the mouse to the destination coordinates, then releases the button.

```
MouseClickDrag, WhichButton, X1, Y1, X2, Y2 [, -Speed, R]
```

Parameters

WhichButton	The button to click: LEFT, RIGHT, MIDDLE, or (in v1.0.09+) just the first letter of each of these. In v1.0.10+, X1 (XButton1, the 4th mouse button) and X2 (XButton2, the 5th mouse button) are also supported on Windows 2000/XP+. For example: MouseClickDrag, X1, ...
X1, Y1	The x/y coordinates of the drag's starting position (the mouse will be moved to these coordinates right before the drag is started). Coordinates are relative to the active window unless CoordMode was used to change that. If omitted, the mouse's current position is used.
X2, Y2	The x/y coordinates to drag the mouse to (that is, while the button is held down). Coordinates are relative to the active window unless CoordMode was used to change that.
Speed	The speed to move the mouse in the range 0 (fastest) to 100 (slowest). Note: a speed of 0 will move the mouse instantly. If omitted, the default speed (as set by SetDefaultMouseSpeed or 2 otherwise) will be used.
R	If this parameter is the letter R, the X1 and Y1 coordinates will be treated as offsets from the current mouse position. In other words, the cursor will be moved from its current position by X1 pixels to the right (left if negative) and Y1 pixels down (up if negative). Similarly, the X2 and Y2 coordinates will be treated as offsets from the X1 and Y1 coordinates. For example, the following would first move the cursor down and to the right by 5 pixels from its starting position, and then drag it from that position down and to the right by 10 pixels: MouseClickDrag, Left, 5, 5, 10, 10, , R

Remarks

Some applications such as certain full-screen games may have trouble tracking the mouse if it moves too quickly. The *Speed* parameter or [SetDefaultMouseSpeed](#) can be used to reduce the speed.

The [BlockInput](#) command can be used to prevent any physical mouse activity by the user from disrupting the simulated mouse events produced by the mouse commands.

There is an automatic delay after every click-down and click-up of the mouse. This delay also occurs after the movement of the mouse during the drag operation. Use [SetMouseDelay](#) to change the length of the delay.

Related

[CoordMode](#), [SetDefaultMouseSpeed](#), [SetMouseDelay](#), [MouseClick](#), [MouseGetPos](#), [MouseMove](#), [BlockInput](#)

Example

MouseClickDrag, left, 0, 200, 600, 400

[MouseGetPos](#)

Retrieves the current position of the mouse cursor, and optionally which window and control it is hovering over.

MouseGetPos, [OutputVarX, OutputVarY, OutputVarWin, OutputVarControl]

Parameters

OutputVarX/Y	The names of the variables in which to store the X and Y coordinates. The retrieved coordinates are relative to the active window unless CoordMode was used to change to screen coordinates; currently always screen wide
OutputVarWin	This optional parameter is the name of the variable in which to store the unique ID number of the window under the mouse cursor. If the window cannot be determined, this variable will be made blank. The window does not have to be active to be detected. Hidden windows cannot be detected.
OutputVarControl	This optional parameter is the name of the variable in which to store the name of the control under the mouse cursor. If the control cannot be determined, this variable will be made blank. The names of controls should always match those shown by the version of Window Spy. However, unlike Window Spy, the window under the mouse cursor does not have to be active for a control to be detected. <i>This option relies on accessibility support. Please read this section to avoid problems.</i>

Remarks

Any of the output variables may be omitted if the corresponding information is not needed.

Related

[CoordMode](#), [WinGet](#), [SetDefaultMouseSpeed](#), [MouseClick](#), [MouseClickDrag](#), [MouseMove](#)

Example

```
MouseGetPos, xpos, ypos  
Msgbox, The cursor is at X%xpos% Y%ypos%.
```

```
; This example allows you to move the mouse around to see  
; the title of the window currently under the cursor:  
#Persistent  
SetTimer, WatchCursor, 100  
return
```

```
WatchCursor:  
MouseGetPos, , , id, control  
WinGetTitle, title, ahk_id %id%  
WinGetClass, class, ahk_id %id%  
ToolTip, ahk_id %id%`nahk_class %class%`n%title%`nControl: %control%  
return
```

MouseMove

Moves the mouse cursor.

MouseMove, X, Y [, Speed, R]

Parameters

X, Y	The x/y coordinates to move the mouse to. Coordinates are relative to the active window unless CoordMode was used to change that.
Speed	The speed to move the mouse in the range 0 (fastest) to 100 (slowest). Note: a speed of 0 will move the mouse instantly. If omitted, the default speed (as set by SetDefaultMouseSpeed or 2 otherwise) will be used.
R	If this parameter is the letter R, the X and Y coordinates will be treated as offsets from the current mouse position. In other words, the cursor will be moved from its current position by X pixels to the right (left if negative) and Y pixels down (up if negative).

Remarks

Some applications such as certain full-screen games may have trouble tracking the mouse if it moves too quickly. The *speed* parameter or [SetDefaultMouseSpeed](#) can be used to reduce the speed.

The [BlockInput](#) command can be used to prevent any physical mouse activity by the user from disrupting the simulated mouse events produced by the mouse commands.

There is an automatic delay after every movement of the mouse. Use [SetMouseDelay](#) to change the length of the delay.

Related

[CoordMode](#), [SetDefaultMouseSpeed](#), [SetMouseDelay](#), [MouseClick](#), [MouseClickDrag](#), [MouseGetPos](#), [BlockInput](#)

Example

```
; Move the mouse to a new position:  
MouseMove, 200, 100  
  
; Move the mouse slowly (speed 50 vs. 2) by 20 pixels to the right and 30 pixels down  
; from its current location:  
MouseMove, 20, 30, 50, R
```

#SetDefaultMouseSpeed

Sets the mouse speed that will be used if unspecified in MouseMove/Click/Drag.

SetDefaultMouseSpeed, Speed

Parameters

Speed

The speed to move the mouse in the range 0 (fastest) to 100 (slowest). Note: a speed of 0 will move the mouse instantly.

Remarks

If this command isn't used, the default mouse speed is 2.

The commands **MouseClick**, **MouseMove**, and **MouseClickDrag** all have a parameter to override the default mouse speed.

The built-in variable **A_DefaultMouseSpeed** contains the current setting.

Every newly launched hotkey, custom menu item, or timed subroutine starts off fresh with the default setting for this command. That default may be changed by using this command in the auto-execute section (top part) of the script.

Related

MouseClick, **MouseMove**, **MouseClickDrag**, **SetWinDelay**, **SetControlDelay**, **SetKeyDelay**, **SetKeyDelay**

Example

SetDefaultMouseSpeed, 0 ; Move the mouse instantly like AutoIt2.

SetMouseDelay

Sets the delay that will occur after each mouse movement or click.

SetMouseDelay, Delay

Parameters

Delay

Time in milliseconds. Use -1 for no delay at all and 0 for the smallest possible delay.

Remarks

A short delay (sleep) is automatically and invisibly done after every mouse movement or click generated by [MouseMove](#), [MouseClick](#), and [MouseClickDrag](#). This is done to improve the reliability of scripts because a window sometimes can't keep up with a rapid flood of mouse events.

~~Due to the granularity of the OS's time keeping system, delays might be rounded up to the nearest multiple of 10. For example, a delay between 1 and 10 (inclusive) is equivalent to 10 on Windows XP (and probably NT & 2k).~~

A delay of 0 internally executes a Sleep(0), which yields the remainder of the script's timeslice to any other process that may need it. If there is none, Sleep(0) will not sleep at all. By contrast, a delay of -1 will never sleep.

If unset, the default delay is 10.

The built-in variable **A_MouseDelay** contains the current setting.

Every newly launched [hotkey](#), [custom menu item](#), or [timed](#) subroutine starts off fresh with the default setting for this command. That default may be changed by using this command in the auto-execute section (top part) of the script.

Related

[MouseMove](#), [MouseClick](#), [MouseClickDrag](#), [SetKeyDelay](#), [SetControlDelay](#), [SetWinDelay](#), [SetBatchLines](#)

Example

```
SetMouseDelay, 0
```

#Process [v1.0.18+]

Performs one of the following operations on a process: checks if it exists; changes its priority; closes it; waits for it to close.

Process, Cmd, PID or Name [, Param3]

Parameters

	<p>One of the following words:</p> <p>Exist: Sets ErrorLevel to the Process ID (PID) if a matching process exists, or 0 otherwise.</p> <p>Close: If a matching process is successfully terminated, ErrorLevel is set to its former Process ID (PID). Otherwise (there was no matching process or there was a problem terminating it), it is set to 0. Since the process will be abruptly terminated — possibly interrupting its work at a critical point or resulting in the loss of unsaved data in its windows (if it has any) — this method should be used only if a process cannot be closed by using WinClose on one of its windows.</p> <p>Priority: Changes the priority (as seen in Windows Task Manager) of the first matching process to <i>Param3</i> and sets ErrorLevel to its Process ID (PID). If the <i>PID or Name</i> parameter is blank, the script's own priority will be changed. If there is no matching process or there was a problem changing its priority, ErrorLevel is set to 0.</p> <p><i>Param3</i> should be one of the following letters or words: L (or Low), B (or BelowNormal), N (or Normal), A (or AboveNormal), H (or High), R (or Realtime). Since BelowNormal and AboveNormal are not supported on Windows 95/98/Me/NT4, normal will be automatically substituted for them on those operating systems. Note: Any process not designed to run at Realtime priority might reduce system stability if set to that level.</p> <p>Wait: Waits up to <i>Param3</i> seconds (can contain a decimal point) for a matching process to exist. If <i>Param3</i> is omitted, the command will wait indefinitely. If a matching process is discovered, ErrorLevel is set to its Process ID (PID). If the command times out, ErrorLevel is set to 0.</p> <p>WaitClose: Waits up to <i>Param3</i> seconds (can contain a decimal point) for ALL matching processes to close. If <i>Param3</i> is omitted, the command will wait indefinitely. If all matching processes are closed, ErrorLevel is set to 0. If the command times out, ErrorLevel is set to the Process ID (PID) of the first matching process that still exists.</p>
Cmd	
PID or Name	<p>This parameter can be either a number (the PID) or a process name as described below. It can also be left blank to change the priority of the script itself.</p> <p>PID: The Process ID, which is a number that uniquely identifies one specific process (this number is valid only during the lifetime of that process). The PID of a newly launched process can be determined via the Run command. Similarly, the PID of a window can be determined with WinGet. The Process command itself can also be used to discover a PID.</p>

Name: The name of a process is usually the same as its executable (without path), e.g. notepad.exe or winword.exe. Since a name might match multiple running processes, only the first process will be operated upon.
The name is not case sensitive.

Param3

See Cmd above for details.

ErrorLevel

See Cmd above for details.

Remarks

For Wait and WaitClose: Processes are checked every 100 milliseconds; the moment the condition is satisfied, the command will cease waiting. In other words, rather than waiting for the timeout to expire, it will immediately set ErrorLevel as described above and then continue execution of the script. Also, while the command is in a waiting state, new threads can be launched via hotkey, custom menu item, or timer.

To work under Windows NT 4.0, this command requires the file PSAPI.DLL, which is normally already present in the AutoHotkey installation directory (i.e. no extra installation steps should be needed even for NT).

Related

Run, WinGet, WinClose, WinKill, WinWait, WinWaitClose, IfWinExist

Examples

```
; Example #1.
Run Notepad.exe, , , NewPID
Process, priority, %NewPID%, High
MsgBox The newly launched notepad's PID is %NewPID%.
```

```
; Example #2.
Process, wait, Notepad.exe, 5.5
NewPID %ErrorLevel% ; Save the value immediately since ErrorLevel is often changed.
if NewPID = 0
+
    MsgBox The specified process did not appear within 5.5 seconds.
    return
+
; Otherwise:
MsgBox A matching process has appeared (Process ID is %NewPID%).
Process, priority, %NewPID%, Low
Process, priority, , High ; Have the script set itself to high priority.
```

```
WinClose Untitled Notepad
Process, WaitClose, %NewPID%, 5
If ErrorLevel <> 0, The PID still exists.
    MsgBox The process did not close within 5 seconds.
```

```
; Example #3: A hotkey to change the priority of the active window's process.
#2.., Win+Z hotkey
WinGet, active_pid, PID, A
WinGetTitle, active_title, A
Gui, 5.Add, Text,, Press ESCAPE to cancel, or double click a new priority level for the following window: %active_title%
Gui, 5.Add, ListBox, vMyListBox r5, Normal+High+Low+BelowNormal+AboveNormal
Gui, 5.Add, Button, default, OK
Gui, 5>Show,, Set Priority
return

 GuiEscape:
 GuiClose:
 Gui, Destroy
return

MyListBox:
If A_GuiControlEvent <> DoubleClick
    return
; else fall through to the next label.
 GuiControlSet, MyListBox
 Gui, Destroy
Process, Priority, %active_pid%, %MyListBox%
If ErrorLevel = 0
    MsgBox Error: Its priority could not be changed to "%MyListBox%".
Else
    MsgBox Success: Its priority was changed to "%MyListBox%".
return
```

Run / RunWait

Runs an external program. Unlike Run, RunWait will wait until the program finishes before continuing. *RunWait only reliably waits when the program is an executable file or program.*

Run, Target [, WorkingDir, Max|Min|Hide|UseErrorLevel, OutputVarPID, *OutputVarStdout*, *OutputVarStderr*]

Parameters

Target	A document, URL, executable file (.exe, .com, .bat, etc. <i>a file with executable rights</i>), shortcut (link soft link), or system verb to launch (see remarks). If <i>Target</i> is a local file and no path was specified with it, %A_WorkingDir% will be searched first. If no matching file is found there, the system will search for and launch the file if it is integrated ("known"), e.g. by being contained in one of the PATH folders. To pass parameters, add them immediately after the program or document name. If a parameter contains spaces, it is safest to enclose it in double quotes (even though it may work without them in some cases).
WorkingDir	The working directory for the launched item. Do not enclose the name in double quotes even if it contains spaces. If omitted, the script's own working directory (%A_WorkingDir%) will be used.
Max Min Hide UseErrorLevel	If omitted, <i>Target</i> will be launched normally. Alternatively, it can contain one or more of these words: Max: launch maximized Min: launch minimized Hide: launch hidden (cannot be used in combination with either of the above) UseErrorLevel: This word can be specified alone or in addition to one of the above words (by separating it from the other word with a space). When present, if the launch succeeds for the Run command (not RunWait), ErrorLevel is set to 0. If the launch fails for either Run or RunWait, the following alternate behavior will occur: 1) No warning dialog will be displayed; 2) The current thread will <u>not</u> be terminated; 3) ErrorLevel will be set to the word ERROR. <i>This only works when the target is a valid, executable command! So it doesn't work with text files or web urls etc.</i> Some applications (e.g. Calc.exe) do not obey the requested startup state and thus Max/Min/Hide will have no effect.
OutputVarPID	For the Run command, this optional parameter is the name of the variable in which to store the newly launched program's unique Process ID (PID) . The variable will be made blank if the PID could not be determined, which usually happens if a system verb, document, or shortcut is launched rather than a direct executable file.
OutputVarStdout	<i>For the RunWait command, this optional parameter is the name of the variable in which to store the program's full text output. This only works when the target is a valid, executable command.</i>
OutputVarStderr	<i>For the RunWait command, this optional parameter is the name of the variable in which to store the program's full error output. This only works when the target is a valid, executable command.</i>

Remarks

Unlike Run, RunWait will wait until *Target* is closed or exits, at which time [ErrorLevel](#) will be set to the program's exit code (as a signed 32-bit value). Some programs will appear to return immediately even though they are still running; these programs spawn another process.

If *Target* cannot be launched, an error window will be displayed, after which the script will behave as though an [Exit](#) command were encountered. To prevent this, include the string **UseErrorLevel** in the 3rd parameter.

Performance may be slightly improved if *Target* is an exact path, e.g. Run, mousepad "/a/b/Test.txt" rather than Run, /a/b/Test.txt

System verbs correspond to actions available in a file's right-click menu in the Explorer. If a file is launched without a verb, the default verb (usually "open") for that particular file type will be used. If specified, the verb should be followed by the name of the target file. The following verbs are currently supported:

properties	Displays the Explorer's properties window for the indicated file. Example: Run, properties C:\autoexec.bat Note: The properties window will automatically close when the script terminates. To prevent this, use WinWaitClose to have the script wait for the properties window to close.
find	Opens an instance of the Explorer's Search Companion or Find File window at the indicated folder. Example: Run, find D\
explore	Opens an instance of Explorer at the indicated folder. Example: Run, explore %ProgramFiles%. You can do that, but folders are being opened just fine without this verb, so it's unnecessary.
edit	Opens the indicated file for editing. It might not work if the indicated file's type does not have an "edit" action associated with it. Example: Run, edit /a/b/My File.txt
open	Opens the indicated file (normally not needed because it is the default action for most file types). Example: Run, open My File.txt
print	Prints the indicated file with the associate application, if any. Example: Run, print My File.txt This currently just sends the file to `lp` which expects you to have a default printer set up. Untested.

While RunWait is in a waiting state, new [threads](#) can be launched via [hotkey](#), [custom menu item](#), or [timer](#).

To launch a program with environment variables set, use [EnvSet](#). Just prefixing them like `Run, VAR=VALUE cmd` won't work.

Native shell features like redirect (>) or background jobs (&) are not available. Also, ahk variables can not span multiple params. For example, if variable abc contains spaces, then `Run, %abc%` would treat the whole contents of abc as the command path and not split it up. If you really need to work around this, you can invoke the shell directly, e.g. Run, bash -c '%abc%'.

Related

[RunAs](#), [Process](#), [Exit](#)

Examples

```
Run, mousepad, /a/b, max
```

```
Run, ReadMe.doc, , Max UseErrorLevel ; Launch maximized and don't display dialog if it fails.
if ErrorLevel = ERROR
    MsgBox The document could not be launched.
```

```
RunWait, echo -n Text to stdout && echo -n Text to stderr >&2 && exit 5,,, stdout, stderr
Echo, Exit code: %errorlevel%, stdout: %stdout%, stderr: %stderr%
```

```
RunWait, ls / >>/a/b/DirTest.txt, , min
Run, /a/b/DirTest.txt
RunWait, ls ~,,, out
MsgBox %out%
```

```
Run, www.autohotkey.com ; i.e. any URL can be launched.
```

```
Run, mailto:support@autohotkey.com ; This should open the default e-mail application.
```

#RunAs

Specifies a set of user credentials to use for all subsequent uses of [Run](#) and [RunWait](#). Also works for graphical applications. Note: Using this command may lead to the specified password to appear in the task manager, visible to other programs.

RunAs [, User, Password, Domain]

Parameters

User	If this and the other parameters are all omitted, the RunAs feature will be turned off, which restores Run and RunWait to their default behavior. Otherwise, this is the username under which new processes will be created.
Password	<i>User's password. Must not be blank.</i>
Domain	<i>User's domain. To use a local account, leave this blank. If that fails to work, try using @YourComputerName.</i>

Remarks

This command does nothing other than notify AutoHotkey to use (or not use) alternate user credentials for all subsequent uses of [Run](#) and [RunWait](#).

[ErrorLevel](#) is not changed by this command. If an invalid *User*, *Password*, or *Domain* is specified, [Run](#) and [RunWait](#) will display an error message explaining the problem.

While the RunAs feature is in effect, [Run](#) and [RunWait](#) will not be able to launch documents, URLs, or system verbs. In other words, the file to be launched must be an executable file.

The "Secondary Logon" service must be set to manual or automatic for this command to work (the OS should automatically start it upon demand if set to manual).

Related

[Run](#), [RunWait](#)

Example

RunAs, Administrator, MyPassword
Run, RegEdit.exe
RunAs ; Reset to normal behavior.

Shutdown

Shuts down, restarts, or logs off the system.

Shutdown, Code

Parameters

Code	A combination of shutdown codes. See "remarks".
------	---

Remarks

The shutdown code is a combination of the following values:

Logoff	0
Shutdown	1
Reboot	2
Force	4
Power down	8

Add the required values together. e.g. if you want to shutdown and power down the code would be **9** (shutdown + power down = $1 + 8 = 9$).

The "Force" value forces all open applications to close. It should only be used in an emergency because it may cause any open applications to lose data. *You may be asked for a password via popup.*

The "Power down" value shuts down the system and turns off the power.

A script can detect when the system is shutting down or the user is logging off via [OnExit](#).

Related

[Run](#), [ExitApp](#), [OnExit](#)

Example

; Force a reboot (reboot + force = 2 + 4 = 6):
Shutdown, 6

#RegDelete

Deletes a subkey or value from the registry.

~~RegDelete, RootKey, SubKey [, ValueName]~~

#RegRead

Reads a value from the registry.

~~RegRead, OutputVar, RootKey, SubKey [, ValueName]~~

~~(Obsolete 5 param method): RegRead, OutputVar, ValueType, RootKey, SubKey [, ValueName]~~

#RegWrite

Writes a value to the registry.

~~RegWrite, ValueType, RootKey, SubKey [, ValueName, Value]~~

SoundGet

Retrieves various settings from a sound device (master mute, master volume, etc.)

~~SoundGet, OutputVar [, ComponentType, ControlType, DeviceNumber]~~

Parameters

OutputVar	The name of the variable in which to store the retrieved setting, which is either a floating point number between 0 and 100 (inclusive) or the word ON or OFF (used only for <i>ControlTypes</i> ONOFF, MUTE, MONO , LOUDNESS , STEREOENH , and BASSBOOST). The variable will be made blank if there was a problem retrieving the setting. The format of the floating point number, such as its decimal places, is determined by <i>SetFormat</i> .
ComponentType	If omitted or blank, it defaults to the word MASTER. Otherwise, it can be one of the following words: MASTER (synonymous with SPEAKERS), DIGITAL , LINE , MICROPHONE , SYNTH , CD , TELEPHONE , PCSPEAKER , WAVE , AUX , or ANALOG . If the sound device lacks the specified ComponentType, ErrorLevel will indicate the problem. The component labeled Auxiliary in some mixers might be accessible as ANALOG rather than AUX. If a device has more than one instance of <i>ComponentType</i> (two of type LINE, for example) and you wish to use one other than the first, append a colon and a number to this parameter. For example: Analog:2 is the second analog input.
ControlType	If omitted or blank, it defaults to VOLUME. Otherwise, it can be one of the following words: VOLUME (or VOL), ONOFF, MUTE, MONO , LOUDNESS , STEREOENH , BASSBOOST , PAN , QSOUNDPAN , BASS , TREBLE , or EQUALIZER . If the specified ComponentType lacks the specified ControlType, ErrorLevel will indicate the problem.
DeviceNumber	If this parameter is omitted, it defaults to 1 (the first sound device), which is usually the system's default device for recording and playback. Specify a number higher than 1 to operate upon a different sound device.

ErrorLevel

ErrorLevel is set to 0 if the command succeeded. Otherwise, it is set to one of the following phrases:

Invalid Control Type or Component Type
Can't Open Specified Mixer
Mixer Doesn't Support This Component Type
Mixer Doesn't Have That Many of That Component Type
Component Doesn't Support This Control Type
Can't Get Current Setting

Remarks

The operating system's mixer panel or the software that came with your sound device may give you some idea of what features it supports. For example, if the Play Control or Master Volume area has an Advanced button, pressing that button may reveal which *ControlTypes* other than VOLUME (such as BASS & TREBLE) are supported when *ComponentType* is MASTER.

Use [SoundSet](#) to change a setting.

Related

[SoundSet](#), [SoundGetWaveVolume](#), [SoundSetWaveVolume](#), [SoundPlay](#)

Examples

```
SoundGet, master_volume  
MsgBox, Master volume is %master_volume% percent.
```

```
SoundGet, master_mute, , mute  
MsgBox, Master Mute is currently %master_mute%.
```

```
SoundGet, bass_level, Master, bass  
if ErrorLevel <> 0  
    MsgBox, Error Description: %ErrorLevel%  
else  
    MsgBox, The BASS level for MASTER is %bass_level% percent.
```

```
SoundGet, microphone_mute, Microphone, mute  
if microphone_mute = Off  
    MsgBox, The microphone is not muted.
```

#SoundGetWaveVolume

Changes the wave output volume for a sound device.

SoundGetWaveVolume, OutputVar [, DeviceNumber]

Parameters

OutputVar

The name of the variable in which to store the retrieved volume level, which is a floating point number between 0 and 100 inclusive. The variable will be made blank if there was a problem retrieving the volume. The format of the floating point number, such as its decimal places, is determined by SetFormat.

DeviceNumber

If this parameter is omitted, it defaults to 1 (the first sound device), which is usually the system's default device for recording and playback. Specify a number higher than 1 to operate upon a different sound device.

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

Remarks

The current wave output volume level can be set via SoundSetWaveVolume. Settings such as Master Volume, Synth, Microphone, Mute, Treble, and Bass can be set and retrieved using SoundSet and SoundGet.

Related

SoundSetWaveVolume, SoundSet, SoundGet, SoundPlay

Examples

SoundGetWaveVolume, OutputVar

MsgBox, The current wave output volume level is %OutputVar%`%.

SoundPlay

Play a sound, video, or other supported file type.

SoundPlay, Filename [, wait]

Parameters

Filename	The name of the file to be played, which is assumed to be in %A_WorkingDir% if an absolute path isn't specified.
wait	If omitted, the script's current thread will continue running while the file is playing. If it's 1 or the word WAIT, the current thread will wait until the file is finished playing. Even while waiting, new threads can be launched via hotkey , custom menu item , or timer .

ErrorLevel

[ErrorLevel](#) is set to 1 if there was a problem or 0 otherwise.

Remarks

All Windows OSes should be able to play .wav files. However, other files (.mp3, .avi, etc.) might not be playable if the right codecs or features aren't installed on the OS. *When in doubt, install the ffmpeg package for your system. It will guarantee that this command works with any type of audio/video file.*

If a file is playing and the current script or a different script plays a second file, the first file will be stopped so that the second one can play.

To stop a file that is currently playing, use SoundPlay on a nonexistent filename as in this example: SoundPlay, Nonexistent.avi

If the script is terminated, any currently-playing file that it started will stop.

Related

[SoundGet](#), [SoundSet](#), [SoundGetWaveVolume](#), [SoundSetWaveVolume](#), [Threads](#)

Example

SoundSet

Changes various settings of a sound device (master mute, master volume, etc.)

SoundSet, NewSetting [, ComponentType, ControlType, DeviceNumber]

Parameters

NewSetting	<p>Percentage number between -100 and 100 inclusive (it can be a floating point number). If the number begins with a plus or minus sign, the current setting will be adjusted up or down by the indicated amount. Otherwise, the setting will be set explicitly to the level indicated by <i>NewSetting</i>.</p> <p>For <i>ControlTypes</i> with only two possible settings -- namely ONOFF, MUTE, MONO, LOUDNESS, STEREOENH, and BASSBOOST -- any positive number will turn on the setting and a zero will turn it off. However, if the number begins with a plus or minus sign, the setting will be toggled (set to the opposite of its current state).</p>
ComponentType	<p>If omitted or blank, it defaults to the word MASTER. Otherwise, it can be one of the following words: MASTER (synonymous with SPEAKERS), DIGITAL, LINE, MICROPHONE, SYNTH, CD, TELEPHONE, PCSPEAKER, WAVE, AUX, or ANALOG. If the sound device lacks the specified <i>ComponentType</i>, ErrorLevel will indicate the problem.</p> <p>The component labeled Auxiliary in some mixers might be accessible as ANALOG rather than AUX.</p> <p>If a device has more than one instance of <i>ComponentType</i> (two of type LINE, for example) and you wish to use one other than the first, append a colon and a number to this parameter. For example: Analog:2 is the 2nd analog input.</p>
ControlType	<p>If omitted or blank, it defaults to VOLUME. Otherwise, it can be one of the following words: VOLUME (or VOL), ONOFF, MUTE, MONO, LOUDNESS, STEREOENH, BASSBOOST, PAN, QSOUNDPAN, BASS, TREBLE, or EQUALIZER. If the specified <i>ComponentType</i> lacks the specified <i>ControlType</i>, ErrorLevel will indicate the problem.</p>
DeviceNumber	<p>If this parameter is omitted, it defaults to 1 (the first sound device), which is usually the system's default device for recording and playback. Specify a number higher than 1 to operate upon a different sound device.</p>

ErrorLevel

ErrorLevel is set to 0 if the command succeeded. Otherwise, it is set to one of the following phrases:

~~Invalid Control Type or Component Type~~

Can't Open Specified Mixer
Mixer Doesn't Support This Component Type
Mixer Doesn't Have That Many of That Component Type
Component Doesn't Support This Control Type
Can't Get Current Setting
Can't Change Setting

Remarks

The operating system's mixer panel or the software that came with your sound device may give you some idea of what features it supports. For example, if the Play Control or Master Volume area has an Advanced button, pressing that button may reveal which *ControlTypes* other than VOLUME (such as BASS & TREBLE) are supported when *ComponentType* is MASTER.

Use [SoundGet](#) to retrieve the current value of a setting.

Related

[SoundGet](#), [SoundGetWaveVolume](#), [SoundSetWaveVolume](#), [SoundPlay](#)

Examples

```
SoundSet, 50 ; Set the master volume to 50%
SoundSet, 1, Microphone, mute ; mute the microphone
SoundSet, +1, , mute ; Toggle the master mute (set it to the opposite state)
SoundSet, +20, Master, bass ; Increase bass level by 20%.
if ErrorLevel <> 0
    MsgBox, The BASS setting is not supported for MASTER.
```

#SoundSetWaveVolume

~~Changes the wave output volume for a sound device.~~

~~SoundSetWaveVolume, Percent [, DeviceNumber]~~

Parameters

Percent

Percentage number between -100 and 100 inclusive (it can be a floating point number). If the number begins with a plus or minus sign, the **current volume level** will be adjusted up or down by the indicated amount. Otherwise, the volume will be set explicitly to the level indicated by Percent.

DeviceNumber

If this parameter is omitted, it defaults to 1 (the first sound device), which is usually the system's default device for recording and playback. Specify a number higher than 1 to operate upon a different sound device.

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

Remarks

The current wave output volume level can be retrieved via SoundGetWaveVolume. Settings such as Master Volume, Synth, Microphone, Mute, Treble, and Bass can be set and retrieved using SoundSet and SoundGet.

Related

[SoundGetWaveVolume](#), [SoundSet](#), [SoundGet](#), [SoundPlay](#)

Examples

SoundSetWaveVolume, 50 ; Set the volume to its half-way point.

SoundSetVolume, -10 ; Decrease the current level by 10 (e.g. 80 would become 70).

SoundSetVolume, +20 ; Increase the current level by 20.

#FormatTime [v1.0.24+]

Transforms a YYYYMMDDHH24MISS timestamp into the specified date/time format.

FormatTime, OutputVar [, YYYYMMDDHH24MISS, Format]

Parameters

OutputVar	The name of the variable in which to store the result.
YYYYMMDD...	Leave this parameter blank to use the current local date and time. Otherwise, specify all or the leading part of a timestamp in the YYYYMMDDHH24MISS format. If the date and/or time portion of the timestamp is invalid — such as February 29th of a non-leap year — the date and/or time will be omitted from OutputVar. Although only years between 1601 and 9999 are supported, a formatted time can still be produced for earlier years as long as the time portion is valid.
Format	If omitted, it defaults to the time followed by the long date, both of which will be formatted according to the current user's locale. Example: 4:55 PM Saturday, November 27, 2004 Otherwise, specify one or more of the date-time formats below, along with any literal spaces and punctuation in between (commas do not need to be escaped; they can be used normally). In the following example, note that MM must be capitalized: MM/dd/yyyy hh:mm tt

Date Formats (case-sensitive)

d	Day of the month without leading zero (1 - 31)
dd	Day of the month with leading zero (01 - 31)
ddd	Abbreviated name for the day of the week (e.g. Mon) in the current user's language
ddd	Full name for the day of the week (e.g. Monday) in the current user's language
M	Month without leading zero (1 - 12)
MM	Month with leading zero (01 - 12)
MMM	Abbreviated month name (e.g. Jan) in the current user's language
MMMM	Full month name (e.g. January) in the current user's language
Y	Year without century, without leading zero (0 - 99)
YY	Year without century, with leading zero (00 - 99)
YYYY	Year with century. Example: 2005
gg	Period/era string for the current user's locale (blank if none)

Time Formats (case-sensitive)

h	Hours without leading zero; 12-hour format (1 - 12)
---	---

hh	Hours with leading zero; 12-hour format (01–12)
H	Hours without leading zero; 24-hour format (0–23)
HHH	Hours with leading zero; 24-hour format (00–23)
m	Minutes without leading zero (0–59)
mm	Minutes with leading zero (00–59)
s	Seconds without leading zero (0–59)
ss	Seconds with leading zero (00–59)
t	Single-character time marker, such as A or P (depends on locale)
tt	Multi-character time marker, such as AM or PM (depends on locale)

The following formats must be used alone; that is, with no other formats or text present in the *Format* parameter. These formats are not case sensitive.

(Blank)	Leave <i>Format</i> blank to produce the time followed by the long date. For example, in some locales it might appear as 4:55 PM Saturday, November 27, 2004
Time	Time representation for the current user's locale, such as 5:26 PM
ShortDate	Short date representation for the current user's locale, such as 02/29/04
LongDate	Long date representation for the current user's locale, such as Friday, April 23, 2004
YearMonth	Year and month format for the current user's locale, such as February, 2004
YDay	Day of the year without leading zeros (1–366)
YDay0	Day of the year with leading zeros (001–366)
WDay	Day of the week (1–7). Sunday is 1.
YWeek	The ISO 8601 full year and week number. Example: 200453. If the week containing January 1st has four or more days in the new year, it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1. Both January 4th and the first Thursday of January are always in week 1.

Additional Options

The following options can appear inside the *YYYYMMDDHH24MISS* parameter immediately after the timestamp (if there is no timestamp, they may be used alone). In the following example, note the lack of commas between the last four items: FormatTime, OutputVar, 20040228 LSys D1-D4

R: Reverse. Have the date come before the time (meaningful only when *Format* is blank).

Ln: If this option is not present, the current user's locale is used to format the string. To use the system's locale instead, specify LSys. To use a specific locale, specify the letter L followed by a hexadecimal or decimal locale identifier (LCID). For information on how to construct an LCID, search www.microsoft.com for the following phrase: Locale Identifiers

~~Dn: Date options. Specify for n one of the following numbers:~~

~~0: Force the default options to be used. This also causes the short date to be in effect.~~

~~1: Use short date (meaningful only when Format is blank; not compatible with 2 and 8).~~

~~2: Use long date (meaningful only when Format is blank; not compatible with 1 and 8).~~

~~4: Use alternate calendar (if any).~~

~~8: Use Year-Month format (meaningful only when Format is blank; not compatible with 1 and 2). Requires Windows 2000 or later.~~

~~0x10: Add marks for left-to-right reading order layout. Requires Windows 2000 or later.~~

~~0x20: Add marks for right-to-left reading order layout. Requires Windows 2000 or later.~~

~~0x80000000: Do not obey any overrides the user may have in effect for the system's default date format.~~

~~0x40000000: Use the system ANSI code page for string translation instead of the locale's code page.~~

~~Tn: Time options. Specify for n one of the following numbers:~~

~~0: Force the default options to be used. This also causes minutes and seconds to be shown.~~

~~1: Omit minutes and seconds.~~

~~2: Omit seconds.~~

~~4: Omit time marker (e.g. AM/PM).~~

~~8: Always use 24-hour time rather than 12-hour time (but specify T12 to additionally omit the AM/PM indicator).~~

~~0x80000000: Do not obey any overrides the user may have in effect for the system's default time format.~~

~~0x40000000: Use the system ANSI code page for string translation instead of the locale's code page.~~

~~**Note:** Dn and Tn may be repeated to put more than one option into effect, such as this example: FormatTime, OutputVar, 20040228-D2-D4-T1-T8~~

Remarks

~~Letters and numbers that you want to be transcribed literally from Format into OutputVar should be enclosed in single quotes as in this example: 'Date:' MM/dd/yy 'Time:' hh:mm:ss tt~~

~~By contrast, non-alphanumeric characters such as spaces, tabs, linefeeds (\n), slashes, colons, commas, and other punctuation do not need to be enclosed in single quotes. The exception to this is the single-quote character itself: to produce it literally, use four consecutive single quotes ("'"), or just two if the quote is already inside an outer pair of quotes.~~

~~If Format contains date and time elements together, they must not be intermixed. In other words, the string should be dividable into two halves: a time half and a date half. For example, a format string consisting of "hh yyyy mm" would not produce the expected result because it has a date element in between two time elements.~~

~~When Format contains a numeric day of the month (either d or dd) followed by the full month name (MMMM), the genitive form of the month name is used (if the language has a genitive form).~~

~~If Format contains more than 2000 characters, OutputVar will be made blank.~~

Related

~~SetFormat, Transform, built-in date and time variables, FileGetTime~~

Examples

```
FormatTime, TimeString
MsgBox The current time and date (time first) is %TimeString%.

FormatTime, TimeString, R
MsgBox The current time and date (date first) is %TimeString%.

FormatTime, TimeString,, Time
MsgBox The current time is %TimeString%.

FormatTime, TimeString, T12, Time
MsgBox The current 24 hour time is %TimeString%.

FormatTime, TimeString,, LongDate
MsgBox The current date (long format) is %TimeString%.

FormatTime, TimeString, 20050423220133, dddd MMMM d, yyyy hh:mm:ss tt
MsgBox The specified date and time, when formatted, is %TimeString%.

FormatTime, TimeString, 200504, 'Month Name': MMMM 'n' 'Day Name': dddd
MsgBox %TimeString%

FormatTime, YearWeek, 20050101, YWeek
MsgBox January 1st of 2005 is in the following ISO year and week number: %YearWeek%.

FileSelectFile, FileName, 3,, Pick a file
If FileName = , The user didn't pick a file.
    return
FileGetTime, FileTime, %FileName%
FormatTime, FileTime, %FileTime%, Since the last parameter is omitted, the long date and time are retrieved.
MsgBox The selected file was last modified at %FileTime%.
```

IfInString / IfNotInString

Checks if a **variable** contains the specified string.

IfInString, var, SearchString
IfNotInString, var, SearchString

Parameters

var	The name of the variable whose contents will be searched for a match.
SearchString	The string to search for.

Remarks

The built-in variables `%A_Space%` and `%A_Tab%` contain a single space and a single tab character, respectively, which might be useful when searching for these characters alone.

Another command can appear on the same line as this one. In other words, both of these are equivalent:

```
IfInString, MyVar, abc, Gosub, Process1  
IfInString, MyVar, abc  
    Gosub, Process1
```

Related

[IfEqual](#), [if var in/contains MatchList](#), [if var between](#), [if var is type](#), [Blocks](#), [Else](#)

Example

```
Haystack = abcdefghijklmnopqrs  
Needle = abc  
IfInString, Haystack, %Needle%  
{  
    MsgBox, The string was found.  
    return  
}  
else  
    Sleep, 1
```

If var [not] in value1,value2,... [v1.0.18+]

If var [not] contains value1,value2,...

Checks whether a [variable's](#) contents match one of the items in a list.

```
if Var in MatchList  
if Var not in MatchList  
if Var contains MatchList  
if Var not contains MatchList
```

Parameters

Var	The name of the variable whose contents will be checked. For the "in" operator, an exact match with one of the list items is required. For the "contains" operator, a match occurs more easily: whenever <i>Var</i> contains one of the list items as a substring.
MatchList	<p>A comma-separated list of strings, each of which will be compared to the contents of <i>Var</i> for a match. Any spaces or tabs around the delimiting commas are significant, meaning that they are part of the match string. For example, if <i>MatchList</i> is "ABC , XYZ", <i>Var</i> must contain either "ABC " (trailing space) or " XYZ" (leading space) to cause a match.</p> <p>Two consecutive commas results in a single literal comma. For example, the following would produce a single literal comma at the end of string1: "string1,,,string2". Similarly, the following list contains only a single item with a literal comma inside it: "single,,item". To include a blank item in the list, make the first character a comma as in this example: ,string1,string2 (when using the "contains" operator, a blank item will always result in a match since the empty string is found in all strings).</p> <p>Because the items in <i>MatchList</i> are not treated as individual parameters, the list can be contained entirely within a variable. In fact, all or part of it must be contained in a variable if its length exceeds 16383 since that is the maximum length of any script line. For example, <i>MatchList</i> might consist of %List1%,%List2%,%List3% -- where each of the sublists contains a large list of match phrases.</p> <p>Any single item in the list that is longer than 16384 characters will have those extra characters treated as a new list item. Thus, it is usually best to avoid including such items.</p>

Remarks

The comparison is always done alphabetically, not numerically. For example, the string "11" would not match the list item "11.0".

The "contains" operator is the same as using [IfInString/IfNotInString](#) except that multiple search strings are supported (any one of which will cause a match).

"[StringCaseSense](#) On" can be used to make the comparison case sensitive.

Related

[if var between](#), [IfEqual/Greater/Less](#), [IfInString](#), [StringCaseSense](#), [Blocks](#), [Else](#)

Examples

```
if var in exe,bat,com  
    MsgBox The file extension is an executable type.
```

```
if var in 1,2,3,5,7,11 ; Avoid spaces in list.  
    MsgBox %var% is a small prime number.  
  
if var in %MyItemList%  
    MsgBox %var% is in the list.  
  
InputBox, UserInput, Enter YES or NO  
if UserInput not in yes,no  
    MsgBox Your input is not valid.  
  
WinGetTitle, active_title, A  
if active_title contains Address List.txt,Customer List.txt  
    MsgBox One of the desired windows is active.  
if active_title not contains metapad,Notepad  
    MsgBox But the file is not open in either Metapad or Notepad.
```

Sort [v1.0.13+]

Arranges a variable's contents in alphabetical or numerical order.

Sort, VarName [, Options]

Parameters

VarName	The name of the variable whose contents will be sorted.
Options	See list below.

Options

A string of zero or more of the following letters (in any order, with optional spaces in between):

C: Case sensitive sort (ignored if the **N** option is also present).

Dx: Specifies **x** as the delimiter character, which determines where each item in *VarName* begins and ends. If this option is not present, **x** defaults to linefeed (`n). When linefeed is the delimiter, *VarName* will be correctly sorted if its lines end in either LF (`n) or CR+LF (`r`n).

N: Numeric sort: Each item is assumed to be a number rather than a string (for example, if this option is not present, the string 233 is considered to be less than the string 40 due to alphabetical ordering). Both decimal and hexadecimal strings (e.g. 0xF1) are considered to be numeric. Strings that don't start with a number are considered to be zero for the purpose of the sort. Numbers are treated as 64-bit signed values.

Pn: Sorts items based on character position **n** (do not use hexadecimal for **n**). If this option is not present, **n** defaults to 1, which is the position of the first character. The sort compares each string to the others starting at its **n**th character. If **n** is greater than the length of any string, that string is considered to be blank for the purpose of the sort. When used with option **N** (numeric sort), the string's character position is used, which is not necessarily the same as the number's digit position.

R: Sorts in reverse order (alphabetically or numerically depending on the other options).

Random Sorts in random order. This option causes all other options except **D** and **Z** to be ignored. Examples:

Sort, MyVar, Random

Sort, MyVar, Random Z D|

Z: To understand this option, consider a variable that contains RED`nGREEN`nBLUE`n. If the **Z** option is not present, the last linefeed (`n) is considered to be part of the last item, and thus there are only 3 items. But by specifying the **Z** option, the last linefeed (if present) will be considered to delimit a blank item at the end of the list, and thus there are 4 items (the last being blank).

~~****(backslash): Sorts items based on the substring that follows the last backslash in each. If an item has no backslash, the entire item is used as the substring. This option is useful for sorting bare filenames (i.e. excluding their paths). In the example below, the AAA.txt line is sorted above the BBB.txt line because their directories are ignored for the purpose of the sort:~~

E:\BBB\AAA.txt

E:\AAA\BBB.txt

~~Note: Options **N** and **P** are ignored when the backslash option is present.~~

Remarks

This command is typically use to sort a variable that contains a list of lines, with each line ending in a linefeed (`n) character.

~~If *VarName* is clipboard and the clipboard contains files (such as those copied from an open Explorer window), those files will be replaced with a sorted list of their filenames. In other words, after the operation, the clipboard will no longer contain the files themselves.~~

~~If *VarName* is an environment variable, a script variable will be created to take its place. To put the sorted contents back into the environment variable and force the script to access it in the future, use this example:~~

VarSort, MyEnvVar

EnvSet, MyEnvVar, %MyEnvVar%

~~MyEnvVar = ; make the script's variable blank so that MyEnvVar will be retrieved from the environment in the future.~~

The maximum capacity of a variable can be increased via [#MaxMem](#).

If a large variable was sorted and later its contents are no longer needed, you can free its memory by making it blank, e.g. MyVar =

Related

[StringSplit](#), [parsing loop](#), [clipboard](#), [#MaxMem](#)

Example

```
MyVar = 5,3,7,9,1,13,999,-4
Sort MyVar, N D, ; Sort numerically, use comma as delimiter.
MsgBox %MyVar% ; The result is -4,1,3,5,7,9,13,999

; This example makes Win+C a hotkey to copy files from an open
; Explorer window and put their sorted filenames onto the clipboard:
#c:::
Clipboard = ; Must be blank for detection to work.
Send ^c
ClipWait 2
if ErrorLevel <> 0
    return
Sort Clipboard
MsgBox Ready to be pasted:`n%Clipboard%
return
```

#StringCaseSense

Determines whether string comparisons are case sensitive (default = no).

StringCaseSense, On|Off

Parameters

On|Off

On: String comparisons are case sensitive.
Off: String comparisons are not case sensitive.

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

Remarks

The command affects the behavior of IfEqual (and its family), IfInString, StringReplace and StringGetPos.

If this command isn't used in a script, string comparisons are never case sensitive.

The built-in variable **A_StringCaseSense** contains the current setting (On or Off).

Every newly launched hotkey, custom menu item, or timed subroutine starts off fresh with the default setting for this command. That default may be changed by using this command in the auto-execute section (top part) of the script.

-

Related

IfEqual, IfInString, if var between, StringReplace, StringGetPos

-

Example

StringCaseSense, on

StringGetPos

Retrieves the position of the specified substring within a string.

StringGetPos, OutputVar, InputVar, SearchText [, L#|R#]

Parameters

OutputVar	The name of the variable in which to store the retrieved position. Position 0 is the first character.
InputVar	The name of the input variable, whose contents will be searched. Do not enclose the name in percent signs unless you want the <i>contents</i> of the variable to be used as the name.
SearchText	The text to search for. Matching is not case sensitive unless changed by StringCaseSense .
L# R#	This affects which occurrence will be found if <i>SearchText</i> occurs more than once within <i>InputVar</i> . If this parameter is omitted, <i>InputVar</i> will be searched starting from the left for the first match. If this parameter is 1 or the letter R, the search will start looking at the right side of <i>InputVar</i> and will continue leftward until the first match is found. In v1.0.09+, to find a match other than the first, specify the letter L or R followed by the number of the occurrence. For example, to find the fourth occurrence from the right, specify r4. Note: If the number is less than or equal to zero, no match will be found.

ErrorLevel

[ErrorLevel](#) is set to 1 if the specified occurrence of *SearchText* could not be found within *InputVar*, or 0 otherwise.

Remarks

Unlike [StringMid](#), 0 is defined as the position of the first character.

If the specified occurrence of *SearchText* does not exist within *InputVar*, *OutputVar* will be set to -1 and [ErrorLevel](#) will be set to 1.

Use [SplitPath](#) to more easily parse a file path into its directory, filename, and extension.

The built-in variables [%A_Space%](#) and [%A_Tab%](#) contain a single space and a single tab character, respectively, which is useful when searching for these characters alone or strings that begin or end with them.

Related

[RegExGetPos](#), [IfInString](#), [SplitPath](#), [StringLeft](#), [StringRight](#), [StringMid](#), [StringTrimLeft](#), [StringTrimRight](#), [StringLen](#), [StringLower](#), [StringUpper](#), [StringReplace](#), if var is type

Examples

```
Haystack = abcdefghijklmnopqrs
Needle = def
StringGetPos, pos, Haystack, %Needle%
if pos >= 0
    MsgBox, The string was found at position %pos%.
```

; Example #2:

; Divides up the full path name of a file into components.

; Note that it would be much easier to use [StringSplit](#) or a

; [parsing loop](#) to do this, so the below is just for illustration.

```
FileSelectFile, file, , , Pick a filename in a deeply nested folder:
if file <>
{
    StringLen, pos_prev, file
    pos_prev++ ; Adjust to be the position after the last char.
    Loop
    {
        ; Search from the right for the Nth occurrence:
        StringGetPos, pos, file, \, R%a_index%
        if ErrorLevel <> 0
            break
        length = %pos_prev%
        length -= %pos%
```

```
length--  
pos_prev = %pos%  
pos += 2 ; Adjust for use with StringMid.  
StringMid, path_component, file, %pos%, %length%  
MsgBox Path component #%a_index% (from the right) is: `n%path_component%  
}  
}
```

#RegExGetPos

Retrieves the position of the specified regular expression substring within a string.

RegExGetPos, OutputVar, InputVar, SearchText [, L#|R#]

Parameters

OutputVar	The name of the variable in which to store the retrieved position. Position 0 is the first character.
InputVar	The name of the input variable, whose contents will be searched. Do not enclose the name in percent signs unless you want the contents of the variable to be used as the name.
RegExSearchText	The regular expression to search for. This is treated as a regular expression, meaning you can find a complex pattern with this. Matching is not case sensitive unless changed by StringCaseSense.
L# R#	This affects which occurrence will be found if RegExSearchText occurs more than once within InputVar. If this parameter is omitted, InputVar will be searched starting from the left for the first match. If this parameter is 1 or the letter R, the search will start looking at the right side of InputVar and will continue leftward until the first match is found. In v1.0.09+, to find a match other than the first, specify the letter L or R followed by the number of the occurrence. For example, to find the fourth occurrence from the right, specify r4. Note: If the number is less than or equal to zero, no match will be found.

ErrorLevel

ErrorLevel is set to 1 if the specified occurrence of RegExSearchText could not be found within InputVar, or 0 otherwise.

Remarks

Unlike StringMid, 0 is defined as the position of the first character.

If the specified occurrence of RegExSearchText does not exist within InputVar, OutputVar will be set to -1 and ErrorLevel will be set to 1.

Use SplitPath to more easily parse a file path into its directory, filename, and extension.

The built-in variables %A_Space% and %A_Tab% contain a single space and a single tab character, respectively, which is useful when searching for these characters alone or strings that begin or end with them.

Examples

```
Haystack = abc4defghijklmnopqrs
Needle = [0-9]
RegExGetPos, pos, Haystack, %Needle%
if pos >= 0
    MsgBox, A number was found at position %pos%.
```

StringLeft / StringRight

Retrieves a number of characters from the left or right hand side of a string.

StringLeft, OutputVar, InputVar, Count
StringRight, OutputVar, InputVar, Count

Parameters

OutputVar	The name of the variable in which to store the substring extracted from <i>InputVar</i> .
InputVar	The name of the variable whose contents will be extracted from. Do not enclose the name in percent signs unless you want the <i>contents</i> of the variable to be used as the name.
Count	The number of characters to extract. If <i>Count</i> is less than or equal to zero, <i>OutputVar</i> will be made empty (blank). If <i>Count</i> exceeds the length of <i>InputVar</i> , <i>OutputVar</i> will be set equal to the entirety of <i>InputVar</i> .

Remarks

For this and all other commands, *OutputVar* is allowed to be the same variable as an *InputVar*.

Related

[IfInString](#), [StringGetPos](#), [StringMid](#), [StringTrimLeft](#), [StringTrimRight](#), [StringLen](#), [StringLower](#), [StringUpper](#), [StringReplace](#)

Example

String = This is a test.

StringLeft, OutputVar, String, 4 ; Stores the string "This" in OutputVar.

StringRight, OutputVar, InputVar, 5 ; Stores the string "test." in OutputVar.

StringLen

Retrieves the count of how many characters are in a string.

StringLen, OutputVar, InputVar

Parameters

OutputVar	The name of the variable in which to store the length.
InputVar	The name of the variable whose contents will be measured. Do not enclose the name in percent signs unless you want the <i>contents</i> of the variable to be used as the name.

Remarks

None

Related

[IfInString](#), [StringGetPos](#), [StringMid](#), [StringTrimLeft](#), [StringTrimRight](#), [StringLeft](#), [StringRight](#), [StringLower](#), [StringUpper](#), [StringReplace](#)

Example

StringLen, length, InputVar
MsgBox, The length is %length%.

StringLower / StringUpper

Converts a string to lowercase or uppercase.

StringLower, OutputVar, InputVar [, T]
StringUpper, OutputVar, InputVar [, T]

Parameters

OutputVar	The name of the variable in which to store newly converted string.
InputVar	The name of the variable whose contents will be read from. Do not enclose the name in percent signs unless you want the <i>contents</i> of the variable to be used as the name.
T	If this parameter is the letter T, the string will be converted to title case. For example, "GONE with the WIND" would become "Gone With The Wind".

Remarks

To detect whether a character or string is entirely uppercase or lowercase, use "[if var is \[not\] upper/lower](#)".

For this and all other commands, *OutputVar* is allowed to be the same variable as an *InputVar*.

Related

[IfInString](#), [StringGetPos](#), [StringMid](#), [StringTrimLeft](#), [StringTrimRight](#), [StringLeft](#), [StringRight](#), [StringLen](#), [StringReplace](#)

Example

StringUpper, String1, String1 ; i.e. output can be the same as input.
StringLower, String2, String2

StringMid

Retrieves a number of characters from the specified position in a string.

StringMid, OutputVar, InputVar, StartChar, Count **L**

Parameters

OutputVar	The name of the variable in which to store the substring extracted from <i>InputVar</i> .
InputVar	The name of the variable whose contents will be extracted from. Do not enclose the name in percent signs unless you want the <i>contents</i> of the variable to be used as the name.
StartChar	The position of the first character to be extracted. Unlike StringGetPos , 1 is the first character. If <i>StartChar</i> is less than 1, it will be assumed to be 1.
Count	The number of characters to extract. If count is less than or equal to zero, <i>OutputVar</i> will be made empty (blank). If count exceeds the length of <i>InputVar</i> measured from <i>StartChar</i> , <i>OutputVar</i> will be set equal to the entirety of <i>InputVar</i> starting at <i>StartChar</i> .
L	<p>In v1.0.15+, the letter L can be used to extract characters from the left of <i>StartChar</i> rather than from the right. In this example, <i>OutputVar</i> will be set to Red:</p> <p>InputVar = The Red Fox StringMid, OutputVar, InputVar, 7, 3, L</p> <p>If the L option is present and <i>StartChar</i> is less than 1, <i>OutputVar</i> will be made blank. If <i>StartChar</i> is beyond the length of <i>InputVar</i>, only those characters within reach of <i>Count</i> will be extracted. For example, the below will set <i>OutputVar</i> to Fox:</p> <p>InputVar = The Red Fox StringMid, OutputVar, InputVar, 14, 6, L</p>

Remarks

For this and all other commands, *OutputVar* is allowed to be the same variable as an *InputVar*.

Related

[IfInString](#), [StringGetPos](#), [StringLeft](#), [StringRight](#), [StringTrimLeft](#), [StringTrimRight](#), [StringLen](#), [StringLower](#), [StringUpper](#), [StringReplace](#)

Example

```
Source = Hello this is a test.  
StringMid, the_word_this, Source, 7, 4
```

StringReplace

Replaces the specified substring with a new string.

```
StringReplace, OutputVar, InputVar, SearchText [, ReplaceText, ReplaceAll?]
```

Parameters

OutputVar	The name of the variable in which to store the result of the replacement process.
InputVar	The name of the variable whose contents will be read from. Do not enclose the name in percent signs unless you want the <i>contents</i> of the variable to be used as the name.
SearchText	The text to search for. Matching is not case sensitive unless changed by StringCaseSense .
ReplaceText	<i>SearchText</i> will be replaced with this text. If omitted or blank, <i>SearchText</i> will be replaced with blank (empty). In other words, it will be omitted from <i>OutputVar</i> .
ReplaceAll?	If omitted, only the first occurrence of <i>SearchText</i> will be replaced. But if this parameter is 1, A, or All, all occurrences will be replaced.

ErrorLevel

[ErrorLevel](#) is set to 1 if *SearchText* could not be found within *InputVar*, or 0 otherwise.

Remarks

For this and all other commands, *OutputVar* is allowed to be the same variable as an *InputVar*.

The built-in variables [%A_Space%](#) and [%A_Tab%](#) contain a single space and a single tab character, respectively, which might be useful when searching for these characters alone.

Related

Examples

; Remove all CR+LF's from the clipboard contents:
 StringReplace, clipboard, clipboard, `r`n, , all
 ; Replace all spaces with pluses:
 StringReplace, NewStr, OldStr, %A_SPACE%, +, all

RegExReplace

Replaces the specified regular expression substring with a new string.

<i>RegExReplace, OutputVar, InputVar, RegExSearchText [, ReplaceText, ReplaceAll?]</i>
--

Parameters

OutputVar	<i>The name of the variable in which to store the result of the replacement process.</i>
InputVar	<i>The name of the variable whose contents will be read from. Do not enclose the name in percent signs unless you want the contents of the variable to be used as the name.</i>
RegExSearchText	<i>The text to search for. This is treated as a regular expression, meaning you can capture complex pattern with this. Matching is not case sensitive unless changed by StringCaseSense.</i>
ReplaceText	<i>RegExSearchText will be replaced with this text. If omitted or blank, RegExSearchText will be replaced with blank (empty). In other words, it will be omitted from OutputVar.</i>
ReplaceAll?	<i>If omitted, only the first occurrence of RegExSearchText will be replaced. But if this parameter is 1, A, or All, all occurrences will be replaced.</i>

ErrorLevel

ErrorLevel is set to 1 if RegExSearchText could not be found within InputVar, or 0 otherwise.

Remarks

For this and all other commands, OutputVar is allowed to be the same variable as an InputVar.

The built-in variables %A_Space% and %A_Tab% contain a single space and a single tab character, respectively, which might be useful when searching for these characters alone.

Examples

*; Replace all number sections with single underscores:
RegExReplace, NewStr, OldStr, [0-9]+, _, all*

StringSplit

Separates a string into an array of substrings using the specified delimiters.

StringSplit, OutputArray, InputVar [, Delimiters, OmitChars, FutureUse]

Parameters

OutputArray	The name of the array in which to store each substring extracted from <i>InputVar</i> . For example, if MyArray is specified, the command will put the number of substrings produced (0 if none) into MyArray0, the 1st substring into MyArray1, the 2nd into MyArray2, and so on.
InputVar	The name of the variable whose contents will be analyzed. Do not enclose the name in percent signs unless you want the <i>contents</i> of the variable to be used as the name.

Delimiters	<p>If this parameter is blank or omitted, each character of <i>InputVar</i> will be treated as a separate substring.</p> <p>Otherwise, <i>Delimiters</i> contains one or more characters (case sensitive) used to determine where the boundaries between substrings occur in <i>InputVar</i>. Since the delimiter characters are not considered to be part of the substrings themselves, they are never copied into <i>OutputArray</i>. If there is nothing between a pair of delimiters within <i>InputVar</i>, the corresponding array element will be blank.</p> <p>For example: ` (an escaped comma) would divide the string based on every occurrence of a comma. Similarly, %A_Tab%%A_Space% would create a new array element every time a space or tab is encountered in <i>InputVar</i>.</p> <p>To use a string as a delimiter rather than a character, first use StringReplace to replace all occurrences of the string with a single character that is never used literally in the text. Consider this example, which uses the string
 as a delimiter:</p> <pre>StringReplace, NewHTML, HTMLString,
, ``, All ; Replace each
 with an accent. StringSplit, MyArray, NewHTML, `` ; Split the string based on the accent character.</pre>
OmitChars	<p>An optional list of characters (case sensitive) to exclude from the beginning and end of each array element. For example, if <i>OmitChars</i> is %A_Space%%A_Tab%, spaces and tabs will be removed from the beginning and end (but not the middle) of every element.</p> <p>If <i>Delimiters</i> is blank, <i>OmitChars</i> indicates which characters should be excluded from the array.</p>
FutureUse	<p>This parameter should be always left blank (omitted).</p>

Remarks

If the array elements already exist, the command will change the values of only the first N elements, where N is the number of substrings present in *InputVar*. Any elements beyond N that existed beforehand will be unchanged. Therefore, its safest to use the zero element (MyArray0) to determine how many items were actually produced by the command.

Whitespace characters such as spaces and tabs will be preserved unless those characters are themselves delimiters. Tabs and spaces can be trimmed from both ends of any variable by assigning it to itself while [AutoTrim](#) is off (the default). For example: MyArray1 = %MyArray1%

To split a string that is in standard CSV (comma separated value) format, use a [parsing loop](#) since it has built-in CSV handling.

If you do not need the substrings to be permanently stored in memory, consider using a [parsing loop](#) (especially if *InputVar* is very large, in which case a large amount of memory would be saved).

To arrange the fields in a different order prior to splitting them, use the [Sort](#) command.

Related

[Parsing loop](#), [Arrays](#), [Sort](#), [SplitPath](#), [IfInString](#), [StringGetPos](#), [StringMid](#), [StringTrimLeft](#), [StringTrimRight](#), [StringLen](#), [StringLower](#), [StringUpper](#), [StringReplace](#)

Examples

```
TestString = This is a test.
StringSplit, word_array, TestString, %A_Space%, . ; Omits periods.
```

```
MsgBox, The 4th word is %word_array4%.  
  
Colors = red,green,blue  
StringSplit, ColorArray, Colors, `,  
Loop, %ColorArray0%  
{  
    StringTrimLeft, this_color, ColorArray%a_index%, 0  
    MsgBox, Color number %a_index% is %this_color%.  
}
```

StringTrimLeft / StringTrimRight

Removes a number of characters from the left or right hand side of a string.

```
StringTrimLeft, OutputVar, InputVar, Count  
StringTrimRight, OutputVar, InputVar, Count
```

Parameters

OutputVar	The name of the variable in which to store the shortened version of <i>InputVar</i> .
InputVar	The name of the variable whose contents will be read from. Do not enclose the name in percent signs unless you want the <i>contents</i> of the variable to be used as the name.
Count	The number of characters to remove. If <i>Count</i> is less than or equal to zero, <i>OutputVar</i> will be set equal to the entirety of <i>InputVar</i> . If <i>Count</i> exceeds the length of <i>InputVar</i> , <i>OutputVar</i> will be made empty (blank).

Remarks

For this and all other commands, *OutputVar* is allowed to be the same variable as an *InputVar*.

Related

[IfInString](#), [StringGetPos](#), [StringMid](#), [StringLeft](#), [StringRight](#), [StringLen](#), [StringLower](#), [StringUpper](#), [StringReplace](#)

Example

~~String = This is a test.~~

~~StringTrimLeft, OutputVar, String, 5 ; Stores the string "is a test." in OutputVar.~~

~~StringTrimRight, OutputVar, String, 6 ; Stores the string "This is a" in OutputVar.~~

#Control [v1.0.09+]

Makes a variety of changes to a control.

~~Control, Cmd [, Value, Control, WinTitle, WinText, ExcludeTitle, ExcludeText]~~

Parameters

~~Cmd, Value~~

The *Cmd* and *Value* parameters are dependent upon each other and their usage is described below:

~~Check~~: Turns on (checks) a radio button or checkbox.

~~Uncheck~~: Turns off a radio button or checkbox.

~~Enable~~: Enables a control if it was previously disabled.

~~Disable~~: Disables or "grays out" a control.

~~Show~~: Shows a control if it was previously hidden.

~~Hide~~: Hides a control.

~~ShowDropDown~~: Drops a ComboBox so that its choices become visible.

~~HideDropDown~~: Reverses the above.

~~TabLeft [, Count]~~: Moves left by one or more tabs in a SysTabControl32. *Count* is assumed to be 1 if omitted or blank.

~~TabRight [, Count]~~: Moves right by one or more tabs in a SysTabControl32. *Count* is assumed to be 1 if omitted or blank.

~~Add, String~~: Adds *String* as a new entry at the bottom of a ListBox or ComboBox.

~~Delete, N~~: Deletes the *N*th entry from a ListBox or ComboBox. *N* should be 1 for the first entry, 2 for the second, etc.

~~Choose, N~~: Sets the selection in a ListBox or ComboBox to be the *N*th entry. *N* should be 1 for the first entry, 2 for the second, etc.

	ChooseString, String : Sets the selection (choice) in a <i>ListBox</i> or <i>ComboBox</i> to be the entry whose leading part matches <i>String</i> . The search is not case sensitive. For example, if a <i>ListBox/ComboBox</i> contains the item "UNIX Text", specifying the word unix (lowercase) would be enough to select it.
Control	Can be either ClassNN (the classname and instance number of the control) or the name/text of the control, both of which can be determined via Window Spy. When using name/text, the matching behavior is determined by SetTitleMatchMode. If this parameter is blank, the target window's topmost control will be used.
WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the next 3 parameters are omitted, the Last Found Window will be used. If this is the letter A and the next 3 parameters are omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number, specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. This option relies on accessibility support.
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. This option relies on accessibility support.

-

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

-

Remarks

To improve reliability, a delay is done automatically after every use of this command. That delay can be changed via SetControlDelay.

To discover the name of the control that the mouse is currently hovering over, use MouseGetPos.

Window titles and text are always case sensitive. Hidden windows are not detected unless DetectHiddenWindows has been turned on.

-

Related

[SetControlDelay](#), [ControlGet](#), [GuiControl](#), [ControlGetText](#), [ControlSetText](#), [ControlMove](#), [ControlGetPos](#), [ControlClick](#), [ControlFocus](#), [ControlSend](#)

-

Example

Control, HideDropDown, , ComboBox1, Some Window Title

#ControlFocus

Sets input focus to a given control on a window.

ControlFocus [, Control, WinTitle, WinText, ExcludeTitle, ExcludeText]

Parameters

Control	Can be either ClassNN (the classname and instance number of the control) or the name/text of the control, both of which can be determined via Window Spy. When using name/text, the matching behavior is determined by SetTitleMatchMode. If this parameter is blank or omitted, the target window's topmost control will be used.
WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the next 3 parameters are omitted, the Last Found Window will be used. If this is the letter A and the next 3 parameters are omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number, specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

Remarks

To improve reliability, a delay is done automatically after every use of this command. That delay can be changed via SetControlDelay.

To discover the name of the control that the mouse is currently hovering over, use MouseGetPos.

Window titles and text are always case sensitive. Hidden windows are not detected unless DetectHiddenWindows has been turned on.

Related

[SetControlDelay](#), [ControlGetFocus](#), [Control](#), [ControlGet](#), [ControlMove](#), [ControlGetPos](#), [ControlClick](#), [ControlGetText](#), [ControlSetText](#), [ControlSend](#)

Example

ControlFocus, OK, Some Window Title , Set focus to the OK button

#ControlGet [v1.0.09]

Retrieves various types of information about a control.

~~ControlGet, OutputVar, Cmd [, Value, Control, WinTitle, WinText, ExcludeTitle, ExcludeText]~~

Parameters

OutputVar	The name of the variable in which to store the result of <i>Cmd</i> .
Cmd, Value	See list below.
Control	Can be either ClassNN (the classname and instance number of the control) or the name/text of the control, both of which can be determined via Window Spy. When using name/text, the matching behavior is determined by SetTitleMatchMode. If this parameter is blank, the target window's topmost control will be used.
WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the next 3 parameters are omitted, the Last Found Window will be used. If this is the letter A and the next 3 parameters are omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number, specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. This option relies on accessibility support.
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. This option relies on accessibility support.

Cmd, Value

The *Cmd* and *Value* parameters are dependent upon each other and their usage is described below. If a problem is encountered — such as a non-existent window or control — *OutputVar* is made blank and *ErrorLevel* is set to 1.

Checked: Sets *OutputVar* to be 1 if the checkbox or radio button is checked or 0 if not.

Enabled: Sets *OutputVar* to be 1 if *Control* is enabled, or 0 if disabled.

Visible: Sets *OutputVar* to be 1 if *Control* is visible, or 0 if hidden.

Tab: Sets *OutputVar* to the tab number of a SysTabControl32 control. The first tab is 1, the second is 2, etc.

FindString, String: Sets *OutputVar* to the entry number of a ListBox or ComboBox that is an exact match for *String*. The first entry in the control is 1, the second 2, and so on. If no match is found, *OutputVar* is made blank and *ErrorLevel* is set to 1.

Choice: Sets *OutputVar* to be the name of the currently selected entry in a ListBox or ComboBox. To instead retrieve the position of the selected item, follow this example (use only one of the first two lines):

SendMessage, 0x188, 0, 0, ListBox1, WinTitle ; 0x188 is LB_GETCURSEL (for a ListBox).

~~SendMessage, 0x147, 0, 0, ComboBox1, WinTitle ; 0x147 is CB_GETCURSEL (for a DropDownList or ComboBox).~~
~~ChoicePos = %ErrorLevel%~~
~~ChoicePos += 1; Convert from 0-based to 1-based, i.e. so that the first item is known as 1 not 0.~~
~~LineCount: Sets OutputVar to be the number of lines in an Edit control. All Edit controls have at least 1 line, even if the control is empty.~~
~~CurrentLine: Sets OutputVar to be the line number in an Edit control where the caret (insert point) resides. The first line is 1. If there is text selected in the control, OutputVar is set to the line number where the selection begins.~~
~~CurrentCol: Sets OutputVar to be the column number in an Edit control where the caret (insert point) resides. The first column is 1. If there is text selected in the control, OutputVar is set to the column number where the selection begins.~~
~~Line, N: Sets OutputVar to be the text of line N in an Edit control. Line 1 is the first line. Depending on the nature of the control, OutputVar might end in a carriage return (`r) or a carriage return + linefeed (`r`n).~~
~~Selected: Sets OutputVar to be the selected text in an Edit control. If no text is selected, OutputVar will be made blank and ErrorLevel will be set to 0 (i.e. no error). Certain types of controls, such as RichEdit20A, might not produce the correct text in some cases (e.g. Metapad).~~
~~Style [v1.0.21+]: Retrieves an 8-digit hexadecimal number representing the style of the control. See Gui for details about style.~~
~~ExStyle [v1.0.21+]: Retrieves an 8-digit hexadecimal number representing the extended style of the control. See Gui for details about extended style.~~

-

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

-

Remarks

Unlike commands that change a control, ControlGet does not have an automatic delay (SetControlDelay does not affect it).

To discover the name of the control that the mouse is currently hovering over, use MouseGetPos.

Window titles and text are always case sensitive. Hidden windows are not detected unless DetectHiddenWindows has been turned on.

-

Related

[Control](#), [GuiControlGet](#), [ControlMove](#), [ControlSetText](#), [ControlGetText](#), [ControlGetPos](#), [ControlClick](#), [ControlFocus](#), [ControlSend](#)

-

Example

```
ControlGet, OutputVar, Line, 1, Edit1, Some Window Title  
ControlGet, WhichTab, Tab, , SysTabControl321, Some Window Title  
If ErrorLevel <> 0  
    MsgBox There was a problem.
```

#ControlGetFocus

Retrieves which control of the target window has input focus, if any.

```
ControlGetFocus, OutputVar [WinTitle, WinText, ExcludeTitle, ExcludeText]
```

Parameters

OutputVar	The name of the variable in which to store the identifier of the control, which consists of its classname followed by its sequence number within its parent window, e.g. Button12.
WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the next 3 parameters are omitted, the Last Found Window will be used. If this is the letter A and the next 3 parameters are omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number, specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. This option relies on accessibility support.
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. This option relies on accessibility support.

ErrorLevel

ErrorLevel is set to 0 if the control with input focus was successfully retrieved. Otherwise (e.g. the target window doesn't exist or none of its controls have input focus) it will be set to 1.

Remarks

The control retrieved by this command is the one that has keyboard focus, that is, the one that would receive keystrokes if the user were to type any.

The target window must be active to have a focused control. If the window is not active, OutputVar will be made blank.

If ControlFocus is executed repeatedly at a high frequency (i.e. every 500 ms or faster), it will probably disrupt the user's ability to double click. There is no known workaround.

Window titles and text are always case sensitive. Hidden windows are not detected unless DetectHiddenWindows has been turned on.

Related

[ControlFocus](#), [ControlMove](#), [ControlClick](#), [ControlGetText](#), [ControlSetText](#), [ControlSend](#)

Example

```
controlGetFocus, OutputVar, Untitled - Notepad
if ErrorLevel = 0
    MsgBox, Control with focus = %OutputVar%
else
    MsgBox, The target window doesn't exist or none of its controls has input focus.
```

ControlGetPos

Retrieves the position and size of a control.

```
ControlGetPos [, X, Y, Width, Height, Control, WinTitle, WinText, ExcludeTitle, ExcludeText]
```

Parameters

X, Y	The names of the variables in which to store the X and Y coordinates of <i>Control</i> 's upper left corner. These coordinates are relative to the target window and thus are the same as those used by ControlMove . If either X or Y is omitted, the corresponding values will not be stored.
Width/Height	The names of the variables in which to store <i>Control</i> 's width and height. If omitted, the corresponding values will not be stored.
Control	Can be either ClassNN (the classname and instance number of the control) (<i>recommended</i>) or the name/text of the control (<i>can be slow</i>), both of which can be determined via Window Spy. When using name/text, the matching behavior is determined by SetTitleMatchMode . If this parameter is blank, the target window's topmost control will be used.
WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the next 3 parameters are omitted, the Last Found Window will be used. If this is the letter A and the next 3 parameters are omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number, specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON.

ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered.

Remarks

This command relies on accessibility support. [Please read this section](#) to avoid problems.

If no matching window or control is found, the output variables will be made blank.

Unlike commands that change a control, ControlGetPos does not have an automatic delay ([SetControlDelay](#) does not affect it).

To discover the name of the control that the mouse is currently hovering over, use [MouseGetPos](#). To retrieve a list of all controls in a window, use [WinGet](#).

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[ControlMove](#), [WinGetPos](#), [Control](#), [ControlGet](#), [ControlGetText](#), [ControlSetText](#), [ControlClick](#), [ControlFocus](#), [ControlSend](#)

Example

```
; This working example will continuously update and display the  
; name and position of the control currently under the mouse cursor:  
Loop  
{  
    Sleep, 100  
    MouseGetPos, , , WhichWindow, WhichControl  
    ControlGetPos, x, y, w, h, %WhichControl%, ahk_id %WhichWindow%  
    ToolTip, %WhichControl%`nX%X%`tY%Y%`nW%W%`t%H%  
}
```

ControlGetText

Retrieves text from a control.

ControlGetText, OutputVar [, Control, WinTitle, WinText, ExcludeTitle, ExcludeText]

Parameters

OutputVar	The name of the variable in which to store the retrieved text.
Control	Can be either ClassNN (the classname and instance number of the control (<i>recommended</i>)) or the name/text of the control (<i>can be slow</i>), both of which can be determined via Window Spy. When using name/text, the matching behavior is determined by SetTitleMatchMode . If this parameter is blank or omitted, the target window's topmost control will be used.
WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the next 3 parameters are omitted, the Last Found Window will be used. If this is the letter A and the next 3 parameters are omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number , specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON.
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered.

ErrorLevel

[ErrorLevel](#) is set to 1 if there was a problem or 0 otherwise.

Remarks

This command relies on accessibility support. Please read this section to avoid problems.

~~The amount of text retrieved is limited to a variable's maximum capacity (which can be changed via the #MaxMem directive). As a result, this command might use a large amount RAM if the target control (e.g. an editor with a large document open) contains a large quantity of text. However, a variable's memory can be freed after use by assigning it to nothing, i.e. OutputVar =~~

To retrieve a list of all controls in a window, use [WinGet](#).

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[ControlSetText](#), [Control](#), [ControlGet](#), [ControlMove](#), [ControlFocus](#), [ControlClick](#), [ControlSend](#), [#MaxMem](#)

Example

#ControlMove [v1.0.09+]

Moves or resizes a control.

```
ControlMove, Control, X, Y, Width, Height [, WinTitle, WinText, ExcludeTitle, ExcludeText]
```

-
Parameters

Control	Can be either ClassNN (the classname and instance number of the control) or the name/text of the control, both of which can be determined via Window Spy. When using name/text, the matching behavior is determined by SetTitleMatchMode. If this parameter is blank, the target window's topmost control will be used.
X, Y	The X and Y coordinates of the upper left corner of Control's new location. If either coordinate is blank or omitted, Control's position in that dimension will not be changed. The coordinates are relative to Control's parent window; ControlGetPos or Window Spy can be used to determine them.
Width, Height	The new size of Control. If either parameter is blank or omitted, Control's size in that dimension will not be changed.
WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the next 3 parameters are omitted, the Last Found Window will be used. If this is the letter A and the next 3 parameters are omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number, specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. This option relies on accessibility support.
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. This option relies on accessibility support.

-
ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

-
Remarks

To improve reliability, a delay is done automatically after every use of this command. That delay can be changed via SetControlDelay.

Window titles and text are always case sensitive. Hidden windows are not detected unless DetectHiddenWindows has been turned on.

Related

[ControlGetPos](#), [WinMove](#), [SetControlDelay](#), [Control](#), [ControlGet](#), [ControlGetText](#), [ControlSetText](#), [ControlClick](#), [ControlFocus](#), [ControlSend](#)

Example

```
SetTimer, ControlMoveTimer
InputBox, OutputVar, My Input Box
return

ControlMoveTimer:
IfWinNotExist, My Input Box, , return
; Otherwise the above set the "last found" window for us.
SetTimer, ControlMoveTimer, off
WinActivate
ControlMove, OK, 10, , 200 ; Move the OK button to the left and increase its width.
return
```

ControlSetText

Changes the text of a control. *Only works for user-editable fields such as text inputs - cannot be used to change button labels etc.*

ControlSetText, [Control](#), NewText [, WinTitle, WinText, ExcludeTitle, ExcludeText]

Parameters

Control	Can be either ClassNN (the classname and instance number of the control (<i>recommended</i>)) or the name/text of the control (<i>can be slow</i>), both of which can be determined via Window Spy. When using name/text, the matching behavior is determined by SetTitleMatchMode . <i>If this parameter is blank, the target window's topmost control will be used.</i>
NewText	The new text to set into the control.
WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the next 3 parameters are omitted, the Last Found Window will be used. If this is the letter A and the next 3 parameters are omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number , specify ahk_id IDNumber.

WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

ErrorLevel

[ErrorLevel](#) is set to 1 if there was a problem or 0 otherwise.

Remarks

This command relies on accessibility support. Please read this section to avoid problems.

~~To improve reliability, a delay is done automatically after every use of this command. That delay can be changed via [SetControlDelay](#).~~

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[SetControlDelay](#), [ControlGetText](#), [ControlGet](#), [Control](#), [ControlMove](#), [ControlGetPos](#), [ControlClick](#), [ControlFocus](#), [ControlSend](#)

Example

ControlSetText, Edit1, New Text Here, Untitled -

#PostMessage / SendMessage [v1.0.08+]

~~Sends a message to a window or control (SendMessage additionally waits for acknowledgement).~~

~~PostMessage, Msg [, wParam, lParam, Control, WinTitle, WinText, ExcludeTitle, ExcludeText]~~
~~SendMessage, Msg [, wParam, lParam, Control, WinTitle, WinText, ExcludeTitle, ExcludeText]~~

#SetControlDelay

Sets the delay that will occur after each Control modifying command.

SetControlDelay, Delay

Parameters

Delay

Time in milliseconds. Use -1 for no delay at all and 0 for the smallest possible delay.

Remarks

A short delay (sleep) is automatically and invisibly done after every Control command that changes a control, namely Control, ControlMove, ControlClick, ControlFocus, and ControlSetText (ControlSend uses SetKeyDelay). This is done to improve the reliability of scripts because a control sometimes needs a period of "rest" after being changed by one of these commands. The rest period allows it to update itself and respond to the next command that the script may attempt to send to it.

Although a delay of -1 (no delay at all) is allowed, it is recommended that at least 0 be used, to increase confidence that the script will run correctly even when the CPU is under load.

A delay of 0 internally executes a Sleep(0), which yields the remainder of the script's timeslice to any other process that may need it. If there is none, Sleep(0) will not sleep at all.

If the CPU is slow or under load, or if window animation is enabled, higher delay values may be needed.

If unset, the default delay is 20.

The built-in variable **A_ControlDelay** contains the current setting.

Every newly launched hotkey, custom menu item, or timed subroutine starts off fresh with the default setting for this command. That default may be changed by using this command in the auto-execute section (top part) of the script.

Related

Control, ControlMove, ControlClick, ControlFocus, ControlSetText, SetWinDelay, SetKeyDelay, SetMouseDelay, SetBatchLines

Example

```
SetControlDelay, 0
```

#WinMenuItemSelect

Invokes a menu item from the menu bar of a window.

```
WinMenuItemSelect, WinTitle, WinText, Menu [, SubMenu1, SubMenu2, SubMenu3, SubMenu4, SubMenu5, SubMenu6, ExcludeTitle, ExcludeText]
```

Parameters

WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the other 3 window parameters are blank or omitted, the Last Found Window will be used. If this is the letter A and the other 3 window parameters are blank or omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number, specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
Menu	The name of the top-level menu, e.g. File, Edit, View. It can also be the position of the desired menu item by using 1& to represent the first menu, 2& the second, and so on.
SubMenu1	The name of the menu item to select or its position (see above).
SubMenu2	If SubMenu1 itself contains a menu, this is the name of the menu item inside, or its position.
SubMenu3	Same as above.
SubMenu4	Same as above.
SubMenu5	Same as above.
SubMenu6	Same as above.
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

ErrorLevel

ErrorLevel is set to 1 if there was a problem or 0 otherwise.

Remarks

For this command to work, the target window need not be active. However, some windows might need to be in a non-minimized state.

This command **will not work** with applications that use non-standard menu bars. Examples include Microsoft Outlook and Outlook Express, which use disguised toolbars for their menu bars. In these cases, consider using ControlSend or PostMessage, which should be able to interact with some of these non-standard menu bars.

The menu name parameters are not case-sensitive (i.e. File->Save is the same as file->save) and the use of ampersand (&) to indicate the underlined letter in a menu item is not necessary (i.e. &File is the same as File).

The menu name parameters can also specify positions. This method exists to support menus that don't contain text (perhaps because they contain pictures of text rather than actual text). Position 1& is the first menu item (e.g. the File menu), position 2& is the second menu item (e.g. the Edit menu), and so on. Menu separator lines count as menu items for the purpose of determining the position of a menu item.

Window titles and text are always case-sensitive. Hidden windows are not detected unless DetectHiddenWindows has been turned on.

Related

[ControlSend](#), [PostMessage](#)

Example

; This will select File->Open in Notepad:

WinMenuItem, Untitled - Notepad, , File, Open

; Same as above except it's done by position vs. name:

WinMenuItem, Untitled - Notepad, , 1&, 2&

#GroupActivate

Activates the next window in a window group that was defined with GroupAdd.

GroupActivate, **GroupName** [, R]

Parameters

GroupName	The name of the group to activate, as originally defined by GroupAdd.
R	This determines whether the oldest or the newest window is activated whenever no members of the group are currently active. If omitted, the oldest window is always activated. If it's the letter R, the newest window (the one most recently active) is activated, but only if no members of the group are active when the command is given. "R" is useful in cases where you temporarily switch to working on an unrelated task. When you return to the group via GroupActivate, GroupDeactivate, or GroupClose, the window you were most recently working with is activated rather than the oldest window.

-

Remarks

This command causes the first window that matches any of the group's window specifications to be activated. Using it a second time will activate the next window in the series and so on. Normally, it is assigned to a hotkey so that this window traversal behavior is automated by pressing that key.

When a window is activated immediately after another window was activated, task bar buttons may start flashing on some systems (depending on OS and settings). To prevent this, use #WinActivateForce.

See GroupAdd for more details about window groups.

-

Related

GroupAdd, GroupDeactivate, GroupClose, #WinActivateForce

-

Example

GroupActivate, MyGroup, R

-

#GroupAdd

Adds a window specification to a window group, creating the group if necessary.

GroupAdd, GroupName, WinTitle [, WinText, Label, ExcludeTitle, ExcludeText]

-

Parameters

GroupName	The name of the group to which to add this window specification. If the group doesn't exist, it will be created.
WinTitle	The title or partial title of the target window. It can be blank. Note: Although SetTitleMatchMode and DetectHiddenWindows do not directly affect the behavior of this command, they do affect the behavior of commands such as GroupActivate and GroupClose. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). The use of a window's unique ID number is <u>not</u> currently supported.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON at the time that GroupActivate, GroupDeactivate, and GroupClose are used.
label	The label of a subroutine to run if no windows matching this specification exist when the GroupActivate command is used. The label is jumped to as though a Gosub had been used. Omit or leave blank for none.
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

Remarks

A window group is typically used to bind together a collection of related windows. Once defined, the GroupActivate command is typically used to traverse the group (visit each member window, one at a time).

Groups are useful for tasks that involve many related windows, or an application that owns many subwindows. For example, if you frequently work with many instances (windows) of a graphics program or text editor, you can use a window group to rapidly visit each instance of that program without having to use alt-tab or task bar buttons to locate them.

Since the entries in each group need to be added only once, this command is typically used in the auto-execute section of the script (the part at the top that occurs prior to the first hotkey label). Attempts to add duplicate entries to a group are ignored.

To include all windows in a group (except the special Program Manager window), use this example:

```
GroupAdd, All, , , , Program Manager
```

Related

[GroupActivate](#), [GroupDeactivate](#), [GroupClose](#)

Examples

; In the autoexecute section at the top of the script:

```
GroupAdd, MSIE, ahk_class IFrame ; Add only Internet Explorer windows to this group.
```

```
return ; End of autoexec section.
```

; Assign a hotkey to activate this group, which traverses

; through all open MSIE windows, one at a time (i.e. each

; press of the hotkey).

```
Numpad1::GroupActivate, MSIE, r
```

```
; Here's a more complex group for MS Outlook 2002:  
; In the autoexecute section at the top of the script:  
SetTitleMatchMode,2  
GroupAdd, mail, Message - Microsoft Word ; This is for mails currently being composed  
GroupAdd, mail, - Message ( ; This is for already opened items  
; Need extra text to avoid activation of a phantom window:  
GroupAdd, mail, Advanced Find, Search for the word(s)  
GroupAdd, mail, , Recurrence:  
GroupAdd, mail, Reminder  
GroupAdd, mail, - Microsoft Outlook  
return ; End of autoexecute section.  
  
NumPad5::GroupActivate, mail ; Assign a hotkey to traverse through the group.
```

#GroupClose

Closes the active window if it was just activated by GroupActivate or GroupDeactivate. It then activates the next window in the series. It can also close all windows in a group.

GroupClose, GroupName [, A|R]

Parameters

GroupName	The name of the group as originally defined by GroupAdd.
A R	If it's the letter A, all members of the group will be closed. Otherwise: If the command closes the active window, it will then activate the next window in the series. This parameter determines whether the oldest or the newest window is activated. If omitted, the oldest window is always activated. If it's the letter R, the newest window (the one most recently active) is activated, but only if no members of the group are active when the command is given. "R" is useful in cases where you temporarily switch to working on an unrelated task. When you return to the group via GroupActivate, GroupDeactivate, or GroupClose, the window you were most recently working with is activated rather than the oldest window.

Remarks

When the A|R parameter is not "A", the behavior of this command is determined by whether the previous action on *GroupName* was GroupActivate or GroupDeactivate. If it was GroupDeactivate, this command will close the active window only if it is **not a member of the group** (otherwise it will do nothing). If it was GroupActivate or nothing, this command will close the active window only if it **is a member of the group** (otherwise it will do nothing). This behavior allows GroupClose to be assigned to a hotkey as a companion to *GroupName*'s GroupActivate or GroupDeactivate hotkey.

See GroupAdd for more details about window groups.

Related

[GroupAdd](#), [GroupActivate](#), [GroupDeactivate](#)

Example

`GroupClose, MyGroup, R`

#GroupDeactivate

Similar to GroupActivate except activates the next window **not** in the group:

`GroupDeactivate, GroupName [, R]`

Parameters

GroupName	The name of the target group, as originally defined by GroupAdd.
R	This determines whether the oldest or the newest non member window is activated whenever a member of the group is currently active. If omitted, the oldest non member window is always activated. If it's the letter R, the newest non member window (the one most recently active) is activated, but only if a member of the group is active when the command is given. "R" is useful in cases where you temporarily switch to working on an unrelated task. When you return to the group via GroupActivate, GroupDeactivate, or GroupClose, the window you were most recently working with is activated rather than the oldest window.

Remarks

~~GroupDeactivate causes the first window that does not match any of the group's window specifications to be activated. Using GroupDeactivate a second time will activate the next window in the series and so on. Normally, GroupDeactivate is assigned to a hotkey so that this window traversal behavior is automated by pressing that key.~~

~~This command is useful in cases where you have a collection of favorite windows that are almost always running. By adding these windows to a group, you can use GroupDeactivate to visit each window that isn't one of your favorites and decide whether to close it. This allows you to clean up your desktop much more quickly than doing it manually.~~

~~See GroupAdd for more details about window groups.~~

-

Related

~~GroupAdd, GroupActivate, GroupClose~~

-

Example

~~GroupDeactivate, MyFavoriteWindows ; Visit non-favorite windows to clean up desktop.~~

-

##WinActivateForce

~~Skips the gentle method of activating a window and goes straight to the forceful method.~~

```
#WinActivateForce
```

-

Parameters

~~None~~

-

Remarks

~~Specifying this anywhere in a script will cause commands that activate a window such as WinActivate, WinActivateBottom, and GroupActivate to skip the "gentle" method of activating a window and go straight to the more forceful methods.~~

~~Although this directive will usually not change how quickly or reliably a window is activated, it might prevent task bar buttons from flashing when different windows are activated quickly one after the other.~~

~~Windows 95 and NT will probably never need this setting since they are more permissive about allowing windows to be activated.~~

-

Related

[WinActivate](#), [WinActivateBottom](#), [GroupActivate](#)

-

Example

`#WinActivateForce`

#DetectHiddenText

Determines whether invisible text in a window is "seen" for the purpose of finding the window. This affects commands such as [IfWinExist](#) and [WinActivate](#).

DetectHiddenText, On|Off

-

Parameters

On Off	On: This is the default. Hidden text will be detected.
	Off

-

Remarks

"Hidden text" is a term that refers to those controls of a window that are not visible. Their text is thus considered "hidden". Turning off **DetectHiddenText** can be useful in cases where you want to detect the difference between the different panes of a multi-pane window or multi-tabbed dialog. Use Window Spy to determine which text of the currently active window is hidden. All commands that accept a **WinText** parameter are affected by this setting, including [WinActivate](#), [IfWinActive](#), [WinWait](#), and [IfWinExist](#).

The built-in variable **A_DetectHiddenText** contains the current setting (On or Off).

Every newly launched hotkey, custom menu item, or timed subroutine starts off fresh with the default setting for this command. That default may be changed by using this command in the auto execute section (top part) of the script.

-

Related

DetectHiddenWindows

-

Example

`DetectHiddenText, off`

-

DetectHiddenWindows

Determines whether invisible windows are "seen" by the script.

`DetectHiddenWindows, On|Off`

Parameters

On|Off

On: Hidden windows are detected.

Off: This is the default. Hidden windows are not detected, except by the [WinShow](#) command.

Remarks

Turning on `DetectHiddenWindows` can make scripting harder in some cases since some hidden system windows might accidentally match the title or text of another window you're trying to work with. So most scripts should leave this setting turn off. However, turning it on may be useful if you wish to work with hidden windows directly without first using [WinShow](#) to unhide them.

All windowing commands except [WinShow](#) are affected by this setting, including [WinActivate](#), [IfWinActive](#), [WinWait](#), [IfWinExist](#). In other words, [WinShow](#) will always unhide a hidden window even if hidden windows are not being detected.

The built-in variable **A_DetectHiddenWindows** contains the current setting (On or Off).

Every newly launched [hotkey](#), [custom menu item](#), or [timed](#) subroutine starts off fresh with the default setting for this command. That default may be changed by using this command in the auto-execute section (top part) of the script.

Related

Example

DetectHiddenWindows, on

IfWinActive / IfWinNotActive

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

```
IfWinActive [, WinTitle, WinText, ExcludeTitle, ExcludeText]  
IfWinNotActive [, WinTitle, WinText, ExcludeTitle, ExcludeText]
```

Parameters

WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the other 3 parameters are omitted, the Last Found Window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number , specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

Remarks

If all parameters are omitted, the [Last Found Window](#) will be used.

If either of these commands determines that the active window is a qualified match, the [Last Found Window](#) will be updated to be the active window. In other words, if *IfWinActive* evaluates to true or *IfWinNotActive* evaluates to false, the [Last Found Window](#) will be updated.

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[SetTitleMatchMode](#), [DetectHiddenWindows](#), [Last Found Window](#), [IfWinExist](#), [WinActivate](#), [WinWaitActive](#), [WinWait](#), [WinWaitClose](#)

Example

```
IfWinActive, Untitled - Notepad
{
    WinMaximize, A ; "A" indicates the active window.
    Send, Some text.{Enter}
    return
}
```

IfWinExist / IfWinNotExist

Checks to see if a specified window exists.

```
IfWinExist [, WinTitle, WinText, ExcludeTitle, ExcludeText]
IfWinNotExist [, WinTitle, WinText, ExcludeTitle, ExcludeText]
```

Parameters

WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the other 3 parameters are all omitted, the Last Found Window will be used (i.e. that specific window is checked to see whether it still exists). To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number , specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

Remarks

If all parameters are omitted, the [Last Found Window](#) will be checked to see if it still exists (or doesn't exist in the case of IfWinNotExist).

If either of these commands determines that a qualified window exists, the [Last Found Window](#) will be updated to be that window. In other words, if [IfWinExist](#) evaluates to true or [IfWinNotExist](#) evaluates to false, the [Last Found Window](#) will be updated.

Related

[SetTitleMatchMode](#), [DetectHiddenWindows](#), [Last Found Window](#), [Process](#), [IfWinActive](#), [WinActivate](#), [WinWaitActive](#), [WinWait](#), [WinWaitClose](#)

Examples

```
IfWinExist, Untitled - Notepad
{
    WinActivate ; Automatically uses the window found above.
    WinMaximize ; same
    Send, Some text.{Enter}
    return
}

IfWinNotExist, Calculator
    return
else
{
    WinActivate ; The above "IfWinNotExist" also set the "last found" window for us.
    WinMove, 40, 40 ; Move it to a new position.
    return
}
```

#SetTitleMatchMode

Determines how window titles are searched for by various commands.

[SetTitleMatchMode](#), [MatchMode](#)
[SetTitleMatchMode](#), [Fast|Slow](#)

-
Parameters

MatchMode	<p>One of the following numbers:</p> <p>1: A window's title must start with the specified <i>WinTitle</i> to be a match. 2: A window's title can contain <i>WinTitle</i> anywhere inside it to be a match. 3 [v1.0.11]: A window's title must exactly match <i>WinTitle</i> to be a match.</p> <p>The above mode also affects <i>ExcludeTitle</i> in the same way. For example, mode 3 requires that a window's title exactly match <i>ExcludeTitle</i> for that window to be excluded.</p> <p>Note that <i>WinTitle</i>, <i>WinText</i>, <i>ExcludeTitle</i> and <i>ExcludeText</i> are always case sensitive.</p>
Fast Slow	<p>Fast: This is the normal behavior. Performance may be substantially better than <i>Slow</i>, but certain <i>WinText</i> elements for some types of windows may not be "seen" by the various window commands.</p> <p>Slow: Can be much slower, but all possible <i>WinText</i> is retrieved from every window as a windowing command searches through them for a match.</p>

Remarks

This command affects the behavior of all windowing commands, e.g. IfWinExist and WinActivate.

If unspecified, TitleMatchMode defaults to 1 and *fast*.

Generally, the *slow* mode should only be used if the target window cannot be uniquely identified by its title and *fast* mode text. This is because the slow mode can be extremely slow if there are any application windows that are busy or "not responding". In addition, such sluggish or unresponsive windows might cause keyboard and mouse lag if the keyboard or mouse hook is installed while the *slow* mode is in effect.

The customized version of Window Spy distributed with AutoHotkey reports *slow* text in a separate section so that its easy to determine whether the *slow* mode is needed.

If you wish to change both attributes, run the command twice as in this example:

```
SetTitleMatchMode,2  
SetTitleMatchMode,slow
```

The built-in variables **A_TitleMatchMode** and **A_TitleMatchModeSpeed** contain the current settings.

Title and text matching are always case sensitive regardless of this setting.

Every newly launched hotkey, custom menu item, or timed subroutine starts off fresh with the default setting for this command. That default may be changed by using this command in the auto-execute section (top part) of the script.

Related

[SetWinDelay](#), [IfWinExist](#), [WinActivate](#)

Example

~~SetTitleMatchMode, 2~~
~~SetTitleMatchMode, slow~~

#SetWinDelay

~~Sets the delay that will occur after each windowing command.~~

SetWinDelay, Delay

-
Parameters

Delay

~~Time in milliseconds. Use -1 for no delay at all and 0 for the smallest possible delay.~~

-
Remarks

~~A short delay (sleep) is automatically and invisibly done after every windowing command (e.g. WinActivate, IfWinExist, WinWait, etc.). This is done to improve the reliability of scripts because a window sometimes needs a period of "rest" after being created, activated, minimized, etc. so that it has a chance to update itself and respond to the next command that the script may attempt to send to it.~~

~~Although a delay of -1 (no delay at all) is allowed, it is recommended that at least 0 be used, to increase confidence that the script will run correctly even when the CPU is under load.~~

~~A delay of 0 internally executes a Sleep(0), which yields the remainder of the script's timeslice to any other process that may need it. If there is none, Sleep(0) will not sleep at all.~~

~~If the CPU is slow or under load, or if window animation is enabled, higher delay values may be needed.~~

~~If unset, the default delay is 250.~~

~~The built-in variable **A_WinDelay** contains the current setting.~~

~~Every newly launched hotkey, custom menu item, or timed subroutine starts off fresh with the default setting for this command. That default may be changed by using this command in the auto execute section (top part) of the script.~~

-
Related

~~SetControlDelay, SetKeyDelay, SetMouseDelay, SetBatchLines~~

-

Example

```
SetWinDelay, 10
```

-

#StatusBarGetText

Retrieves the text from a standard status bar control.

```
StatusBargetText, OutputVar [, Part#, WinTitle, WinText, ExcludeTitle, ExcludeText]
```

-

Parameters

OutputVar	The name of the variable in which to store the retrieved text.
Part#	Which part number of the bar to retrieve. Default 1, which is usually the part that contains the text of interest.
WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the other 3 window parameters are blank or omitted, the Last Found Window will be used. If this is the letter A and the other 3 window parameters are blank or omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number, specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

-

Remarks

This function attempts to read the first standard status bar on a window (Microsoft common control: msctls_statusbar32). Some programs use their own status bars or special versions of the MS common control—the function cannot read these.

Rather than using this command in a loop, it is usually more efficient to use StatusBarWait, which contains optimizations that avoid the overhead of repeated calls to StatusBarGetText.

Window titles and text are always case sensitive. Hidden windows are not detected unless DetectHiddenWindows has been turned on.

-

Related

[StatusBarWait](#), [WinGetTitle](#), [WinGetText](#), [ControlGetText](#)

Example

```
StatusBarGetText, RetrievedText, 1, Search Results  
IfInString, RetrievedText, found, MsgBox, Search results have been found.
```

#StatusBarWait

Waits until a window's status bar contains the specified string.

```
StatusBarWait [, BarText, Seconds, Part#, WinTitle, WinText, Interval, ExcludeTitle, ExcludeText]
```

Parameters

BarText	The text or partial text for the which the command will wait to appear. Default is blank (empty), which means to wait for the status bar to become blank. The text is case sensitive and the matching behavior is determined by SetTitleMatchMode, similar to WinTitle below.
Seconds	The number of seconds (can contain a decimal point) to wait before timing out, in which case ErrorLevel will be set to 1. Default is blank, which means wait indefinitely. Specifying 0 is the same as specifying 0.5.
Part#	Which part number of the bar to retrieve. Default 1, which is usually the part that contains the text of interest.
WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the other 3 window parameters are blank or omitted, the Last Found Window will be used. If this is the letter A and the other 3 window parameters are blank or omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number, specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. This option relies on accessibility support.
Interval	How often the status bar should be checked while the command is waiting (in milliseconds). Default is 50.
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. This option relies on accessibility support.

ErrorLevel

ErrorLevel is set to 1 if the command times out before a match could be found in the status bar. It is set to 2 if the status bar could not be accessed. It is set to 0 if a match is found.

Remarks

This function attempts to read the first standard status bar on a window (Microsoft common control: msctls_statusbar32). Some programs use their own status bars or special versions of the MS common control - the function cannot read these.

Rather than using StatusBarGetText in a loop, it is usually more efficient to use this command, which contains optimizations that avoid the overhead that repeated calls to StatusBarGetText would incur.

While the command is in a waiting state, new threads can be launched via hotkey, custom menu item, or timer.

Window titles and text are always case sensitive. Hidden windows are not detected unless DetectHiddenWindows has been turned on.

Related

[StatusBarGetText](#), [WinGetTitle](#), [WinGetText](#), [ControlGetText](#)

Example

```
#IfWinExist, Search Results, Sets the Last Found window to simplify the below.  
+  
    winActivate  
    Send, (tab 2)!o*.txt{enter}  
    Sleep, 400, Give the status bar time to change to "Searching".  
    StatusBarWait, found, 30  
    if ErrorLevel = 0  
        MsgBox, The search successfully completed.  
    else  
        MsgBox, The command timed out or there was a problem.  
+
```

WinActivate

Activates (brings to the foreground) a window.

WinActivate [, WinTitle, WinText, ExcludeTitle, ExcludeText]

Parameters

WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If all parameters are omitted, the Last Found Window will be activated. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number , specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

Remarks

~~Six attempts will be made to activate the target window over the course of 60ms. Thus, it is usually unnecessary to follow it with the WinWaitActive command.~~

If a matching window is already active, that window will be kept active rather than activating any other matching window beneath it. In general, if more than one window matches, the uppermost (most recently used) will be activated. You can activate the bottommost (least recently used) via [WinActivateBottom](#).

When a window is activated immediately after another window was activated, task bar buttons might start flashing on some systems (depending on OS and settings). To prevent this, use [#WinActivateForce](#).

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[WinActivateBottom](#), [#WinActivateForce](#), [SetTitleMatchMode](#), [DetectHiddenWindows](#), [Last Found Window](#), [IfWinExist](#), [IfWinActive](#), [WinWaitActive](#), [WinWait](#), [WinWaitClose](#), [WinClose](#), [GroupActivate](#), [WinSet](#)

Example

```
IfWinExist, Untitled - Notepad  
    WinActivate ; use the window found above  
else  
    WinActivate, Calculator
```

#WinActivateBottom

~~Same as WinActivate except it activates the least recently used (bottommost) matching window rather than the newest.~~

~~WinActivateBottom [, WinTitle, WinText, ExcludeTitle, ExcludeText]~~

-

Parameters

WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number, specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

-

Remarks

If there is only one matching window, WinActivateBottom behaves identically to WinActivate.

Window groups are more advanced than this command, so consider using them for more features and flexibility.

Six attempts will be made to activate the target window over the course of 60ms. Thus, it is usually unnecessary to follow it with the WinWaitActive command.

Unlike WinActivate, the Last Found Window cannot be used because it might not be the bottommost window. Therefore, at least one of the parameters must be non-blank.

When a window is activated immediately after another window was activated, task bar buttons may start flashing on some systems (depending on OS and settings). To prevent this, use #WinActivateForce.

Window titles and text are always case sensitive. Hidden windows are not detected unless DetectHiddenWindows has been turned on.

-

Related

[WinActivate](#), [#WinActivateForce](#), [SetTitleMatchMode](#), [DetectHiddenWindows](#), [IfWinExist](#), [IfWinActive](#), [WinWaitActive](#), [WinWait](#), [WinWaitClose](#), [GroupActivate](#)

-

Example

~~; This hotkey allows you to visit all open browser windows in order~~

~~; from oldest to newest:~~

~~#::~~

~~SetTitleMatchMode, 2~~

```
WinActivateBottom, - Microsoft Internet Explorer  
return
```

WinClose

Closes a window.

```
WinClose [, WinTitle, WinText, SecondsToWait, ExcludeTitle, ExcludeText]
```

Parameters

WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the other 3 window parameters are blank or omitted, the Last Found Window will be used. If this is the letter A and the other 3 window parameters are blank or omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number , specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON.
SecondsToWait	If omitted or blank, the command will not wait at all. If 0, it will wait 500ms. Otherwise, it will wait the indicated number of seconds (can contain a decimal point) for the window to close. If the window does not close within that period, the script will continue. ErrorLevel is not set by this command, so use IfWinExist or WinWaitClose if you need to determine for certain that a window is closed. While the command is in a waiting state, new threads can be launched via hotkey, custom menu item, or timer.
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

Remarks

This command sends a close message to a window. The result depends on the window (it may ask to save data, etc.)

If a matching window is active, that window will be closed in preference to any other matching window beneath it. In general, if more than one window matches, the uppermost (most recently used) will be closed.

If a window does not close via WinClose, you can force it to close with [WinKill](#).

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[WinKill](#), [WinWaitClose](#), [Process](#), [WinActivate](#), [SetTitleMatchMode](#), [DetectHiddenWindows](#), [Last Found Window](#), [IfWinExist](#), [IfWinActive](#), [WinWaitActive](#), [WinWait](#), [GroupActivate](#)

Example

```
IfWinExist, Untitled - Notepad  
    WinClose ; use the window found above  
else  
    WinClose, Calculator
```

#WinGet [v1.0.12+]

Retrieves a window's unique ID, process ID/name, or a list of its controls. It can also find all windows of a certain class, title, or text.

```
WinGet, OutputVar [, Cmd, WinTitle, WinText, ExcludeTitle, ExcludeText]
```

Parameters

OutputVar	The name of the variable in which to store the result of <i>Cmd</i> .
Cmd	See list below.
WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the other 3 parameters are omitted, the Last Found Window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy).
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support</i> .
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support</i> .

Cmd is the operation to perform, which if blank defaults to *ID*. It can be one of the following words:

ID: Retrieves the unique ID number for the first window matching the specified *WinTitle*, *WinText*, *ExcludeTitle*, and *ExcludeText*. If there is none, *OutputVar* is made blank.

IDLast: Same as above except it retrieves the ID of the last/bottommost window if there is more than one match. If there is only one match, it performs identically to *ID*. This concept is similar to that used by *WinActivateBottom*.

PID: Retrieves the Process ID (PID) for the first window matching the specified *WinTitle*, *WinText*, *ExcludeTitle*, and *ExcludeText*.

ProcessName: Retrieves the name of the process (e.g. notepad.exe) that owns the first matching window. If there are no matching windows, *OutputVar* is made blank.

Count: Retrieves the number of matching windows that exist (0 if none).

List: Retrieves the unique ID numbers for all matching windows (to retrieve all windows on the entire system, leave *WinTitle* and *WinText* blank but specify Program Manager or a non-existent title for *ExcludeTitle*). Each ID number is stored in an array element whose name begins with *OutputVar*'s own name, while *OutputVar* itself is set to the number of retrieved items (0 if none). For example, if *OutputVar* is MyArray and two matching windows are discovered, MyArray1 will be set to the ID of the first window, MyArray2 will be set to the ID of the second window, and MyArray itself will be set to the number 2.

MinMax [v1.0.22+]: Retrieves the minimized/maximized state for the first matching window. *OutputVar* is made blank if no matching window exists; otherwise, it is set to one of the following numbers:

- 1: The window is minimized (WinRestore can unminimize it).
- +1: The window is maximized (WinRestore can unmaximize it).
- 0: The window is neither minimized nor maximized.

ControlList Retrieves the control names for all *interactive* controls in the first window matching the specified *WinTitle*, *WinText*, *ExcludeTitle*, and *ExcludeText*. If there are no controls or the target window does not exist, *OutputVar* is made blank. Otherwise, each control name consists of its class name followed immediately by its sequence number, as shown by Window Spy.

Each item except the last is terminated by a linefeed (`n). To examine the individual control names one by one, use a [parsing loop](#) as shown in the examples section below.

Controls are sorted according to their Z-order, which is usually the same order as TAB key navigation if the window supports tabbing.

The control currently under the mouse cursor can be retrieved with [MouseGetPos](#).

Style or **ExStyle** [v1.0.23+]: Retrieves an 8 digit hexadecimal number representing style or extended style (respectively) of the first matching window. If there are no matching windows, *OutputVar* is made blank. The following example determines whether a window has the WS_DISABLED style:

```
WinGet, Style, My Window Title  
Transform, Result, BitAnd, %Style%, 0x8000000, 0x8000000 is WS_DISABLED.  
if Result <> 0  
... the window is disabled, so perform appropriate action.
```

The next example determines whether a window has the WS_EX_TOPMOST style (always-on-top):

```
WinGet, ExStyle, My Window Title  
Transform, Result, BitAnd, %ExStyle%, 0x8, 0x8 is WS_EX_TOPMOST.  
if Result <> 0  
... the window is always on top, so perform appropriate action.
```

See [Gui](#) for more details about styles and extended styles.

-

Remarks

A window's ID number is valid only during its lifetime. In other words, if an application restarts, all of its windows will get new ID numbers.

ID numbers retrieved by this command are numeric (the prefix "ahk_id" is not included) and are stored in hexadecimal format regardless of the setting of [SetFormat](#).

The ID of the window under the mouse cursor can be retrieved with [MouseGetPos](#).

Although ID numbers are currently 32-bit unsigned values, they may become 64-bit in future versions. Therefore, it is unsafe to perform numerical operations such as addition on these values because such operations require that their input strings be parsable as signed rather than unsigned values.

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[WinGetClass](#), [Process](#), [WinGetTitle](#), [MouseGetPos](#), [Control](#), [ControlFocus](#), [GroupAdd](#)

Examples

; Example #1: Maximize the active window and report its unique ID:

```
WinGet, active_id, ID, A  
WinMaximize, ahk_id %active_id%  
MsgBox, The active window's ID is "%active_id%".
```

; Example #2: This will visit all windows on the entire system and display info about each of them:

```
WinGet, id, list,,, Program Manager  
Loop, %id%  
{  
    StringTrimRight, this_id, id%a_index%, 0  
    WinActivate, ahk_id %this_id%  
    WinGetClass, this_class, ahk_id %this_id%  
    WinGetTitle, this_title, ahk_id %this_id%  
    MsgBox, 4, , Visiting All Windows`n%a_index% of %id%`nahk_id %this_id%`nahk_class %this_class%`n%this_title%`n`nContinue?  
    IfMsgBox, NO, break  
}
```

Example #3: Extract the individual control names from a ControlList:

```
WinGet, ActiveControlList, ControlList, A  
Loop, Parse, ActiveControlList, `n  
{  
    MsgBox, 4,, Control %a_index% is "%A_LoopField%". Continue?  
    IfMsgBox, No  
        break  
}
```

Example #4: Display in real time the active window's control list:

```
#Persistent  
SetTimer, WatchActiveWindow, 200  
return  
WatchActiveWindow:
```

```
WinGet, ControlList, ControlList, A  
ToolTip, %ControlList%  
return
```

#WinGetActiveStats

Combines the functions of WinGetActiveTitle and WinGetPos into one command.

```
WinGetActiveStats, Title, Width, Height, X, Y
```

Parameters

Title	The name of the variable in which to store the title of the active window.
Width/Height	The names of the variables in which to store the width and height of the active window.
X, Y	The names of the variables in which to store the X and Y coordinates of the active window's upper left corner.

Remarks

If no matching window is found, the output variables will be made blank.

This command is equivalent to the following sequence:

```
WinGetTitle, Title, A
```

```
WinGetPos, X, Y, Width, Height, A
```

If the active window is a hidden window and DetectHiddenWindows is off (the default), all commands except WinShow will fail to "see" it. If there is no active window for this reason or any other, this command will set all of its output variables to be blank.

Related

[WinGetPos](#), [WinGetActiveTitle](#), [WinGetTitle](#), [WinGetClass](#), [WinGetText](#), [ControlGetText](#)

Example

~~WinGetActiveStats, Title, Width, Height, X, Y~~
~~MsgBox, The active window "%Title%" is %Width% wide`%, %Height% tall`%, and positioned at %X%`%, %Y%.~~

#WinGetActiveTitle

~~Retrieves the title of the active window.~~

~~WinGetActiveTitle, OutputVar~~

Parameters

~~OutputVar~~

~~The name of the variable in which to store the title of the active window.~~

Remarks

~~This command is equivalent to: WinGetTitle, OutputVar, A~~

Related

~~WinGetPos, WinGetActiveStats, WinGetTitle, WinGetClass, WinGetText, ControlGetText~~

Example

~~WinGetActiveTitle, Title~~

~~MsgBox, The active window is "%Title%".~~

WinGetClass

Retrieves a window's class name.

`WinGetClass, OutputVar [, WinTitle, WinText, ExcludeTitle, ExcludeText]`

Parameters

OutputVar	The name of the variable in which to store the retrieved class name.
WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the next 3 parameters are omitted, the Last Found Window will be used. If this is the letter A and the next 3 parameters are omitted, the active window will be used. To use a window's unique ID number , specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

Remarks

Only the class name is retrieved (the prefix "ahk_class" is not included in *OutputVar*).

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[WinGet](#), [WinGetTitle](#)

Example

```
WinGetClass, class, A  
MsgBox, The active window's class is "%class%".
```

[# WinGetPos](#)

Retrieves the position and size of a given window.

WinGetPos [, X, Y, Width, Height, WinTitle, WinText, ExcludeTitle, ExcludeText]

Parameters

X, Y	The names of the variables in which to store the X and Y coordinates of the target window's upper left corner. If omitted, the corresponding values will not be stored.
Width/Height	The names of the variables in which to store the width and height of the target window. If omitted, the corresponding values will not be stored.
WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the next 3 parameters are omitted, the Last Found Window will be used. If this is the letter A and the next 3 parameters are omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number , specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

Remarks

If no matching window is found, the output variables will be made blank.

~~If the WinTitle Program Manager is used, the command will retrieve the size of the desktop, which is usually the same as the current screen resolution.~~

A minimized window will still have a position and size. The values returned in this case may vary depending on OS and configuration.

To discover the name of the window and control that the mouse is currently hovering over, use [MouseGetPos](#).

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[WinMove](#), [ControlGetPos](#), [WinGetActiveStats](#), [WinGetActiveTitle](#), [WinGetTitle](#), [WinGetText](#), [ControlGetText](#)

Example

```
WinGetPos, X, Y, Width, Height, Calculator  
MsgBox, Calculator is at %X%,%Y%
```

```
WinGetPos, X, Y, , , A ; "A" to get the active window's pos.
```

```
MsgBox, The active window is at %X%,%Y%
IfWinExist, Untitled - Notepad
{
    WinGetPos, Xpos, Ypos ; Uses the window found above.
    MsgBox, Notepad is at %Xpos%,%Ypos%
}
```

WinGetText

Retrieves the text from a window.

```
WinGetText, OutputVar [, WinTitle, WinText, ExcludeTitle, ExcludeText]
```

Parameters

OutputVar	The name of the variable in which to store the retrieved text.
WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the next 3 parameters are omitted, the Last Found Window will be used. If this is the letter A and the next 3 parameters are omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number , specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

ErrorLevel

[ErrorLevel](#) is set to 1 if there was a problem or 0 otherwise.

Remarks

This command relies on accessibility support. [Please read this section](#) to avoid problems.

The text retrieved is generally the same as what Window Spy shows for that window.

Each text element ends with a carriage return and linefeed (CR+LF), which can be represented in the script as `r`n

~~The amount of text retrieved is limited to a variable's maximum capacity (which can be changed via the #MaxMem directive). As a result, this command might use a large amount of RAM if the target window (e.g. an editor with a large document open) contains a large quantity of text. To avoid this, it might be possible to retrieve only portions of the window's text by using ControlGetText instead. However, a variable's memory can be freed after use by assigning it to nothing, i.e. OutputVar =~~

~~Windows 95/98/ME may be limited to 64K for some text elements of certain windows.~~

To retrieve a list of all controls in a window, use [WinGet](#).

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

at-spi children are only iterated over 1,000 children per node max (both in/visible) to prevent too long execution time. This means that for windows with lists of many elements, this command may not return all of them, even if the ones visible are but few.

Related

[WinGetActiveStats](#), [WinGetActiveTitle](#), [WinGetTitle](#), [ControlGetText](#), [WinGetPos](#), [#MaxMem](#)

Example

```
Run, calc  
WinWait, Calculator  
WinGetText, text ; The window found above will be used.  
MsgBox, The text is:`n%text%
```

WinGetTitle

Retrieves the full title from a window.

WinGetTitle, OutputVar [, WinTitle, WinText, ExcludeTitle, ExcludeText]

Parameters

OutputVar	The name of the variable in which to store the retrieved title.
-----------	---

WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the next 3 parameters are omitted, the Last Found Window will be used. If this is the letter A and the next 3 parameters are omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number, specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

Remarks

To discover the name of the window that the mouse is currently hovering over, use [MouseGetPos](#).

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[WinGetActiveStats](#), [WinGetActiveTitle](#), [WinGetClass](#), [WinGet](#), [WinGetText](#), [ControlGetText](#), [WinGetPos](#)

Example

```
WinGetTitle, Title, A  
MsgBox, The active window is "%Title%".
```

WinHide

Hides a window.

WinHide [, WinTitle, WinText, ExcludeTitle, ExcludeText]

Parameters

WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the other 3 parameters are omitted, the Last Found Window will be used. If this is the letter A and the other 3 parameters are omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number , specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

Remarks

Use [WinShow](#) to unhide a hidden window ([DetectHiddenWindows](#) can be either On or Off to do this).

Related

[WinShow](#), [SetTitleMatchMode](#), [DetectHiddenWindows](#), [Last Found Window](#), [WinSet](#)

Example

```
Run, notepad.exe
WinWait, Untitled - Notepad
Sleep, 500
WinHide ; use the window found above
Sleep, 1000
WinShow
```

WinKill

Forces a window to close.

WinKill [, WinTitle, WinText, SecondsToWait, ExcludeTitle, ExcludeText]

Parameters

WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the other 3 window parameters are blank or omitted, the Last Found Window will be used. If this is the letter A and the other 3 window parameters are blank or omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number , specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
SecondsToWait	If omitted or blank, the command will not wait at all. If 0, it will wait 500ms. Otherwise, it will wait the indicated number of seconds (can contain a decimal point) for the window to close. If the window does not close within that period, the script will continue. ErrorLevel is not set by this command, so use IfWinExist or WinWaitClose if you need to determine for certain that a window is closed.
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

Remarks

~~This command first makes a brief attempt to close the window normally. If that fails, it will attempt to force the window closed by terminating its process.~~

If a matching window is active, that window will be closed in preference to any other matching window beneath it. In general, if more than one window matches, the uppermost (most recently used) will be closed.

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[WinClose](#), [WinWaitClose](#), [Process](#), [WinActivate](#), [SetTitleMatchMode](#), [DetectHiddenWindows](#), [Last Found Window](#), [IfWinExist](#), [IfWinActive](#), [WinWaitActive](#), [WinWait](#), [GroupActivate](#)

Example

```
IfWinExist, Untitled - Notepad  
    WinKill ; use the window found above  
else  
    WinKill, Calculator
```

#WinMaximize

Enlarges a window to its maximum size.

`WinMaximize [, WinTitle, WinText, ExcludeTitle, ExcludeText]`

Parameters

WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the next 3 parameters are omitted, the Last Found Window will be used. If this is the letter A and the next 3 parameters are omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number , specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

Remarks

Use [WinRestore](#) to unmaximize a window and [WinMinimize](#) to minimize it.

If a particular type of window does not respond correctly to `WinMaximize`, try using the following instead:

`PostMessage, 0x112, 0xF030,, WinTitle, WinText ; 0x112 = WM_SYSCOMMAND, 0xF030 = SC_MAXIMIZE`

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[WinRestore](#), [WinMinimize](#)

Example

```
Run, notepad.exe  
WinWait, Untitled - Notepad  
WinMaximize ; use the window found above
```

```
^Up::WinMaximize, A ; Assign a hotkey to maximize the active window.
```

#WinMinimize

Collapses a window into a button on the task bar.

```
WinMinimize [, WinTitle, WinText, ExcludeTitle, ExcludeText]
```

Parameters

WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the next 3 parameters are omitted, the Last Found Window will be used. If this is the letter A and the next 3 parameters are omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number , specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

Remarks

Use [WinRestore](#) or [WinMaximize](#) to unminimize a window.

~~If a particular type of window does not respond correctly to WinMinimize, try using the following instead:~~

~~PostMessage, 0x112, 0xF020,, WinTitle, WinText, 0x112 - WM_SYSCOMMAND, 0xF020 - SC_MINIMIZE~~

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[WinRestore](#), [WinMaximize](#)

Example

```
Run, notepad.exe  
WinWait, Untitled - Notepad
```

WinMinimize ; use the window found above

[^]Down::WinMinimize, A ; Assign a hotkey to minimize the active window.

WinMinimizeAll / WinMinimizeAllUndo

Minimizes all windows or undoes a previous WinMinimizeAll.

WinMinimizeAll

WinMinimizeAllUndo

Parameters

None

Remarks

On most systems, this is equivalent to the Explorer's Win-M and Win-D hotkeys.

Related

None

Example

WinMinimizeAll

WinMinimizeAllUndo

#WinMove

Changes the position and (optionally) the size of a window.

WinMove, X, Y

WinMove, WinTitle, WinText, X, Y [, Width, Height, ExcludeTitle, ExcludeText]

Parameters

X, Y	The X and Y coordinates of the upper left corner of the target window's new location. If these are the only parameters given with the command, the Last Found Window will be used as the target window.
WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the other 3 window parameters are blank or omitted, the Last Found Window will be used. If this is the letter A and the other 3 window parameters are blank or omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number , specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
Width, Height	The new size of the window. If either is omitted, blank, or the word DEFAULT, the size in that dimension will not be changed.
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

Remarks

~~If Width and Height are small (or negative), the window will go no smaller than 112 x 27 pixels. If Width and Height are large, the window will go no larger than approximately 12 pixels beyond the dimensions of the desktop.~~

Negative values are allowed for the x and y coordinates to support multi-monitor systems and to allow a window to be moved entirely off screen.

Although WinMove cannot move minimized windows, it can move hidden windows if [DetectHiddenWindows](#) is on.

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[ControlMove](#), [WinGetPos](#), [WinHide](#), [WinMinimize](#), [WinMaximize](#), [WinSet](#)

Example

```
Run, calc.exe
WinWait, Calculator
WinMove, 0, 0 ; Move the window found by WinWait.

SplashTextOn, 400, 300, Clipboard, The clipboard contains: `n%clipboard%
WinMove, Clipboard, , 0, 0 ; Move the splash window to the top left corner.
Msgbox, Press OK to dismiss the SplashText
SplashTextOff
```

WinRestore

Unminimizes or unmaximizes a window if it is ~~minimized or~~ maximized.

```
WinRestore [, WinTitle, WinText, ExcludeTitle, ExcludeText]
```

Parameters

WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the next 3 parameters are omitted, the Last Found Window will be used. If this is the letter A and the next 3 parameters are omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number , specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

Remarks

~~If a particular type of window does not respond correctly to WinRestore, try using the following instead:~~
~~PostMessage, 0x112, 0xF120,, WinTitle, WinText, 0x112 = WM_SYSCOMMAND, 0xF120 = SC_RESTORE~~

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[WinMinimize](#), [WinMaximize](#)

Example

WinRestore, Untitled - Notepad

WinSet

Makes a window "always on top" and/or transparent.

`WinSet, Attribute, Value [, WinTitle, WinText, ExcludeTitle, ExcludeText]`

Parameters

Attribute, Value	See list below.
WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the next 3 parameters are omitted, the Last Found Window will be used. If this is the letter A and the next 3 parameters are omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number , specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

Attribute, Value

AlwaysOnTop, [On|Off|Toggle]: Makes a window stay on top of all other windows. Use ON to turn on the setting, OFF to turn it off, or TOGGLE to set it to the opposite of its current state. If omitted, it defaults to TOGGLE. The word Topmost can be used in place of AlwaysOnTop.

Bottom Sends a window to the bottom of stack; that is, beneath all other windows. The effect is similar to pressing Alt-Escape. Example: `WinSet, Bottom,, WinTitle`

Transparent, N: Makes a window semi-transparent. Specify for *N* a number between 0 and 255 to indicate the degree of transparency: 0 makes the window invisible while 255 makes it opaque. In v1.0.23+, transparency may be turned off completely for a window by specifying the word OFF. This is different than specifying 255 because it may improve performance and reduce system resource usage.

TransColor, Color [N]: Makes all pixels of the chosen color invisible inside the target window, which allows the contents of the window behind it to show through. If the user clicks on an invisible pixel, the click will "fall through" to the window behind it. Specify for *Color* a color name or RGB value (see the [color chart](#) for guidance, or use [PixelGetColor](#) in its RGB mode). To additionally make the visible part of the window partially transparent, append a space (not a comma) followed by the transparency level (0-255). Example: ~~WinSet, TransColor, FFAA99 150, WinTitle~~

TransColor is often used to create on-screen displays and other visual effects; see the bottom of the [Gui](#) page for an example. TransColor requires v1.0.23+.

Only works in conjunction with Gui, Color. Not supported on non-Gui windows.

Remarks

To make the task bar transparent, use ~~WinSet, Transparent, 150, ahk_class Shell_TrayWnd~~. Similarly, to make the Start Menu transparent, follow this example:
DetectHiddenWindows, on
~~WinSet, Transparent, 150, ahk_class BaseBar~~

To make all or selected menus on the entire system transparent, keep a script such as the following always running. Note that although such a script cannot make its own menus transparent, it can make those of other scripts transparent:

```
#Persistent
SetTimer, WatchForMenu, 5
return ; End of auto-execute section.

WatchForMenu:
DetectHiddenWindows, on ; Might allow detection of menu sooner.
IfWinExist, ahk_class #32768
    WinSet, Transparent, 150 ; Uses the window found by the above line.
return
```

A script's [SplashText](#) window can be made non-AlwaysOnTop with this command.

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[WinGet](#), [WinHide](#), [WinSetTitle](#), [WinMove](#), [WinActivate](#)

Example

`WinSet, transparent, 200, Untitled - Notepad ; Make the window a little bit transparent.`

`WinSet, AlwaysOnTop, toggle, Calculator ; Toggle the always-on-top status of Calculator.`

WinSetTitle

Changes the title of a window.

WinSetTitle, NewTitle

WinSetTitle, WinTitle, WinText, NewTitle [, ExcludeTitle, ExcludeText]

Parameters

NewTitle	The new title for the window. If this is the only parameter given, the Last Found Window will be used.
WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the next 3 parameters are omitted, the Last Found Window will be used. If this is the letter A and the next 3 parameters are omitted, the active window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number , specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

Remarks

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[WinMove](#), [WinGetActiveStats](#), [WinGetActiveTitle](#), [WinGetText](#), [ControlGetText](#), [WinGetPos](#), [WinSet](#)

Example

```
WinSetTitle, Untitled - Notepad, , This is a new title
```

```
; Alternate:
```

```
Run, notepad.exe
```

WinShow

Unhides a hidden window.

WinShow [, WinTitle, WinText, ExcludeTitle, ExcludeText]

Parameters

WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the other 3 parameters are omitted, the Last Found Window will be used. To use a window class, specify <code>ahk_class ExactClassName</code> (shown by Window Spy). To use a window's unique ID number , specify <code>ahk_id IDNumber</code> .
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

Remarks

By default, WinShow is the only command that can detect hidden windows. You can change this via [DetectHiddenWindows](#).

Related

[WinHide](#), [SetTitleMatchMode](#), [DetectHiddenWindows](#), [Last Found Window](#)

Example

```
Run, notepad.exe
WinWait, Untitled - Notepad
Sleep, 500
```

```
WinHide ; use the window found above  
Sleep, 1000  
WinShow
```

WinWait

Waits until the requested window exists.

```
WinWait, WinTitle, WinText, Seconds [, ExcludeTitle, ExcludeText]
```

Parameters

WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). To use a window class, specify ahk_class ExactClassName (shown by Window Spy).
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
Seconds	How many seconds to wait before timing out and setting ErrorLevel to 1. Leave blank to wait indefinitely. Specifying 0 is the same as specifying 0.5.
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

ErrorLevel

[ErrorLevel](#) is set to 1 if the command timed out or 0 otherwise.

Remarks

If a matching window comes into existence, the command will not wait for *Seconds* to expire. Instead, it will immediately set [ErrorLevel](#) to 0, update the [Last Found Window](#), and the script will continue executing.

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

While the command is in a waiting state, new [threads](#) can be launched via [hotkey](#), [custom menu item](#), or [timer](#).

The seconds waited may not be very precise as the program internally merely does a loop with exponential back-off delay.

Related

[WinWaitActive](#), [WinWaitClose](#), [IfWinExist](#), [IfWinActive](#), [Process](#), [SetTitleMatchMode](#), [DetectHiddenWindows](#)

Example

```
Run, notepad.exe
WinWait, Untitled - Notepad, , 2
if ErrorLevel <> 0
{
    MsgBox, WinWait timed out.
    return
}
else
    WinMinimize ; minimize the window found by WinWait.
```

WinWaitActive / WinWaitNotActive

Waits until the requested window is active or not active.

WinWaitActive [, WinTitle, WinText, Seconds, ExcludeTitle, ExcludeText]
WinWaitNotActive [, WinTitle, WinText, Seconds, ExcludeTitle, ExcludeText]

Parameters

WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the other 3 window parameters are blank or omitted, the Last Found Window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number , specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
Seconds	How many seconds to wait before timing out and setting ErrorLevel to 1. Leave blank to wait indefinitely. Specifying 0 is the same as specifying 0.5.
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

ErrorLevel

[ErrorLevel](#) is set to 1 if the command timed out or 0 otherwise.

Remarks

If a matching window satisfies the command's expectation, it will not wait for *Seconds* to expire. Instead, it will immediately set [ErrorLevel](#) to 0 and the script will continue executing.

Both of these commands will update the [Last Found Window](#) if a qualified window is active when the command begins.

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

While the command is in a waiting state, new [threads](#) can be launched via [hotkey](#), [custom menu item](#), or [timer](#).

The seconds waited may not be very precise as the program internally merely does a loop with exponential back-off delay.

Related

[WinWait](#), [WinWaitClose](#), [IfWinExist](#), [IfWinActive](#), [SetTitleMatchMode](#), [DetectHiddenWindows](#)

Example

```
Run, notepad.exe
WinWaitActive, Untitled - Notepad, , 2
if ErrorLevel <> 0
{
    MsgBox, WinWait timed out.
    return
}
else
    WinMinimize ; minimize the window found by WinWaitActive.
```

WinWaitClose

Waits until the requested window does not exist.

[WinWaitClose](#), [WinTitle](#), [WinText](#), [Seconds](#) [, [ExcludeTitle](#), [ExcludeText](#)]

Parameters

WinTitle	The title or partial title of the target window (the matching behavior is determined by SetTitleMatchMode). If this and the other 3 window parameters are blank or omitted, the Last Found Window will be used. To use a window class, specify ahk_class ExactClassName (shown by Window Spy). To use a window's unique ID number , specify ahk_id IDNumber.
WinText	If present, this parameter must be a substring from a single text element of the target window. Hidden text elements are detected if DetectHiddenText is ON. <i>This option relies on accessibility support.</i>
Seconds	How many seconds to wait before timing out and setting ErrorLevel to 1. Leave blank to wait indefinitely. Specifying 0 is the same as specifying 0.5.
ExcludeTitle	Windows whose titles include this value will not be considered.
ExcludeText	Windows whose text include this value will not be considered. <i>This option relies on accessibility support.</i>

ErrorLevel

[ErrorLevel](#) is set to 1 if the command timed out or 0 otherwise.

Remarks

Whenever no instances of the specified window exist, the command will not wait for *Seconds* to expire. Instead, it will immediately set [ErrorLevel](#) to 0 and the script will continue executing.

While the command is in a waiting state, new [threads](#) can be launched via [hotkey](#), [custom menu item](#), or [timer](#).

Window titles and text are always case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

The seconds waited may not be very precise as the program internally merely does a loop with exponential back-off delay.

Related

[WinClose](#), [WinWait](#), [WinWaitActive](#), [IfWinExist](#), [IfWinActive](#), [Process](#), [SetTitleMatchMode](#), [DetectHiddenWindows](#)

Example

Run, notepad.exe

WinWait, Untitled - Notepad

WinWaitClose ; Wait for the exact window found by WinWait to be closed.

MsgBox, Notepad is now closed.

##CommentFlag

~~Changes the script's comment symbol from semicolon to some other string.~~

```
#CommentFlag NewString
```

Parameters

NewString	One or more characters that should be used as the new comment flag. Up to 15 characters may be specified.
-----------	---

Remarks

The default comment flag is semicolon (;).

The comment flag is used to indicate that text that follows it should not be acted upon by the script (comments aren't even loaded into memory when a script is launched).

A comment flag that appears on the same line as a command is not considered to mark a comment unless it has at least one space or tab to its left. For example:

~~MsgBox, Test1, This is a comment.~~

~~MsgBox, Test2, This is not a comment and will be displayed by MsgBox.~~

Related

[#EscapeChar](#)

Example

```
#CommentFlag // ; Change to C++ style of comments.
```

~~Sends any syntax error that prevents a script from launching to stdout rather than displaying a dialog.~~

#ErrorStdOut

-

Parameters

None

-

Remarks

This allows fancy editors such as Textpad, Scite, Crimson, and EditPlus to jump to the offending line when a syntax error occurs.

Since this directive would have to be added to every script, it's usually better to set up your editor to use the command line parameter /ErrorStdOut when launching an AutoHotkey script. The following example is for EditPlus:

Command: C:\Program Files\AutoHotkey\AutoHotkey.exe

Argument: /ErrorStdOut \$(FilePath)

Initial directory: \$(FileDir)

Capture output: Yes

-

Related

None

-

Example

#ErrorStdOut

##EscapeChar (and discussion of escape sequences)

~~Changes the script's escape character (e.g. accent vs. backslash).~~

#EscapeChar NewChar

Parameters

NewChar

Specifies a single character.

Remarks

The escape character defaults to accent (`).

The escape character is used to indicate that the character immediately following it should be interpreted differently than it normally would.

Escape Sequences (when accent is the escape character)

Type This	To Get This
\,	, (literal comma) Note: Commas that appear within the last parameter of a command do not need to be escaped because the program knows to treat them literally. The same is generally true for all parameters of MsgBox because it has smart comma handling.
\%	% (literal percent)
\`	` (accent, i.e. two consecutive escape characters result in a single literal character)
\;	; (literal semicolon) Note: It's necessary to escape semicolons <u>only</u> if they have a space or tab to their left.
\::	A literal pair of colons.
\n	newline (linefeed/LF)
\r	carriage return (CR)
\b	backspace
\t	tab (the more typical horizontal variety)
\v	vertical tab — corresponds to Ascii value 11. It can also be manifest in some applications by typing Control+K.
\a	alert (bell) — corresponds to Ascii value 7. It can also be manifest in some applications by typing Control+G.
\f	formfeed — corresponds to Ascii value 12. It can also be manifest in some applications by typing Control+L.

Related

The following rarely used directives also exist; their usage is shown in these examples:

#DerefChar # ; Change it from its normal default, which is %

#Delimiter / ; Change it from its normal default, which is comma.

-

Example

#EscapeChar \; Change it to be the same as AutoIt2's.

##MaxMem [v1.0.19+]

Sets the the maximum capacity of each variable to the specified number of megabytes.

#MaxMem Megabytes

-

Parameters

Megabytes

The number of megabytes to allow for each variable. A value larger than 4095 is considered to be 4095. A value less than 1 is considered to be 1.

Remarks

If this directive is unspecified in the script, it will behave as though set to 64.

The purpose of limiting each variable's capacity is to prevent a buggy script from consuming all available system memory. Raising or lowering the limit does not affect the performance of a script, nor does it change how much memory the script actually uses (except in the case of WinGetText and ControlGetText, which will be capable of retrieving more text if #MaxMem is increased).

This setting is global, meaning that it needs to be specified only once (anywhere in the script) to affect the behavior of the entire script.

-

Related

Variables, Sort, WinGetText, ControlGetText, #MaxThreads

-

Example

#Persistent [v1.0.09+]

Keeps a non-hotkey script permanently running (that is, until [ExitApp](#) is encountered).

```
#Persistent
```

Parameters

None

Remarks

If this directive is present anywhere in the script, that script will stay running after the auto-execute section (top part) of the script completes. This is useful in cases where a script contains [timers](#) and/or [custom menu items](#) but not [hotkeys](#), [hotstrings](#), or any use of the [Gui](#) command.

If this directive is added to an existing script, you might want to change some or all occurrences of [Exit](#) to be [ExitApp](#). This is because [Exit](#) will not terminate a persistent script; it terminates only the [current thread](#).

In v1.0.16+, this directive also makes scripts [#SingleInstance](#) unless [#SingleInstance Off](#) has been specified.

Related

[SetTimer](#), [Menu](#), [Exit](#), [ExitApp](#)

Example

```
#Persistent
```

#

Acknowledgements

In addition to the AutoIt authors already [mentioned](#):

Robert Yaklin: A lot of tireless testing to isolate bugs, a great first draft of the installer, as well as great suggestions for how commands **ought** to work :)

Jason Payam Ahdot: For suggesting and describing floating point support.

Jay D. Novak: For discovering many Win9x problems with the Send command, Capslock, and hotkey modifiers; and for generously sharing his wealth of code and wisdom for hotkeys, hot-strings, hook usage, and typing acceleration.

Rajat: For creating stylish replacements for the original AHK icons; a great product logo; making the syntax customizations for TextPad; discovering some bugs with the registry commands and AutoIt v2 compatibility; making SmartGUI Creator; and many other things.

beardboy: For NT4 testing to fix GetKeyState and the Send command; for a lot of help on the forum; and for many suggestions and bug reports.

Gregory F. Hogg of Hogg's Software: For writing the source code for multi-monitor support in the SysGet command.

And to everyone else who's sent in bug reports or suggestions: Thanks!

#

Arrays

In AutoHotkey, arrays are mostly conceptual: Each array is really just a collection of sequentially numbered [variables](#), with each variable being perceived by the user as an *element* of the array. AutoHotkey does not link these variables together in any way. In addition to [StringSplit](#) and [WinGet](#), any command that accepts an OutputVar or that assigns a value to a variable can be used to create an array. Once created, a command such as [StringTrimLeft](#) is usually used to access the individual elements.

Example

```
; Write to the array:  
loop, 5  
{  
    Random, array%a_index%, 1, 100 ; Put a random number into each element.  
}  
  
; Read from the array:  
loop  
{  
    StringTrimRight, element, array%a_index%, 0  
    if element =  
        break ; The end of the array has been reached.  
    MsgBox, %element%  
}
```

#

Clipboard

This is a built-in [variable](#) that reflects the current contents of the Windows clipboard if those contents can be expressed as text.

Files (such as those copied from an open Explorer window with Control-C) are considered to be text: They are auto-converted to their filenames (with full path) whenever the clipboard variable is referenced in the script. Each filename except the last is terminated by a carriage return and linefeed ([CR+LF](#)), which can be expressed in the script as [`n](#).

See the examples section of [parsing loop](#) for an easy method of extracting the individual filenames from the clipboard. To arrange the filenames in alphabetical order, use the [Sort](#) command. To write the filenames on the clipboard to a file via [FileAppend](#), first replace [`r`n](#) with [`n](#) as in this example:

```
StringReplace, clipboard, clipboard, `r`n, `n, All  
FileAppend, %clipboard%`n, C:\My File.txt
```

Other Examples

clipboard = my text ; Give the clipboard entirely new contents.

clipboard = ; Empty the clipboard.

clipboard = %clipboard% Text to append. ; Append some text to the clipboard.

StringReplace, clipboard, clipboard, ABC, DEF, All ; Replace all occurrences of ABC with DEF.

; This next example uses Control-C to copy, then displays the selected files or text:

```
clipboard =  
Send, ^c  
ClipWait  
MsgBox, Control-C copied the following contents to the clipboard: `n`n%clipboard%
```

#

Context Sensitive Help in Any Editor — by Rajat

This script Makes Ctrl+2 show the help file section for the selected command or keyword. If nothing is selected, the command name will be extracted from the editor's current line.

[Download This Script](#) | [Other Sample Scripts](#) | [Home](#)

```
$^2::  
+ If desired, uncomment and adjust the section below to make this hotkey  
+ operate only when a certain editor is active and/or it has a script file open.  
/*  
+ Do it this way to avoid the case sensitivity of IfWinNotActive.  
WinGetTitle, ActiveTitle, A
```

```
++ ActiveTitle not contains .ahk ; A script does not appear to be open.
+
+    send, ^2
+    return
+
+
+if a_OSType == WIN32_WINDOWS ; Windows 9x
+    Sleep, 500 ; Give time for the user to release the key.

clipboard_prev := clipboard
clipboard
; Use the highlighted word if there is one (since sometimes the user might
; intentionally highlight something that isn't a command).
Send, ^c
clipWait, 0.05
if ErrorLevel <> 0
+
    ; Get the entire line because editors treat cursor navigation keys differently.
    Send, {home}{end}^c
    clipWait, 0.2
    if ErrorLevel <> 0 ; Rare, so no error is reported.
+
        clipboard = clipboard_prev
        return
+
+
+autoTrim, on ; Make sure it's at its default setting.
end clipboard ; This will trim leading and trailing tabs & spaces.
clipboard = clipboard_prev ; Restore the original clipboard for the user.
loop, parse, cmd, %a_spaces%, ; The first space or comma is the end of the command.
+
    cmd = %a_LoopField%
    break ; i.e. we only need one iteration.
+
+ifWinNotExist, AutoHotkey Help
+
    ; Use non abbreviated root key to support older versions of AHK.
    RegRead, ahk_dir, HKEY_LOCAL_MACHINE, SOFTWARE\AutoHotkey, InstallDir
    if ErrorLevel <> 0
+
        ; Older versions of AHK might not have the above registry entry,
        ; so use a best guess location instead.
        ahk_dir = %ProgramFiles%\AutoHotkey
+
        ahk_help_file = %ahk_dir%\AutoHotkey.chm
        IfNotExist, %ahk_help_file%
+
            MsgBox, Could not find the help file: %ahk_help_file%.
            return
+
Run, %ahk_help_file%
```

```
WinWait, AutoHotkey Help
+
; The above has set the "last found" window which we use below.
WinActivate
WinWaitActive
StringReplace, cmd, cmd, #, {#}
Send, !n!{home}!{end}{cmd}{enter}
Return
```

#

Easy Window Dragging (requires XP/2k/NT)

Normally, a window can only be dragged by clicking on its title bar. This script extends that so that any point inside a window can be dragged. To activate this mode, hold down CapsLock or the middle mouse button while clicking, then drag the window to a new position.

[Download This Script](#) | [Other Sample Scripts](#) | [Home](#)

```
; Note: You can optionally release Capslock or the middle mouse button after
; the first click rather than holding it down the whole time.

~MButton & LButton::           ; CapsLock & LButton::           ; CoordMode, Mouse ; Switch to screen/absolute coordinates.
CapsLock & LButton::           ; MouseGetPos, EWD_MouseStartX, EWD_MouseStartY, EWD_MouseWin
SetTimer, EWD_WatchMouse, 10 ; Track the mouse as the user drags it.
Return

EWD_WatchMouse:
GetKeyState, LButtonState, LButton, P
If LButtonState = U ; Button has been released, so drag is complete.
{
    SetTimer, EWD_WatchMouse, off
    Return
}
; Otherwise, reposition the window to match the change in mouse coordinates
; caused by the user having dragged the mouse:
CoordMode, Mouse
MouseGetPos, EWD_MouseX, EWD_MouseY
EWD_DeltaX = %EWD_MouseX%
EWD_DeltaX -= %EWD_MouseStartX%
EWD_DeltaY = %EWD_MouseY%
EWD_DeltaY -= %EWD_MouseStartY%
EWD_MouseStartX = %EWD_MouseX% ; Update for the next timer call to this subroutine.
EWD_MouseStartY = %EWD_MouseY%
WinGetPos, EWD_WinX, EWD_WinY,,, ahk_id %EWD_MouseWin%
EWD_WinX += %EWD_DeltaX%
EWD_WinY += %EWD_DeltaY%
```

```
SetWinDelay, -1 ; Makes the below move faster/smoothier.  
WinMove, ahk_id %EWD_MouseWin%, %EWD_WinX%, %EWD_WinY%  
return
```

#

ErrorLevel

This is a built-in variable that is set to indicate the success or failure of some of the commands (not all commands change the value of ErrorLevel). A value of 0 usually indicates success, and any other value usually indicates failure. You can also set the value of ErrorLevel yourself.

Of special interest is that [RunWait](#) sets ErrorLevel to be the exit code of the program it ran. Most programs yield an exit code of zero if they completed successfully.

Each [thread](#) retains its own value of ErrorLevel, meaning that if the [current thread](#) is interrupted by another, when the original thread is resumed it will still have its original value of ErrorLevel, not the ErrorLevel that may have been set by the interrupting thread.

Note: Since some commands set ErrorLevel to values higher than 1, it is best not check whether ErrorLevel is 1, but instead whether ErrorLevel is not zero.

Example

```
WinWait, MyWindow, , 1
if ErrorLevel <> 0
    MsgBox, The window does not exist.
else
    MsgBox, The window exists.
```

#

Commands based on Accessibility

Some commands, such as `ControlSend` or the `ExcludeText` of window matching options, can only work if accessibility information (AT-SPI) is available. This is the only way to make them work. This means that **on the computer where the script is run**, 1. `at-spi2` needs to be installed, 2. assistive technologies need to be enabled (distribution setting), 3. several applications such as Chrome need special configuration flags. You can verify if a window properly works with `at-spi` with Window Spy ("Focused Control"). Most windows work, but these are the steps every user of your script may need to take:

- Enable the assistive technologies setting for your distribution. There's usually a single checkbox somewhere to be found to enable it. After enabling, you need to reboot.
- If the window is a Chromium-based browser such as Chrome or Brave or an Electron-based application such as VSCode, Slack, Spotify, Discord and many more (www.electronjs.org/apps), it needs to be launched with two tweaks
 - 1. Set environment variable `ACCESSIBILITY_ENABLED` to value 1. You can e.g. enable this globally by adding another line with content `ACCESSIBILITY_ENABLED=1` into the file `/etc/environment` and then restarting your computer.
 - 2. Add argument `--force-renderer-accessibility`. You can do so by editing the program's `\"Desktop file\"`, or starting it from command line and passing it there.
- Example for Chrome:

```
export ACCESSIBILITY_ENABLED=1
chrome --force-renderer-accessibility
```
- If the window is a Java application, you need to install the ATK bridge: For Debian-based applications, this is `libatk-wrapper-java`. For Arch Linux based ones, it's `java-atk-wrapper-openjdk8` (depending on the Java version).
- In the rare case that the window is an exotic, old application built with Qt4, such as some programs that haven't been maintained since 2015, you need to install `qt-at-spi`.
- According to the internet, these following environment variables may also help: `GNOME_ACCESSIBILITY=1`, `QT_ACCESSIBILITY=1`, `GTK_MODULES=gail:atk-bridge` and `QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1`. This is probably only relevant for Qt4 applications.
- If you tried all of that and it still doesn't work, this program may not support control access at all. Please consider opening up an issue at github.com/phil294/ahk_x11 so we can investigate. Almost every program out there supports accessibility.
- Most games and programs built with Tk (rare) usually never work.

Another thing to note: While all these accessibility-related commands usually perform reasonably fast, the *very first* usage of any such command will bootstrap the protocol. This can take a while, sometimes *multiple seconds*.

#

Easy Access to Favorite Folders -- by Savage

When you click the middle mouse button while certain types of windows are active, this script displays a menu of your favorite folders. Upon selecting a favorite, the script will instantly switch to that folder within the active window. The following window types are supported: 1) Standard file open or file save dialogs; 2) Explorer windows; 3) Console (command prompt) windows. The menu can also be optionally shown for unsupported window types, in which case the chosen favorite will be opened as a new Explorer window.

[Download This Script](#) | [Other Sample Scripts](#) | [Home](#)

~~+ Note: In Windows Explorer, if "View > Toolbars > Address Bar" is not enabled, the menu will not be shown if the hotkey chosen below has a tilde. If it does have a tilde, the menu will be shown, but the favorite will be opened in a new Explorer window rather than switching the active Explorer window to that folder.~~

~~+ CONFIG: CHOOSE YOUR HOTKEY~~
~~+ If your mouse has more than 3 buttons, you could try using XButton1 (the 4th) or XButton2 (the 5th) instead of MButton.~~

```
    You could also use a modified mouse button (such as ^MButton) or
    a keyboard hotkey. In the case of MBButton, the tilde (~) prefix
    is used so that MBButton's normal functionality is not lost when
    you click in other window types, such as a browser. The presence
    of a tilde tells the script to avoid showing the menu for
    unsupported window types. In other words, if there is no tilde,
    the hotkey will always display the menu, and upon selecting a
    favorite while an unsupported window type is active, a new
    Explorer window will be opened to display the contents of that
    folder.

;_Hotkey MBUTTON

; CONFIG: CHOOSE YOUR FAVORITES
; Update the special commented section below to list your favorite
; folders. Specify the name of the menu item first, followed by a
; semicolon, followed by the name of the actual path of the favorite.
; Use a blank line to create a separator line.

;+
;ITEMS IN FAVORITES MENU < Do not change this string.
Desktop ; %UserProfile%\Desktop
Favorites ; %UserProfile%\Favorites
My Documents ; %UserProfile%\My Documents

Program Files ; %ProgramFiles%
;-

; END OF CONFIGURATION SECTION
; Do not make changes below this point unless you want to change
; the basic functionality of the script.

;SingleInstance ; Needed since the hotkey is dynamically created.

Hotkey ; %f_Hotkey%; f_DisplayMenu
StringLeft ; f_HotkeyFirstChar; f_Hotkey; 1
; f_HotkeyFirstChar ; Show menu only for certain window types.
; f_AlwaysShowMenu
else
; f_AlwaysShowMenu

; Used to reliably determine whether script is compiled.
splitPath ; %A_ScriptName%; ; ; f_FileExt
;if f_FileExt = Exe ; Read the menu items from an external file.
; f_FavoritesFile = %A_ScriptDir%\Favorites.ini
;else ; Read the menu items directly from this script file.
; f_FavoritesFile = %A_ScriptFullPath%

; Read the configuration file.
; AtStartingPos = 0
; MenuItemCount = 0
```

```
loop, Read, %f_FavoritesFile?
+
+ if f_FileExt <> Exec
+
+ ; Since the menu items are being read directly from this
; script, skip over all lines until the starting line is
; arrived at.
if f_AtStartingPos == n
+
+     #f1String, A_LoopReadLine, ITEMS IN FAVORITES MENU
+     f_AtStartingPos = y
+     continue ; Start a new loop iteration.
+
+ ; Otherwise, the closing comment symbol marks the end of the list.
if A_LoopReadLine == /*
    break ; terminate the loop
+
+ ; Menu separator lines must also be counted to be compatible
; with A_ThisMenuItemPos.
+ MenuItemCount++
+ if A_LoopReadLine == ; Blank indicates a separator line.
    Menu, Favorites, Add
else
+
+     StringSplit, f_line, A_LoopReadLine, \
+     f_line1 %f_line1% ; Trim leading and trailing spaces.
+     f_line2 %f_line2% ; Trim leading and trailing spaces.
+ ; Resolve any references to variables within either field, and
+ ; create a new array element containing the path of this favorite.
+ Transform, f_path%f_MenuItemCount%, deref, %f_line2%
+ Transform, f_line1, deref, %f_line1%
+ Menu, Favorites, Add, %f_line1%, f_OpenFavorite
+
+
+ return ; End of auto execute section.

;
; Open the selected favorite
+ OpenFavorite
+ Fetch the array element that corresponds to the selected menu item.
StringTrimLeft, f_path, f_path%A_ThisMenuItemPos%, 0
+ if f_path ==
+
+     return
+ if f_class == #32770 ; It's a dialog.
+
+     if f_Edit1Pos <> ; And it has an Edit1 control.
+
+         ; Activate the window so that if the user is middle clicking
+         ; outside the dialog, subsequent clicks will also work.
+ WinActivate shk_id %f_window_id%
+ ; Retrieve any filename that might already be in the field or
```

```
    ; that it can be restored after the switch to the new folder.
ControlGetText, f_text, Edit1, ahk_id %f_window_id%
ControlSetText, Edit1, %f_path%, ahk_id %f_window_id%
ControlSend, Edit1, (Enter), ahk_id %f_window_id%
Sleep, 100 ; It needs extra time on some dialogs or in some cases.
ControlSetText, Edit1, %f_text%, ahk_id %f_window_id%
return

+
; else fall through to the bottom of the subroutine to take standard action.

+
else if f_class in ExploreWClass,CabinetWClass ; In Explorer, switch folders.
+
if f_Edit1Pos <> ; And it has an Edit1 control.
+
    ControlSetText, Edit1, %f_path%, ahk_id %f_window_id%
    ; Tck1 reported the following: "If I want to change to Folder L:\folder
    ; then the addressbar shows http://www.L:\folder.com. To solve this,
    ; I added a (right) before (Enter)".
    ControlSend, Edit1, (Right)(Enter), ahk_id %f_window_id%
return

+
; else fall through to the bottom of the subroutine to take standard action.

+
else if f_class = ConsoleWindowClass ; In a console window, CD to that directory
+
WinActivate, ahk_id %f_window_id% ; Because sometimes the mellick deactivates it.
SetKeyDelay, 0 ; This will be in effect only for the duration of this thread.
#fInString, f_path, : ; It contains a drive letter
+
    StringLeft, f_path_drive, f_path, 1
    Send %f_path_drive%.(enter)
+
Send, cd %f_path%(Enter)
return

+
Since the above didn't return, one of the following is true:
1) It's an unsupported window type but f_AlwaysShowMenu is y (yes).
2) It's a supported type but it lacks an Edit1 control to facilitate the custom
action, so instead do the default action below.
Run, Explorer %f_path% ; Might work on more systems without double quotes.
return

;
Display the menu
#DisplayMenu
;
These first few variables are set here and used by f_OpenFavorite+
WinGet, f_window_id, ID_A
WinGetClass, f_class, ahk_id %f_window_id%
if f_class in #32770,ExploreWClass,CabinetWClass ; Dialog or Explorer.
    ControlGetPos, f_Edit1Pos,Edit1, ahk_id %f_window_id%
if f_AlwaysShowMenu = n ; The menu should be shown only selectively.
```

```
+    if f_class in #32770,ExploreWClass,CabinetWClass ; Dialog or Explorer.
+
+        if f_EditPos = ; The control doesn't exist, so don't display the menu
+            return
+
+        else if f_class <> ConsoleWindowClass
+            return ; Since it's some other window type, don't display menu.
+
+    Otherwise, the menu should be presented for this type of window.
Menu, Favorites, show
return
```

#

Advanced Hotkey Features

Users of Windows NT/2000/XP and beyond may take advantage of the following additional features, some of which are unique to AutoHotkey and not available even in the most powerful commercial hotkey software.

Some of the easiest keys to reach on the keyboard are also the least frequently used. Make these keys do something useful! For example, if you rarely use the right ALT key, make it perform the action you do most often:

```
RAlt::
```

```
MsgBox You pressed the right ALT key.
```

```
return
```

You can even do the above without losing Right ALT's native function by assigning Right ALT to be a "prefix" for at least one other hotkey. In the below example, Right ALT has become a prefix, which automatically allows it to modify all other keys as it normally would. But if you press and release Right Alt without having used it to modify another key, its hotkey action (above) will take effect immediately:

```
RAlt & j::AltTab
```

-
Don't be limited to using only CTRL, ALT, SHIFT, and WIN as modifiers; you can combine **any** two keys or mouse buttons to form a custom hotkey. For example: Hold down NumPad0 and press NumPad1 to launch a hotkey (syntax: NumPad0 & NumPad1::); hold down CapsLock and press another key, or click a mouse button (syntax: CapsLock & RButton::). In this case, the state of the CapsLock key is not changed when it is used to launch the hotkey.

-
Convert the mouse wheel (or any other keys of your choice) into a complete substitute for Alt-Tab. Click the wheel to show or hide the menu, and turn it to navigate through the menu. The wheel will still function normally whenever the Alt-Tab menu isn't visible. Syntax:

```
MButton::AltTabMenu
```

```
WheelDown::AltTab
```

```
WheelUp::ShiftAltTab
```

-
Make a keyboard key **become** a mouse button, or have an action repeated continuously while you're holding down a key or mouse button. See example at the bottom of the GetKeyState page.

-
Make your hotkeys context sensitive: Have your easiest-to-reach hotkeys perform an action appropriate to the type of window you're working with. For example:

```
RControl::
```

```
IfWinActive, Untitled - Notepad
```

```
{
```

```
WinMenuItemSelectItem,,, File, Save
```

```
}
```

```
else IfWinActive, Calculator
```

```
{
```

```
Send, ^e!{tab}
```

```
}
```

```
return
```

[Hot strings](#): Define abbreviations that expand as you type them (auto-replace). No special training or scripting experience is needed. For example, a script containing the following lines would expand ceo, cfo, and btw wherever you type them:

```
::ceo::Chief Executive Officer  
::cfo::Chief Financial Officer  
::btw::by the way  
(more details)
```

[Gamers, rejoice!](#)

- Reduce wear & tear on your fingers by using virtually any key as a hotkey, including single letters, arrow keys, Numpad keys, and even the modifier keys themselves (CTRL/ALT/WIN/SHIFT).
- Create mouse hotkeys, including the mouse wheel button (MButton) and the turning of the wheel up and down (WheelUp and WheelDown). You can also combine a keyboard key with a mouse button. For example, control-left-button would be expressed as ^LButton::.
- Create "pass-through" hotkeys. For example, the left mouse button can trigger a hotkey action even while the click itself is being sent into the game normally (syntax: -LButton::).
- Use commands such as PixelSearch and PixelGetColor to automate game actions.
- Have the option of using the keyboard hook to implement hotkeys, which might be more responsive than other hotkey methods while the CPU is under load in a game. The hook might also be able to override any restrictions a game may have about which keys can be "mapped" to game actions.

See the Hotkeys section for more detailed information.

#

[AutoHotkey Script Showcase](#)

[Context Sensitive Help in Any Editor](#) — by Rajat: This script Makes Ctrl+I 2 show the help file section for the selected command or keyword. If nothing is selected, the command name will be extracted from the editor's current line.

[Easy Window Dragging](#) (requires XP/2k/NT): Normally, a window can only be dragged by clicking on its title bar. This script extends that so that any point inside a window can be dragged. To activate this mode, hold down CapsLock or the middle mouse button while clicking, then drag the window to a new position.

[Easy Access to Favorite Folders](#) — by Savage: When you click the middle mouse button while certain types of windows are active, this script displays a menu of your favorite folders. Upon selecting a favorite, the script will instantly switch to that folder within the active window. The following window types are supported: 1) Standard file open or file save dialogs; 2) Explorer windows; 3) Console (command prompt) windows. The menu can also be optionally shown for unsupported window types, in which case the chosen favorite will be opened as a new Explorer window.

[Using a Joystick as a Mouse](#): This script converts a joystick into a two button mouse. It allows each button to drag just like a mouse button and it uses virtually no CPU time. Also, it will move the cursor faster depending on how far you push the joystick from center. You can personalize various settings at the top of the script.

[Joystick Test Script](#): This script might help determine the button numbers of your joystick. It might also reveal if your joystick is in need of calibration, that is, whether the range of motion of each of its axes is from 0 to 100 percent as it should be. If calibration is needed, use the operating system's control panel or the software that came with your joystick.

[On-Screen Keyboard](#) (requires XP/2k/NT) — by Jon: This script creates a mock keyboard at the bottom of your screen that shows the keys you are pressing in real time. I made it to help me to learn to touch type (to get used to not looking at the keyboard). The size of the on-screen keyboard can be customized at the top of the script. Also, you can double click the tray icon to show or hide the keyboard.

[Minimize Window to Tray Menu](#): This script assigns a hotkey of your choice to hide any window so that it becomes an entry at the bottom of the script's tray menu. Hidden windows can then be restored individually or all at once by selecting the corresponding item on the menu. If the script exits for any reason, all the windows that it hid will be restored automatically.

[Changing MsgBox's Button Names](#): This is a working example script that uses a timer to change the names of the buttons in a MsgBox dialog. Although the button names are changed, the IfMsgBox command still requires that the buttons be referred to by their original names.

~~Numpad 000 Key~~: This example script makes the special 000 key that appears on certain keypads into an equals key. You can change the action by replacing the "Send, = " line with line(s) of your choice.

~~Using Keyboard Numpad as a Mouse~~ — by deguix: This script makes mousing with your keyboard almost as easy as using a real mouse (maybe even easier for some tasks). It supports up to five mouse buttons and the turning of the mouse wheel. It also features customizable movement speed, acceleration, and "axis inversion".

~~Seek~~ — by Phi: Navigating the Start Menu can be a hassle, especially if you have installed many programs over time. 'Seek' lets you specify a case-insensitive key word/phrase that it will use to filter only the matching programs and directories from the Start Menu, so that you can easily open your target program from a handful of matched entries. This eliminates the drudgery of searching and traversing the Start Menu.

~~ToolTip Mouse Menu~~ (requires XP/2k/NT) — by Rajat: This script displays a popup menu in response to briefly holding down the middle mouse button. Select a menu item by left clicking it. Cancel the menu by left clicking outside of it. A recent improvement is that the contents of the menu can change depending on which type of window is active (Notepad and Word are used as examples here).

~~Volume On-Screen Display (OSD)~~ — by Rajat: This script assigns hotkeys of your choice to raise and lower the master and/or wave volume. Both volumes are displayed as different color bar graphs.

~~Window Shading~~ (roll up a window to its title bar) — by Rajat: This script reduces a window to its title bar and then back to its original size by pressing a single hotkey. Any number of windows can be reduced in this fashion (the script remembers each). If the script exits for any reason, all "rolled up" windows will be automatically restored to their original heights.

[Home](#)

#

Using a Joystick as a Mouse

This script converts a joystick into a two-button mouse. It allows each button to drag just like a mouse button and it uses virtually no CPU time. Also, it will move the cursor faster depending on how far you push the joystick from center. You can personalize various settings at the top of the script.

[Download This Script](#) | [Other Sample Scripts](#) | [Home](#)

```
Increase this value to make the mouse cursor move faster.
JoyMultiplier = 0.40

Decrease this value to require less joystick displacement from center
to start moving the mouse. However, you may need to calibrate your
joystick ensuring it's properly centered to avoid cursor drift.
A perfectly tight and centered joystick could use a value of 1.
JoyThreshold = 3

Change these values to use joystick button numbers other than 1 & 2 for the
left & right mouse buttons, respectively.
ButtonLeft = 1
ButtonRight = 2

If your system has more than one joystick, increase this value to use a joystick
other than the first.
JoystickNumber = 1

END OF CONFIG SECTION Don't change anything below this point unless you want
to alter the basic nature of the script.

#SingleInstance
```

```
Hotkey, %JoystickNumber%Joy%ButtonLeft%, ButtonLeft
Hotkey, %JoystickNumber%Joy%ButtonRight%, ButtonRight

; Calculate the axis displacements that are needed to start moving the cursor.
JoyThresholdUpper = 50
JoyThresholdUpper != JoyThreshold?
JoyThresholdLower = 50
JoyThresholdLower != JoyThreshold?

SetTimer, WatchJoystick, 10 ; Monitor the movement of the joystick.
return ; End of auto execute section.

; The subroutines below do not use KeyWait because that would sometimes trap the
; WatchJoystick quasi thread beneath the wait for button up thread, which would
; effectively prevent mouse dragging with the joystick.

ButtonLeft:
SetMouseDelay, 1 ; Makes movement smoother.
MouseClick, left,,, 1, 0, D ; Hold down the left mouse button.
SetTimer, WaitForLeftButtonUp, 10
return

ButtonRight:
SetMouseDelay, 1 ; Makes movement smoother.
MouseClick, right,,, 1, 0, D ; Hold down the right mouse button.
SetTimer, WaitForRightButtonUp, 10
return

WaitForLeftButtonUp:
GetKeyState, jstate1, %JoystickNumber%Joy%ButtonLeft%
if jstate1 = D ; The button is still, down, so keep waiting.
    return
; Otherwise, the button has been released.
SetTimer, WaitForLeftButtonUp, off
SetMouseDelay, 1 ; Makes movement smoother.
MouseClick, left,,, 1, 0, U ; Release the mouse button.
return

WaitForRightButtonUp:
GetKeyState, jstate2, %JoystickNumber%Joy%ButtonRight%
if jstate2 = D ; The button is still, down, so keep waiting.
    return
; Otherwise, the button has been released.
SetTimer, WaitForRightButtonUp, off
MouseClick, right,,, 1, 0, U ; Release the mouse button.
return

WatchJoystick:
MoveMouse? = n ; Set default.
```

```
SetFormat, float, 03
GetKeyState, joyx, %JoystickNumber%JoyX
GetKeyState, joyy, %JoystickNumber%JoyY
If joyx > %JoyThresholdUpper%
+
    MoveMouse? y
    DeltaX = %joyx%
    DeltaX = %JoyThresholdUpper%
+
Else If joyx < %JoyThresholdLower%
+
    MoveMouse? y
    DeltaX = %joyx%
    DeltaX = %JoyThresholdLower%
+
Else
    DeltaX = 0
If joyy > %JoyThresholdUpper%
+
    MoveMouse? y
    DeltaY = %joyy%
    DeltaY = %JoyThresholdUpper%
+
Else If joyy < %JoyThresholdLower%
+
    MoveMouse? y
    DeltaY = %joyy%
    DeltaY = %JoyThresholdLower%
+
Else
    DeltaY = 0
If MoveMouse? y
+
    DeltaX *= %JoyMultiplier%
    DeltaY *= %JoyMultiplier%
    SetMouseDelay, 1 ; Makes movement smoother.
    MouseMove, %DeltaX%, %DeltaY%, 0, R
+
Return
```

#

Joystick Test Script

This script might help determine the button numbers of your joystick. It might also reveal if your joystick is in need of calibration, that is, whether the range of motion of each of its axes is from 0 to 100 percent as it should be. If calibration is needed, use the operating system's control panel or the software that came with your joystick.

[Download This Script](#) | [Other Sample Scripts](#) | [Home](#)

```
#singleInstance
SetFormat, float, 03 ; Omit decimal point from axis position percentages.
JoystickNumber = 1 ; Increase this to test a joystick other than the first.
GetKeyState, axis_count, %JoystickNumber%JoyAxes
## axis_count < 1
+
    MsgBox Joystick #!JoystickNumber! does not appear to be attached to the system.
    ExitApp
+
GetKeyState, joy_buttons, %JoystickNumber%JoyButtons
GetKeyState, joy_name, %JoystickNumber%JoyName
GetKeyState, joy_info, %JoystickNumber%JoyInfo
GetKeyState, joy_axes, %JoystickNumber%JoyAxes
Loop
+
buttons_down
Loop, %joy_buttons%
+
    GetKeyState, joyxa_index%, %JoystickNumber%joyxa_index%
    if joyxa_index% == 0
        buttons_down = !buttons_down!!a_space!!a_index%
+
GetKeyState, joyx, %JoystickNumber%JoyX
axis_info = !joyx%
GetKeyState, joyy, %JoystickNumber%JoyY
axis_info = !axis_info!!a_space!!a_space!!Y!!joyy%
##InString, joy_info, R
+
    GetKeyState, joyz, %JoystickNumber%JoyZ
    axis_info = !axis_info!!a_space!!a_space!!Z!!joyz%
+
##InString, joy_info, R
+
    GetKeyState, joyr, %JoystickNumber%JoyR
    axis_info = !axis_info!!a_space!!a_space!!R!!joyr%
+
##InString, joy_info, U
+
    GetKeyState, joyu, %JoystickNumber%JoyU
    axis_info = !axis_info!!a_space!!a_space!!U!!joyu%
+
##InString, joy_info, V
+
    GetKeyState, joyv, %JoystickNumber%JoyV
    axis_info = !axis_info!!a_space!!a_space!!V!!joyv%
+
##InString, joy_info, P
+
    GetKeyState, joyp, %JoystickNumber%JoyPOV
    axis_info = !axis_info!!a_space!!a_space!!POV!!joyp%
```

```
+  
ToolTip, %joy_name%`n%axis_info%`nButtons_Down: %buttons_down%`n`n(right click the tray icon to exit)  
Sleep, 100  
+
```

```
return
```

```
#
```

On-Screen Keyboard (requires XP/2k/NT) -- by Jon

This script creates a mock keyboard at the bottom of your screen that shows the keys you are pressing in real time. I made it to help me to learn to touch type (to get used to not looking at the keyboard). The size of the on-screen keyboard can be customized at the top of the script. Also, you can double-click the tray icon to show or hide the keyboard.

[Download This Script](#) | [Other Sample Scripts](#) | [Home](#)

```
/* Configuration Section: Customize the size of the on-screen keyboard and  
other options here.
```

```
/* Changing this font size will make the entire on-screen keyboard get  
larger or smaller.  
+FontSize 10  
+FontName Verdana ; This can be blank to use the system's default font.  
+FontStyle Bold ; Example of an alternative: Italic Underline
```

```
/* Names for the tray menu items:  
+MenuItemHide Hide on-screen keyboard  
+MenuItemShow Show on-screen keyboard
```

```
/* To have the keyboard appear on a monitor other than the primary, specify  
a number such as 2 for the following variable. Leave it blank to use  
the primary.  
+Monitor
```

```
/* End of configuration section. Don't change anything below this point  
unless you want to alter the basic nature of the script.
```

```
/* Alter the tray icon menu:  
Menu, Tray, Add, %k_MenuItemHide%, k_ShowHide  
Menu, Tray, Add, &Exit, k_MenuExit  
Menu, Tray, Default, %k_MenuItemHide%  
Menu, Tray, NoStandard
```

```
/* Calculate object dimensions based on chosen font size:  
+KeyWidth = %k_FontSize%  
+KeyWidth *= 2  
+KeyHeight = %k_FontSize%
```

```
*_KeyHeight + 3  
*_KeyMargin = %k_FontSize%  
*_KeyMargin / 6  
*_SpacebarWidth = %k_FontSize%  
*_SpacebarWidth += 25  
*_KeyWidthHalf = %k_KeyWidth%  
*_KeyWidthHalf / 2  
  
*_KeySize = w%k_KeyWidth% h%k_KeyHeight%  
*_Position = x%k_KeyMargin% y%k_KeySize%  
  
/* Create a GUI window for the on screen keyboard.  
Gui, Font, s%k_FontSize% %k_FontStyle%, %k_FontName%  
Gui, Caption, EB0X200 +ToolWindow  
Transcolor, F1ECCB  
Gui, Color, %TransColor%; This color will be made transparent later below.  
  
/* Add a button for each key. Position the first button with absolute  
coordinates so that all other buttons can be positioned relative to it.  
gui, Add, Button, section %k_KeySize% xm+%k_KeyWidth%, 1  
gui, Add, Button, %k_Position%, 2  
gui, Add, Button, %k_Position%, 3  
gui, Add, Button, %k_Position%, 4  
gui, Add, Button, %k_Position%, 5  
gui, Add, Button, %k_Position%, 6  
gui, Add, Button, %k_Position%, 7  
gui, Add, Button, %k_Position%, 8  
gui, Add, Button, %k_Position%, 9  
gui, Add, Button, %k_Position%, 0  
gui, Add, Button, %k_Position%,  
gui, Add, Button, %k_Position%,  
gui, Add, Button, %k_Position%, Bk  
  
gui, Add, Button, xm+ y%k_KeyMargin% h%k_KeyHeight%, Tab , Auto width.  
gui, Add, Button, %k_Position%, Q  
gui, Add, Button, %k_Position%, W  
gui, Add, Button, %k_Position%, E  
gui, Add, Button, %k_Position%, R  
gui, Add, Button, %k_Position%, T  
gui, Add, Button, %k_Position%, Y  
gui, Add, Button, %k_Position%, U  
gui, Add, Button, %k_Position%, I  
gui, Add, Button, %k_Position%, O  
gui, Add, Button, %k_Position%, P  
gui, Add, Button, %k_Position%, [,  
gui, Add, Button, %k_Position%, ]  
gui, Add, Button, %k_Position%, ^  
  
gui, Add, Button, xs+ %k_KeyWidthHalf% y+ %k_KeyMargin% %k_KeySize%, A  
gui, Add, Button, %k_Position%, S  
gui, Add, Button, %k_Position%, D
```

```
gui, Add, Button, %k_Position%, P
gui, Add, Button, %k_Position%, S
gui, Add, Button, %k_Position%, H
gui, Add, Button, %k_Position%, J
gui, Add, Button, %k_Position%, K
gui, Add, Button, %k_Position%, L
gui, Add, Button, %k_Position%, ;
gui, Add, Button, %k_Position%, !
gui, Add, Button, x!%k_KeyMargin% h%k_KeyHeight%, Enter ; Auto width.

; The first button below adds %A_Space% at the end to widen it a little,
; making the layout of keys next to it more accurately reflect a real keyboard.
gui, Add, Button, xm y!%k_KeyMargin% h%k_KeyHeight%, Shift%A_Space%%A_Space%
gui, Add, Button, %k_Position%, Z
gui, Add, Button, %k_Position%, X
gui, Add, Button, %k_Position%, C
gui, Add, Button, %k_Position%, V
gui, Add, Button, %k_Position%, B
gui, Add, Button, %k_Position%, N
gui, Add, Button, %k_Position%, M
gui, Add, Button, %k_Position%, ,
gui, Add, Button, %k_Position%, .
gui, Add, Button, %k_Position%, /
gui, Add, Button, xm y!%k_KeyMargin% h%k_KeyHeight%, Ctrl ; Auto width.
gui, Add, Button, h%k_KeyHeight% x!%k_KeyMargin%, Win ; Auto width.
gui, Add, Button, h%k_KeyHeight% x!%k_KeyMargin%, Alt ; Auto width.
gui, Add, Button, h%k_KeyHeight% x!%k_KeyMargin% w%k_SpacebarWidth%, Space

Show the window.
gui, Show
k_IsVisible = y

WinGet, k_ID, ID, A ; Get its window ID.
WinGetPos,,, k_WindowWidth, k_WindowHeight, A

; Position the keyboard at the bottom of the screen (taking into account
; the position of the taskbar)
sysGet, k_WorkArea, MonitorWorkArea, %k_Monitor%

; Calculate window's X position.
k_WindowX = %k_WorkAreaRight%
k_WindowX = %k_WorkAreaLeft% ; Now k_WindowX contains the width of this monitor.
k_WindowX = %k_Windowwidth%
k_WindowX /= 2 ; Calculate position to center it horizontally.
; The following is done in case the window will be on a non primary monitor
; or if the taskbar is anchored on the left side of the screen.
k_WindowX += %k_WorkAreaLeft%

; Calculate window's Y position.
```

```
*Window  tk_WorksAreaBottom*
*Window  tk_WindowHeight*

WinMove, A,, tk_WindowX, tk_WindowY
WinSet, AlwaysOnTop, On, ahk_id %k_ID%
WinSet, TransColor, !TransColor! 220, ahk_id %k_ID%

; Set all keys as hotkeys. See www.asciiitable.com
; ASCII 45

Loop
+
    transform, k_char, Chr, tk_ASCII
    StringUpper, k_char, k_char
    if k_char not in <,>,^,,,
        Hotkey, +%k_char%, k_KeyPress
        ; In the above, the asterisk prefix allows the key to be detected regardless
        ; of whether the user is holding down modifier keys such as Control and Shift.
        if k_ASCII == 93
            break
    tk_ASCII++
+
return ; End of auto execute section.

; When a key is pressed by the user, click the corresponding button on screen.

+Backspace:
ControlClick, Bk, ahk_id %k_ID%, , LEFT, 1, D
KeyWait, Backspace
ControlClick, Bk, ahk_id %k_ID%, , LEFT, 1, U
return

; LShift and RShift are used rather than "Shift" because when used as a hotkey,
; "Shift" would default to firing upon release of the key (in older AHK versions).
+LShift:
+RShift:
+LCtrl:
+RCtrl:
+LAlt:
+RAlt:
+LWin:
+RWin:
StringTrimLeft, k_ThisHotkey, A_ThisHotkey, 3
ControlClick, %k_ThisHotkey%, ahk_id %k_ID%, , LEFT, 1, D
KeyWait, %k_ThisHotkey%
ControlClick, %k_ThisHotkey%, ahk_id %k_ID%, , LEFT, 1, U
```

```

return

. . .
+!:
+Space:
+Enter:
+Tab:; ; The TAB key is commented out for now, since it interferes with Alt Tab.
+KeyPress:
StringReplace, %ThisHotkey%, A_ThisHotkey,
StringReplace, %_ThisHotkey%, k_ThisHotkey, *
SetTitleMatchMode, 3 ; Prevents the T and B keys from being confused with Tab and Backspace.
ControlClick, %k_ThisHotkey%, ahk_id %k_ID%, , LEFT, 1, D
KeyWait, %k_ThisHotkey%
ControlClick, %k_ThisHotkey%, ahk_id %k_ID%, , LEFT, 1, 0
Return

+ShowHide:
++ k_IsVisible y
+
    Gui, Cancel
    Menu, Tray, Rename, %k_MenuItemHide%, %k_MenuItemShow%
    k_IsVisible n
+
else
+
    Gui, Show
    Menu, Tray, Rename, %k_MenuItemShow%, %k_MenuItemHide%
    k_IsVisible y
+
return

GuiClose:
+MenuExit:
ExitApp

```

#

Language values contained in the [A_Language](#) variable. Note: Values containing letters might use either upper or lower case.

Value	Meaning
0436	Afrikaans
041c	Albanian
0401	Arabic_Saudi_Arabia

0801	Arabic_Iraq
0c01	Arabic_Egypt
1001	Arabic.Libya
1401	Arabic_Algeria
1801	Arabic_Morocco
1c01	Arabic_Tunisia
2001	Arabic_Oman
2401	Arabic_Yemen
2801	Arabic_Syria
2c01	Arabic_Jordan
3001	Arabic_Lebanon
3401	Arabic_Kuwait
3801	Arabic_UAE
3c01	Arabic_Bahrain
4001	Arabic_Qatar
042b	Armenian
042c	Azeri_Latin
082c	Azeri_Cyrillic
042d	Basque
0423	Belarusian
0402	Bulgarian
0403	Catalan
0404	Chinese_Taiwan
0804	Chinese_PRC
0c04	Chinese_Hong_Kong
1004	Chinese_Singapore
1404	Chinese_Macau
041a	Croatian

0405	Czech
0406	Danish
0413	Dutch_Standard
0813	Dutch_Belgian
0409	English_United_States
0809	English_United_Kingdom
0c09	English_Australian
1009	English_Canadian
1409	English_New_Zealand
1809	English_Irish
1c09	English_South_Africa
2009	English_Jamaica
2409	English_Caribbean
2809	English_Belize
2c09	English_Trinidad
3009	English_Zimbabwe
3409	English_Philippines
0425	Estonian
0438	Faeroese
0429	Farsi
040b	Finnish
040c	French_Standard
080c	French_Belgian
0c0c	French_Canadian
100c	French_Swiss
140c	French_Luxembourg
180c	French_Monaco
0437	Georgian

0407	German_Standard
0807	German_Swiss
0c07	German_Austrian
1007	German_Luxembourg
1407	German_Liechtenstei
408	Greek
040d	Hebrew
0439	Hindi
040e	Hungarian
040f	Icelandic
0421	Indonesian
0410	Italian_Standard
0810	Italian_Swiss
0411	Japanese
043f	Kazakh
0457	Konkani
0412	Korean
0426	Latvian
0427	Lithuanian
042f	Macedonian
043e	Malay_Malaysia
083e	Malay_Brunei_Darussalam
044e	Marathi
0414	Norwegian_Bokmal
0814	Norwegian_Nynorsk
0415	Polish
0416	Portuguese_Brazilian
0816	Portuguese_Standard

0418	Romanian
0419	
044f	Sanskrit
081a	Serbian_Latin
0c1a	Serbian_Cyrillic
041b	Slovak
0424	Slovenian
040a	Spanish_Traditional_Sort
080a	Spanish_Mexican
0c0a	Spanish_Modern_Sort
100a	Spanish_Guatemala
140a	Spanish_Costa_Rica
180a	Spanish_Panama
1c0a	Spanish_Dominican_Republic
200a	Spanish_Venezuela
240a	Spanish_Colombia
280a	Spanish_Peru
2c0a	Spanish_Argentina
300a	Spanish_Ecuador
340a	Spanish_Chile
380a	Spanish_Uruguay
3c0a	Spanish_Paraguay
400a	Spanish_Bolivia
440a	Spanish_El_Salvador
480a	Spanish_Honduras
4c0a	Spanish_Nicaragua
500a	Spanish_Puerto_Rico
0441	Swahili

041d	Swedish
081d	Swedish_Finland
0449	Tamil
0444	Tatar
041e	Thai
041f	Turkish
0422	Ukrainian
0420	Urdu
0443	Uzbek_Latin
0843	Uzbek_Cyrillic
042a	Vietnamese

#

Minimize Window to Tray Menu

This script assigns a hotkey of your choice to hide any window so that it becomes an entry at the bottom of the script's tray menu. Hidden windows can then be restored individually or all at once by selecting the corresponding item on the menu. If the script exits for any reason, all the windows that it hid will be restored automatically.

[Download This Script](#) | [Other Sample Scripts](#) | [Home](#)

CHANGES:

↑ November 3, 2004 (changes provided by tregdar):
→ Program manager is prevented from being hidden.
→ If there is no active window, the minimize to tray hotkey will have
no effect rather than waiting indefinitely.

↑

↑ October 23, 2004:
→ The taskbar is prevented from being hidden.
→ Some possible problems with long window titles have been fixed.
→ Windows without a title can be hidden without causing problems.
→ If the script is running under AHK v1.0.22 or greater, the
maximum length of each menu item is increased from 100 to 260.

↑ CONFIGURATION SECTION: Change the below values as desired.

↑ This is the maximum number of windows to allow to be hidden (having a
limit helps performance).

```
mwt_MaxWindows 50  
  
; This is the hotkey used to hide the active window.  
mwt_Hotkey #H , Win+H  
  
; If you prefer to have the tray menu empty of all the standard items,  
; such as Help and Pause, use N. Otherwise, use Y.  
mwt_StandardMenu N  
  
; These next few performance settings help to keep the action within the  
; #HotkeyModifierTimeout period, and thus avoid the need to release and  
; press down the hotkey's modifier if you want to hide more than one  
; window in a row. These settings are not needed if you choose to have the  
; script use the keyboard hook via #InstallKeyboardHook or other means.  
#HotkeyModifierTimeout 100  
SetWinDelay 10  
SetKeyDelay 0  
  
#SingleInstance ; Allow only one instance of this script to be running.  
  
; END OF CONFIGURATION SECTION (do not make changes below this point  
; unless you want to change the basic functionality of the script).  
  
Hotkey, %mwt_Hotkey%, mwt_Minimize  
  
; If the user terminates the script by any means, unhide all the  
; windows first.  
OnExit, mwt_RestoreAllThenExit  
  
if mwt_StandardMenu Y  
    Menu, Tray, Add  
else  
+  
    Menu, Tray, NoStandard  
    Menu, Tray, Add, Exit and Restore All, mwt_RestoreAllThenExit  
+  
Menu, Tray, Add, &Restore All Hidden Windows, mwt_RestoreAll  
Menu, Tray, Add ; Another separator line to make the above more special.  
  
if a_AhkVersion = ; Since it's blank, version is older than 1.0.22.  
    mwt_MaxLength = 100  
else  
    mwt_MaxLength = 260 ; Reduce this to restrict the width of the menu.  
  
return ; End of auto execute section.  
  
mwt_Minimize  
if mwt_WindowCount >= %mwt_MaxWindows%  
+  
    MsgBox No more than %mwt_MaxWindows% may be hidden simultaneously.
```

```
    return;
+
/* Set the "last found window" to simplify and help performance.
 * Since in certain cases it is possible for there to be no active window,
 * a timeout had been added.
WinWait, A,, 2
If ErrorLevel <> 0 ; It timed out, so do nothing.
    return;

/* Otherwise, the "last found window" has been set and can now be used.
WinGet, mwt_ActiveID, ID
WinGetTitle, mwt_ActiveTitle
WinGetClass, mwt_ActiveClass
If mwt_ActiveClass in Shell_TrayWnd,Program
+
    MsgBox The desktop and taskbar cannot be hidden.
    return;
+
/* Because hiding the window won't deactivate it, activate the window
beneath this one (if any). I tried other ways, but they wound up
activating the task bar. This way sends the active window (which is
about to be hidden) to the back of the stack, which seems best.
Send, !{esc}
/* Hide it only now that WinGetTitle/WinGetClass above have been run (since
by default, these commands cannot detect hidden windows).
WinHide

/* If the title is blank, use the class instead. This serves two purposes:
1) A more meaningful name is used as the menu name.
2) Allows the menu item to be created (otherwise, blank items wouldn't
be handled correctly by the various routines below).
If mwt_ActiveTitle
    mwt_ActiveTitle ahk_class %mwt_ActiveClass%
/* Ensure the title is short enough to fit. mwt_ActiveTitle also serves to
uniquely identify this particular menu item.
StringLeft, mwt_ActiveTitle, mwt_ActiveTitle, %mwt_MaxLength%

/* In addition to the tray menu requiring that each menu item name be
unique, it must also be unique so that we can reliably look it up in
the array when the window is later unhidden. So make it unique if it
isn't already.
Loop, %mwt_MaxWindows%
+
    If mwt_WindowTitle%a_index% = %mwt_ActiveTitle%
+
        /* Match found, so it's not unique.
        /* First remove the 0x from the hex number to conserve menu space.
StringTrimLeft, mwt_ActiveIDShort, mwt_ActiveID, 2
StringLen, mwt_ActiveIDShortLength, mwt_ActiveIDShort
StringLen, mwt_ActiveTitleLength, mwt_ActiveTitle
```

```
mwt_ActiveTitleLength + $mwt_ActiveIDShortLength  
mwt_ActiveTitleLength + 1 ; +1 the 1 space between title & ID.  
if mwt_ActiveTitleLength > $mwt_MaxLength  
+  
    ; Since menu item names are limited in length, trim the title  
    ; down to allow just enough room for the Window's Short ID at  
    ; the end of its name.  
    TrimCount = $mwt_ActiveTitleLength  
    TrimCount = $mwt_MaxLength  
    StringTrimRight, mwt_ActiveTitle, mwt_ActiveTitle, $TrimCount  
+  
    ; Build unique title.  
    mwt_ActiveTitle = $mwt_ActiveTitle$ $mwt_ActiveIDShort  
    break  
+  
+  
    ; First, ensure that this ID doesn't already exist in the list, which can  
    ; happen if a particular window was externally unhidden (or its app unhid  
    ; it) and now it's about to be re-hidden.  
    mwt_AlreadyExists = n  
    loop, $mwt_MaxWindows$  
+  
    if mwt_WindowID%a_index% = $mwt_ActiveID%  
    +  
        mwt_AlreadyExists = y  
        break  
+  
+  
    ; Add the item to the array and to the menu.  
    if mwt_AlreadyExists = n  
+  
        Menu, Tray, add, $mwt_ActiveTitle%, RestoreFromTrayMenu  
        mwt_WindowCount + 1  
        loop, $mwt_MaxWindows$ ; Search for a free slot.  
+  
    ; It should always find a free slot if things are designed right.  
    if mwt_WindowID%a_index% = ; An empty slot was found.  
+  
        mwt_WindowID%a_index% = $mwt_ActiveID%  
        mwt_WindowTitle%a_index% = $mwt_ActiveTitle%  
        break  
+  
+  
return  
  
RestoreFromTrayMenu:  
Menu, Tray, delete, %>_ThisMenuItem%
```

```
# Find window based on its unique title stored as the menu item name.  
Loop, $mwt_MaxWindows$  
+  
    If $mwt_WindowTitle$ = $A_ThisMenuItem$, Match found.  
    +  
        StringTrimRight, IDToRestore, $mwt_WindowID$&_index$, 0  
        WinShow, ahk_id $IDToRestore$  
        WinActivate ahk_id $IDToRestore$ ; Sometimes needed.  
        $mwt_WindowID$&_index$ ; Make it blank to free up a slot.  
        $mwt_WindowTitle$&_index$  
        $mwt_WindowCount$ - 1  
        Break  
    +  
+  
Return  
  
$mwt_RestoreAllThenExit$  
Gosub, $mwt_RestoreAll$  
ExitApp ; Do a true exit.  
  
$mwt_RestoreAll$  
Loop, $mwt_MaxWindows$  
+  
    If $mwt_WindowID$&_index$ <>  
    +  
        StringTrimRight, IDToRestore, $mwt_WindowID$&_index$, 0  
        WinShow, ahk_id $IDToRestore$  
        WinActivate ahk_id $IDToRestore$ ; Sometimes needed.  
        ; Do it this way vs. DeleteAll so that the sep. line and first  
        ; item are retained.  
        StringTrimRight, MenuToDelete, $mwt_WindowTitle$&_index$, 0  
        Menu, Tray, delete, %MenuToDelete%  
        $mwt_WindowID$&_index$ ; Make it blank to free up a slot.  
        $mwt_WindowTitle$&_index$  
        $mwt_WindowCount$ - 1  
    +  
    If $mwt_WindowCount$ = 0  
        Break  
+  
Return
```

#

Changing MsgBox's Button Names

This is a working example script that uses a timer to change the names of the buttons in a MsgBox dialog. Although the button names are changed, the IFMsgBox command still requires that the buttons be referred to by their original names.

```
#SingleInstance
SetTimer, ChangeButtonNames, 50
MsgBox, 4, Add or Delete, Choose a button...
IfMsgBox, YES
    MsgBox, You chose Add.
Else
    MsgBox, You chose Delete.
Return

ChangeButtonNames:
IfWinNotExist, Add or Delete, , return, Keep waiting.
SetTimer, ChangeButtonNames, off
WinActivate
ControlSetText, &Yes, &Add
ControlSetText, &No, &Delete
Return
```

#

Numpad 000 Key

This example script makes the special 000 key that appears on certain keypads into an equals key. You can change the action by replacing the "Send, = " line with line(s) of your choice.

```
#MaxThreadsPerHotkey 5 ; Allow multiple threads for this hotkey.
$Numpad0::
+MaxThreadsPerHotkey+
; Above: Use the $ to force the hook to be used, which prevents an
; infinite loop since this subroutine itself sends Numpad0, which
; would otherwise result in a recursive call to itself.
SetBatchLines, 100 ; Make it run a little faster in this case.
DelayBetweenKeys = 30 ; Adjust this value if it doesn't work.
If A_PriorHotkey = $A_ThisHotkey*
+
    If A_TimeSincePriorHotkey < %DelayBetweenKeys%
    +
        If Numpad0Count
            Numpad0Count = 2 ; i.e. This one plus the prior one.
        Else If Numpad0Count = 0
            Numpad0Count = 2
        Else
        +
            ; Since we're here, Numpad0Count must be 2 as set by
```

```
; prior calls, which means this is the third time the
; the key has been pressed. Thus, the hotkey sequence
; should fire.
Numpad0Count = 0
Send, - ; ***** This is the action for the 000 key
+
; In all the above cases, we return without further action.
CalledReentrantly = y
return
+
; Otherwise, this Numpad0 event is either the first in the series
; or it happened too long after the first one (e.g. perhaps the
; user is holding down the Numpad0 key to auto repeat it, which
; we want to allow). Therefore, after a short delay - during
; which another Numpad0 hotkey event may re-entrantly call this
; subroutine - we'll send the key on through if no reentrant
; calls occurred.
Numpad0Count = 0
CalledReentrantly = n
; During this sleep, this subroutine may be reentrantly called
; (i.e. a simultaneous "thread" which runs in parallel to the
; call we're in now).
Sleep, %DelayBetweenKeys%
if CalledReentrantly = y ; Another "thread" changed the value.
+
; Since it was called reentrantly, this key event was the first in
; the sequence so should be suppressed (hidden from the system).
CalledReentrantly = n
return
+
; Otherwise it's not part of the sequence so we send it through normally.
; In other words, the *real* Numpad0 key has been pressed, so we want it
; to have its normal effect.
Send, (Numpad0)
return
```

#

Using Keyboard Numpad as a Mouse — by deguix

This script makes mousing with your keyboard almost as easy as using a real mouse (maybe even easier for some tasks). It supports up to five mouse buttons and the turning of the mouse wheel. It also features customizable movement speed, acceleration, and "axis inversion".

[Download This Script](#) | [Other Sample Scripts](#) | [Home](#)



```
+Using Keyboard Numpad as a Mouse
←→
+ By deguix           / A Script file for AutoHotkey 1.0.22+
+
+
+ This script is an example of use of AutoHotkey. It uses
+ the remapping of numpad keys of a keyboard to transform it
+ into a mouse. Some features are the acceleration which
+ enables you to increase the mouse movement when holding
+ a key for a long time, and the rotation which makes the
+ numpad mouse to "turn". I.e. NumPadDown as NumPadUp
+ and vice versa. See the list of keys used below.
+
+
+ Keys          | Description
+
+ ScrollLock (toggle on) | Activates numpad mouse mode.
+
+ NumPad0        | Left mouse button click.
+ NumPad5        | Middle mouse button click.
+ NumPadDot      | Right mouse button click.
+ NumPadDiv/NumPadMult | X1/X2 mouse button click. (Win 2k+)
+ NumPadSub/NumPadAdd | Moves up/down the mouse wheel.
+
+
+ NumLock (toggled off) | Activates mouse movement mode.
+
+ NumPadEnd/Down/PgDn/ | Mouse movement.
+ /Left/Right/Home/Up/ |
+ /PgUp
+
+
+ NumLock (toggled on) | Activates mouse speed adj. mode.
+
+ NumPad7/NumPad1    | Inc./dec. acceleration per
+                     | button press.
+ NumPad8/NumPad2    | Inc./dec. initial speed per
+                     | button press.
+ NumPad9/NumPad3    | Inc./dec. maximum speed per
+                     | button press.
+ ^NumPad7/^NumPad1  | Inc./dec. wheel acceleration per
+                     | button press*.
+ ^NumPad8/^NumPad2  | Inc./dec. wheel initial speed per
+                     | button press*.
+ ^NumPad9/^NumPad3  | Inc./dec. wheel maximum speed per
+                     | button press*.
+ NumPad4/NumPad6    | Inc./dec. rotation angle to
+                     | right in degrees. (i.e. 180°
+                     | = inversed controls).
+
+
+ * - These options are affected by the mouse wheel speed
```

```
adjusted on Control Panel. If you don't have a mouse with
wheel, the default is 3 +/- lines per option button press.
+
+START OF CONFIG SECTION
#SingleInstance force
#MaxHotkeysPerInterval 500

Using the keyboard hook to implement the Numpad hotkeys prevents
them from interfering with the generation of ANSI characters such
as à. This is because AutoHotkey generates such characters
by holding down ALT and sending a series of Numpad keystrokes.
Hook hotkeys are smart enough to ignore such keystrokes.
#UseHook

MouseSpeed 1
MouseAccelerationSpeed 1
MouseMaxSpeed 5

Mouse wheel speed is also set on Control Panel. As that
will affect the normal mouse behavior, the real speed of
these three below are times the normal mouse wheel speed.
MouseWheelSpeed 1
MouseWheelAccelerationSpeed 1
MouseWheelMaxSpeed 5

MouseRotationAngle 0

+END OF CONFIG SECTION

This is needed or key presses would faulty send their natural
actions. Like NumPadUp would send sometimes "/" to the
screen.
#InstallKeyboardHook

Temp 0
Temp2 0

MouseRotationAnglePart = %MouseRotationAngle%
Divide by 45° because MouseMove only supports whole numbers,
and changing the mouse rotation to a number lesser than 45°
could make strange movements.
+
For example: 22.5° when pressing NumPadUp.
First it would move upwards until the speed
to the side reached 1.
MouseRotationAnglePart /= 45

MouseCurrentAccelerationSpeed 0
```

```
MouseCurrentSpeed = MouseSpeed;
MouseWheelCurrentAccelerationSpeed = 0;
MouseWheelCurrentSpeed = MouseSpeed;

SetKeyDelay, 1
SetMouseDelay, 1

Hotkey, *NumPad0, ButtonLeftClick
Hotkey, *NumPadIns, ButtonLeftClickIn
Hotkey, *NumPad5, ButtonMiddleClick
Hotkey, *NumPad0Clear, ButtonMiddleClickClear
Hotkey, *NumPad0Bot, ButtonRightClick
Hotkey, *NumPad0Del, ButtonRightClickDel
Hotkey, *NumPadDiv, ButtonX1Click
Hotkey, *NumPadMult, ButtonX2Click

Hotkey, *NumPadSub, ButtonWheelUp
Hotkey, *NumPadAdd, ButtonWheelDown

Hotkey, *NumPadUp, ButtonUp
Hotkey, *NumPadDown, ButtonDown
Hotkey, *NumPadLeft, ButtonLeft
Hotkey, *NumPadRight, ButtonRight
Hotkey, *NumPadHome, ButtonUpLeft
Hotkey, *NumPadEnd, ButtonUpRight
Hotkey, *NumPadPgUp, ButtonDownLeft
Hotkey, *NumPadPgDn, ButtonDownRight

Hotkey, NumPad0, ButtonSpeedUp
Hotkey, NumPad2, ButtonSpeedDown
Hotkey, NumPad7, ButtonAccelerationSpeedUp
Hotkey, NumPad1, ButtonAccelerationSpeedDown
Hotkey, NumPad9, ButtonMaxSpeedUp
Hotkey, NumPad3, ButtonMaxSpeedDown

Hotkey, NumPad6, ButtonRotationAngleUp
Hotkey, NumPad4, ButtonRotationAngleDown

Hotkey, !NumPad8, ButtonWheelSpeedUp
Hotkey, !NumPad2, ButtonWheelSpeedDown
Hotkey, !NumPad7, ButtonWheelAccelerationSpeedUp
Hotkey, !NumPad1, ButtonWheelAccelerationSpeedDown
Hotkey, !NumPad9, ButtonWheelMaxSpeedUp
Hotkey, !NumPad3, ButtonWheelMaxSpeedDown

Gosub, ScrollLock ; Initialize based on current ScrollLock state.
Return

; Key activation support
```

```
ScrollLock::  
    Wait for it to be released because otherwise the hook state gets reset  
    while the key is down, which causes the up event to get suppressed,  
    which in turn prevents toggling of the ScrollLock state/light.  
KeyWait, ScrollLock  
GetKeyState, ScrollLockState, ScrollLock, T  
## ScrollLockState ##  
+  
    Hotkey, *NumPad0, on  
    Hotkey, *NumpadIns, on  
    Hotkey, *NumPad5, on  
    Hotkey, *NumPadDot, on  
    Hotkey, *NumPadDel, on  
    Hotkey, *NumPadDiv, on  
    Hotkey, *NumPadMule, on  
  
    Hotkey, *NumpadSub, on  
    Hotkey, *NumpadAdd, on  
  
    Hotkey, *NumPadUp, on  
    Hotkey, *NumPadDown, on  
    Hotkey, *NumPadLeft, on  
    Hotkey, *NumPadRight, on  
    Hotkey, *NumPadHome, on  
    Hotkey, *NumPadEnd, on  
    Hotkey, *NumPadPgUp, on  
    Hotkey, *NumPadPgDn, on  
  
    Hotkey, Numpad0, on  
    Hotkey, Numpad2, on  
    Hotkey, Numpad7, on  
    Hotkey, Numpad1, on  
    Hotkey, Numpad9, on  
    Hotkey, Numpad3, on  
  
    Hotkey, Numpad6, on  
    Hotkey, Numpad4, on  
  
    Hotkey, !Numpad8, on  
    Hotkey, !Numpad2, on  
    Hotkey, !Numpad7, on  
    Hotkey, !Numpad1, on  
    Hotkey, !Numpad9, on  
    Hotkey, !Numpad3, on  
+  
else  
+  
    Hotkey, *NumPad0, off  
    Hotkey, *NumpadIns, off  
    Hotkey, *NumPad5, off  
    Hotkey, *NumPadDot, off
```

```
Hotkey, *NumPadDel, off
Hotkey, *NumPadDiv, off
Hotkey, *NumPadMult, off

Hotkey, *NumpadSub, off
Hotkey, *NumpadAdd, off

Hotkey, *NumPadUp, off
Hotkey, *NumPadDown, off
Hotkey, *NumPadLeft, off
Hotkey, *NumPadRight, off
Hotkey, *NumPadHome, off
Hotkey, *NumPadEnd, off
Hotkey, *NumPadPgUp, off
Hotkey, *NumPadPgDn, off

Hotkey, Numpad0, off
Hotkey, Numpad2, off
Hotkey, Numpad7, off
Hotkey, Numpad1, off
Hotkey, Numpad9, off
Hotkey, Numpad3, off

Hotkey, Numpad6, off
Hotkey, Numpad4, off

Hotkey, !Numpad0, off
Hotkey, !Numpad2, off
Hotkey, !Numpad7, off
Hotkey, !Numpad1, off
Hotkey, !Numpad9, off
Hotkey, !Numpad3, off
+
return

/*Mouse click support

ButtonLeftClick:
GetKeyState, already_down_state, LButton
if already_down_state == D
    return
Button2 = NumPad0
ButtonClick = Left
Geto_ButtonClickStart
ButtonleftClickIncr
GetKeyState, already_down_state, LButton
if already_down_state == D
    return
Button2 = NumPadIns
ButtonClick = Left
Geto_ButtonClickStart
```

```
ButtonMiddleClick:  
GetKeyState, already_down_state, MButton  
if already_down_state == D  
    return  
Button2 = NumPad5  
ButtonClick_Middle  
Goto ButtonClickStart  
ButtonMiddleClickClear:  
GetKeyState, already_down_state, MButton  
if already_down_state == D  
    return  
Button2 = NumPadClear  
ButtonClick_Middle  
Goto ButtonClickStart  
  
ButtonRightClick:  
GetKeyState, already_down_state, RButton  
if already_down_state == D  
    return  
Button2 = NumPadDot  
ButtonClick_Right  
Goto ButtonClickStart  
ButtonRightClickDel:  
GetKeyState, already_down_state, RButton  
if already_down_state == D  
    return  
Button2 = NumPadDel  
ButtonClick_Right  
Goto ButtonClickStart  
  
ButtonX1Click:  
GetKeyState, already_down_state, XButton1  
if already_down_state == D  
    return  
Button2 = NumPadDiv  
ButtonClick_X1  
Goto ButtonClickStart  
  
ButtonX2Click:  
GetKeyState, already_down_state, XButton2  
if already_down_state == D  
    return  
Button2 = NumPadMult  
ButtonClick_X2  
Goto ButtonClickStart  
  
ButtonClickStart:  
MouseClick, %ButtonClick%,,,, 1, 0, D  
SetTimer, ButtonClickEnd, 10  
return
```

```
ButtonClickEnd:  
GetKeyState, keclickstate, %Button2%, P  
## keclickstate = D  
    return  
  
SetTimer, ButtonClickEnd, off  
MouseClick, %ButtonClick%, 1, 0, U  
    return  
  
/*Mouse movement support  
  
ButtonSpeedUp:  
MouseSpeed++  
ToolTip, Mouse speed: %MouseSpeed% pixels  
SetTimer, RemoveToolTip, 1000  
    return  
ButtonSpeedDown:  
## MouseSpeed > 1  
    MouseSpeed--  
## MouseSpeed = 1  
    ToolTip, Mouse speed: %MouseSpeed% pixel  
else  
    ToolTip, Mouse speed: %MouseSpeed% pixels  
SetTimer, RemoveToolTip, 1000  
    return  
ButtonAccelerationSpeedUp:  
MouseAccelerationSpeed++  
ToolTip, Mouse acceleration speed: %MouseAccelerationSpeed% pixels  
SetTimer, RemoveToolTip, 1000  
    return  
ButtonAccelerationSpeedDown:  
## MouseAccelerationSpeed > 1  
    MouseAccelerationSpeed--  
## MouseAccelerationSpeed = 1  
    ToolTip, Mouse acceleration speed: %MouseAccelerationSpeed% pixel  
else  
    ToolTip, Mouse acceleration speed: %MouseAccelerationSpeed% pixels  
SetTimer, RemoveToolTip, 1000  
    return  
  
ButtonMaxSpeedUp:  
MouseMaxSpeed++  
ToolTip, Mouse maximum speed: %MouseMaxSpeed% pixels  
SetTimer, RemoveToolTip, 1000  
    return  
ButtonMaxSpeedDown:  
## MouseMaxSpeed > 1  
    MouseMaxSpeed--  
## MouseMaxSpeed = 1  
    ToolTip, Mouse maximum speed: %MouseMaxSpeed% pixels
```

```
else
    ToolTip, Mouse maximum speed: %MouseMaxSpeed% pixels
SetTimer, RemoveToolTip, 1000
return

ButtonRotationAngleUp:
MouseRotationAnglePart:=
If MouseRotationAnglePart >= 0
    MouseRotationAnglePart:=0
MouseRotationAngle = %MouseRotationAnglePart%
MouseRotationAngle += 45
ToolTip, Mouse rotation angle: %MouseRotationAngle%°
SetTimer, RemoveToolTip, 1000
return

ButtonRotationAngleDown:
MouseRotationAnglePart:=
If MouseRotationAnglePart < 0
    MouseRotationAnglePart+=7
MouseRotationAngle = %MouseRotationAnglePart%
MouseRotationAngle += 45
ToolTip, Mouse rotation angle: %MouseRotationAngle%°
SetTimer, RemoveToolTip, 1000
return

ButtonUp:
ButtonDown:
ButtonLeft:
ButtonRight:
ButtonUpLeft:
ButtonUpRight:
ButtonDownLeft:
ButtonDownRight:
If Button <> 0
+
    IfNotInString, A_ThisHotkey, %Button%
+
        MouseCurrentAccelerationSpeed:=0
        MouseCurrentSpeed=%MouseSpeed%
+
+
+
StringReplace, Button, A_ThisHotkey, *

ButtonAccelerationStart:
If MouseAccelerationSpeed >= 1
+
    If MouseMaxSpeed > %MouseCurrentSpeed%
+
        Temp = 0.001
        Temp += %MouseAccelerationSpeed%
        MouseCurrentAccelerationSpeed += %Temp%
        MouseCurrentSpeed = %MouseCurrentAccelerationSpeed%
```

```
+  
+  
MouseRotationAngle conversion to speed of button direction  
+  
MouseCurrentSpeedToDirection = %MouseRotationAngle%  
MouseCurrentSpeedToDirection / 90.0  
Temp = %MouseCurrentSpeedToDirection%  
  
+ if Temp > 0  
+ if Temp < 1  
+  
    MouseCurrentSpeedToDirection = 1  
    MouseCurrentSpeedToDirection += %Temp%  
    Goto EndMouseCurrentSpeedToDirectionCalculation  
  
+ if Temp > 1  
+ if Temp < 2  
+  
    MouseCurrentSpeedToDirection = 0  
    Temp = 1  
    MouseCurrentSpeedToDirection += %Temp%  
    Goto EndMouseCurrentSpeedToDirectionCalculation  
  
+ if Temp > 2  
+ if Temp < 3  
+  
    MouseCurrentSpeedToDirection = -1  
    Temp = 2  
    MouseCurrentSpeedToDirection += %Temp%  
    Goto EndMouseCurrentSpeedToDirectionCalculation  
  
+ if Temp >= 3  
+ if Temp < 4  
+  
    MouseCurrentSpeedToDirection = 0  
    Temp = 3  
    MouseCurrentSpeedToDirection += %Temp%  
    Goto EndMouseCurrentSpeedToDirectionCalculation  
  
+  
EndMouseCurrentSpeedToDirectionCalculation:
```

```
ionAngle conversion to speed of 90 degrees to right

secCurrentSpeedToSide = %MouseRotationAngle%
secCurrentSpeedToSide / = 90.0
p = %MouseCurrentSpeedToSide%
nsform, Temp, mod, %Temp%, +
Temp > 0

+if Temp < 1
+
MouseCurrentSpeedToSide = 0
MouseCurrentSpeedToSide + %Temp%
Goto EndMouseCurrentSpeedToSideCalculation
+
Temp > 1

+if Temp < 2
+
MouseCurrentSpeedToSide = 1
Temp = 1
MouseCurrentSpeedToSide + %Temp%
Goto EndMouseCurrentSpeedToSideCalculation
+
Temp > 2

+if Temp < 3
+
MouseCurrentSpeedToSide = 0
Temp = 2
MouseCurrentSpeedToSide + %Temp%
Goto EndMouseCurrentSpeedToSideCalculation
+
Temp > 3

+if Temp < 4
+
MouseCurrentSpeedToSide = 1
Temp = 3
MouseCurrentSpeedToSide + %Temp%
Goto EndMouseCurrentSpeedToSideCalculation
+
rentSpeedToSideCalculation:
tSpeedToDirection + = %MouseCurrentSpeed%
tSpeedToSide + = %MouseCurrentSpeed%  
tSpeedToSide + = %MouseCurrentSpeed%
```

```
Temp = %MouseRotationAnglePart%
Transform, Temp, Mod, %Temp%, 2

If Button = NumPadUp
+
    If Temp = 1
    +
        MouseCurrentSpeedToSide += 2
        MouseCurrentSpeedToDirection += 2
    +
        MouseCurrentSpeedToDirection -= 1
        MouseMove, %MouseCurrentSpeedToSide%, %MouseCurrentSpeedToDirection%, 0, R
    +
Else if Button = NumPadDown
+
    If Temp = 1
    +
        MouseCurrentSpeedToSide += 2
        MouseCurrentSpeedToDirection += 2
    +
        MouseCurrentSpeedToSide -= 1
        MouseMove, %MouseCurrentSpeedToSide%, %MouseCurrentSpeedToDirection%, 0, R
    +
Else if Button = NumPadLeft
+
    If Temp = 1
    +
        MouseCurrentSpeedToSide += 2
        MouseCurrentSpeedToDirection += 2
    +
        MouseCurrentSpeedToSide -= 1
        MouseCurrentSpeedToDirection -= 1
        MouseMove, %MouseCurrentSpeedToDirection%, %MouseCurrentSpeedToSide%, 0, R
    +
Else if Button = NumPadRight
+
    If Temp = 1
    +
        MouseCurrentSpeedToSide += 2
        MouseCurrentSpeedToDirection += 2
    +
        MouseMove, %MouseCurrentSpeedToDirection%, %MouseCurrentSpeedToSide%, 0, R
    +
Else if Button = NumPadHome
+
    Temp = %MouseCurrentSpeedToDirection%
```

```
Temp = %MouseCurrentSpeedToSide%
Temp += 1
Temp2 = %MouseCurrentSpeedToDirection%
Temp2 += %MouseCurrentSpeedToSide%
Temp2 += 1
Temp2 -= 1
MouseMove, %Temp%, %Temp2%, 0, R
+
else if Button == NumPadPgUp
+
Temp = %MouseCurrentSpeedToDirection%
Temp += %MouseCurrentSpeedToSide%
Temp2 = %MouseCurrentSpeedToDirection%
Temp2 += %MouseCurrentSpeedToSide%
Temp2 += 1
Temp2 -= 1
MouseMove, %Temp%, %Temp2%, 0, R
+
else if Button == NumPadEnd
+
Temp = %MouseCurrentSpeedToDirection%
Temp += %MouseCurrentSpeedToSide%
Temp += 1
Temp2 = %MouseCurrentSpeedToDirection%
Temp2 += %MouseCurrentSpeedToSide%
MouseMove, %Temp%, %Temp2%, 0, R
+
else if Button == NumPadPgDn
+
Temp = %MouseCurrentSpeedToDirection%
Temp = %MouseCurrentSpeedToSide%
Temp2 *= 1
Temp2 = %MouseCurrentSpeedToDirection%
Temp2 += %MouseCurrentSpeedToSide%
MouseMove, %Temp%, %Temp2%, 0, R
+
SetTimer, ButtonAccelerationEnd, 10
return

ButtonAccelerationEnd:
GetKeyState, kystate, %Button%, P
if kystate == D
    Goto ButtonAccelerationStart

SetTimer, ButtonAccelerationEnd, off
MouseCurrentAccelerationSpeed = 0
MouseCurrentSpeed = %MouseSpeed%
Button = 0
return

//Mouse wheel movement support
```

```
ButtonWheelSpeedUp:  
MouseWheelSpeed++  
RegRead, MouseWheelSpeedMultiplier, HKCU, Control Panel\Desktop, WheelScrollLines  
## MouseWheelSpeedMultiplier <= 0  
    MouseWheelSpeedMultiplier = 1  
MouseWheelSpeedReal = MouseWheelSpeed%  
MouseWheelSpeedReal * MouseWheelSpeedMultiplier  
ToolTip, Mouse wheel speed: %MouseWheelSpeedReal% lines  
SetTimer, RemoveToolTip, 1000  
Return  
ButtonWheelSpeedDown:  
RegRead, MouseWheelSpeedMultiplier, HKCU, Control Panel\Desktop, WheelScrollLines  
## MouseWheelSpeedMultiplier <= 0  
    MouseWheelSpeedMultiplier = 1  
## MouseWheelSpeedReal > MouseWheelSpeedMultiplier  
+  
    MouseWheelSpeed =  
    MouseWheelSpeedReal = MouseWheelSpeed%  
    MouseWheelSpeedReal * MouseWheelSpeedMultiplier%  
+  
## MouseWheelSpeedReal = 1  
    ToolTip, Mouse wheel speed: %MouseWheelSpeedReal% line  
Else  
    ToolTip, Mouse wheel speed: %MouseWheelSpeedReal% lines  
SetTimer, RemoveToolTip, 1000  
Return  
  
ButtonWheelAccelerationSpeedUp:  
MouseWheelAccelerationSpeed++  
RegRead, MouseWheelSpeedMultiplier, HKCU, Control Panel\Desktop, WheelScrollLines  
## MouseWheelSpeedMultiplier <= 0  
    MouseWheelSpeedMultiplier = 1  
MouseWheelAccelerationSpeedReal = MouseWheelAccelerationSpeed%  
MouseWheelAccelerationSpeedReal * MouseWheelSpeedMultiplier  
ToolTip, Mouse wheel acceleration speed: %MouseWheelAccelerationSpeedReal% lines  
SetTimer, RemoveToolTip, 1000  
Return  
ButtonWheelAccelerationSpeedDown:  
RegRead, MouseWheelSpeedMultiplier, HKCU, Control Panel\Desktop, WheelScrollLines  
## MouseWheelSpeedMultiplier <= 0  
    MouseWheelSpeedMultiplier = 1  
## MouseWheelAccelerationSpeed > 1  
+  
    MouseWheelAccelerationSpeed =  
    MouseWheelAccelerationSpeedReal = MouseWheelAccelerationSpeed%  
    MouseWheelAccelerationSpeedReal * MouseWheelSpeedMultiplier%  
+  
## MouseWheelAccelerationSpeedReal = 1  
    ToolTip, Mouse wheel acceleration speed: %MouseWheelAccelerationSpeedReal% line  
Else  
    ToolTip, Mouse wheel acceleration speed: %MouseWheelAccelerationSpeedReal% lines
```

```
SetTimer, RemoveToolTip, 1000
return

ButtonWheelMaxSpeedUp:
MouseWheelMaxSpeed++
RegRead, MouseWheelSpeedMultiplier, HKCU, Control Panel\Desktop, WheelScrollLines
++ MouseWheelSpeedMultiplier <= 0
    MouseWheelSpeedMultiplier = 1
MouseWheelMaxSpeedReal = %MouseWheelMaxSpeed%
MouseWheelMaxSpeedReal *= %MouseWheelSpeedMultiplier%
ToolTip, Mouse wheel maximum speed: %MouseWheelMaxSpeedReal% lines
SetTimer, RemoveToolTip, 1000
return

ButtonWheelMaxSpeedDown:
RegRead, MouseWheelSpeedMultiplier, HKCU, Control Panel\Desktop, WheelScrollLines
++ MouseWheelSpeedMultiplier <= 0
    MouseWheelSpeedMultiplier = 1
++ MouseWheelMaxSpeed > 1
+
    MouseWheelMaxSpeed =
    MouseWheelMaxSpeedReal = %MouseWheelMaxSpeed%
    MouseWheelMaxSpeedReal *= %MouseWheelSpeedMultiplier%
+
++ MouseWheelMaxSpeedReal = 1
    ToolTip, Mouse wheel maximum speed: %MouseWheelMaxSpeedReal% line
else
    ToolTip, Mouse wheel maximum speed: %MouseWheelMaxSpeedReal% lines
SetTimer, RemoveToolTip, 1000
return

ButtonWheelUp:
ButtonWheelDown:

++ Button <> 0
+
++ if Button <> %A_ThisHotkey%
+
    MouseWheelCurrentAccelerationSpeed = 0
    MouseWheelCurrentSpeed = %MouseWheelSpeed%
+
+
StringReplace, Button, A_ThisHotkey, *

ButtonWheelAccelerationStart:
++ MouseWheelAccelerationSpeed > -1
+
++ if MouseWheelMaxSpeed > %MouseWheelCurrentSpeed%
+
    Temp = 0.001
    Temp += %MouseWheelAccelerationSpeed%
    MouseWheelCurrentAccelerationSpeed += %Temp%
```

```
MouseWheelCurrentSpeed + MouseWheelCurrentAccelerationSpeed
```

+

```
If Button = NumPadSub  
    MouseClick, wheelup,,, %MouseWheelCurrentSpeed%, 0, D  
Else If Button = NumPadAdd  
    MouseClick, wheeldown,,, %MouseWheelCurrentSpeed%, 0, D
```

```
SetTimer, ButtonWheelAccelerationEnd, 100  
Return
```

```
ButtonWheelAccelerationEnd:  
GetKeyState, kstate, %Button%, P  
If kstate >  
    Goto ButtonWheelAccelerationStart
```

```
MouseWheelCurrentAccelerationSpeed = 0  
MouseWheelCurrentSpeed = %MouseWheelSpeed%  
Button = 0  
Return
```

```
RemoveToolTip  
SetTimer, RemoveToolTip, Off  
ToolTip  
Return
```

#

Overriding or Disabling Hotkeys

You can disable all built-in Windows hotkeys except Win+L and Win+U by making the following change to the registry (should work on all OSes, reboot is probably required):
~~HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer~~
~~NoWinKeys REG_DWORD 0x00000001(1)~~

~~But read on if you want to do more than just disable them all. Note that most of the below examples are not supported on Windows Me/98/95.~~

~~Hotkeys owned by another application can be overridden or disabled simply by assigning them to an action in the script. The most common use for this feature is to change the hotkeys that are built into Windows itself. For example, if you wish Win+E (the shortcut key that launches Windows Explorer) to perform some other action, use this:~~

```
#e::  
MsgBox, This hotkey is now owned by the script.  
return
```

~~In this next example, the Win+R hotkey, which is used to open the RUN window, is completely disabled:~~

```
#r::return
```

~~Similarly, to disable both Windows keys, use this:~~

```
Lwin::return  
Rwin::return
```

~~To disable or change an application's non-global hotkey (that is, a shortcut key that only works when that application is the active window), consider the following example which disables Control+P (Print) only for Notepad, leaving the key in effect for all other types of windows:~~

```
$^p::  
SetTitleMatchMode, 2  
IfWinActive, Notepad, , Return  
Send, ^p  
return
```

~~In the above example, the \$ prefix is needed so that the hotkey can "send itself" without activating itself (which would otherwise trigger a warning dialog about an infinite loop).~~

You can try out any of the above examples by copying them into a new text file such as "Override.ahk", then launching the file.

#

Script & Hotkey Performance

Scripts are semi-compiled while they're being loaded and syntax-checked. In addition to reducing the memory consumed by the script, this process also enhances runtime performance (by orders of magnitude in some cases). The performance boost is greatest when `SetBatchLines` is -1, meaning that the script is running at maximum speed.

In addition to `SetBatchLines`, the following commands may also affect performance depending on the nature of the script: `SetKeyDelay`, `SetMouseDelay`, `SetWinDelay`, `SetControlDelay`, and `SetDefaultMouseSpeed`.

Here are the technical details of the semi-compiling process:

- Input and output variables (when their names don't contain references to other variables) and group names are resolved to memory addresses.
- Loops, blocks, IFs, and ELSEs are given the memory addresses of their related jump points in the script.
- The destination of each Hotkey, Gosub, and Goto is resolved to a memory address unless it's a variable.
- Command names are replaced with their addresses in a jump table.
- Each line is parsed into a list of parameters.
- Each parameter is parsed into a list of variable references (if any).
- Each variable reference is resolved to a memory address.

#

Remapping Keys (Keyboard, Mouse and Controller)

Introduction

Limitation: AutoHotkey's remapping feature described below is generally not as pure and effective as remapping directly via the Windows registry [X keyboard extension XKB](#), see for example [here](#).

Remapping the Keyboard and Mouse

The syntax for the built-in remapping feature is `OriginKey::DestinationKey`. For example, a [script](#) consisting only of the following line would make A behave like B:

a::b

The above example does not alter B itself. B would continue to send the "b" keystroke unless you remap it to something else as shown in the following example:

```
a::b  
b::a
```

The examples above use lowercase, which is recommended for most purposes because it also remaps the corresponding uppercase letters (that is, it will send uppercase when CapsLock is "on" or Shift is held down). By contrast, specifying an uppercase letter on the right side forces uppercase. For example, the following line would produce an uppercase B when you type either "a" or "A" (as long as CapsLock is off):

```
a::B
```

However, a remapping opposite to the one above would not work as one might expect, as a remapping never "releases" the modifier keys which are used to trigger it. For example, A::b is typically equivalent to A::B and ^a::b is equivalent to ^a::^b. This is because each remapping [internally uses {Blind}](#) to allow the key or key combination to be combined with other modifiers.

Mouse Remapping

To remap the mouse instead of the keyboard, use the same approach. For example:

Example	Description
MButton::Shift	Makes the middle button behave like Shift.
XButton1::LButton	Makes the fourth mouse button behave like the left mouse button.
RAlt::RButton	Makes the right Alt behave like the right mouse button.

Other Useful Remappings

Example	Description
CapsLock::Ctrl	Makes CapsLock become Ctrl. <i>Doesn't work great currently; the CapsLock functionality cannot be disabled.</i> To retain the ability to turn CapsLock on and off, add the remapping +CapsLock::CapsLock first. This toggles CapsLock on and off when you hold down Shift and press CapsLock. Because both remappings allow additional modifier keys to be held down, the more specific +CapsLock::CapsLock remapping must be placed first for it to work.
XButton2::^LButton	Makes the fifth mouse button (XButton2) produce a control-click.
RAlt::AppsKey	Makes the right Alt become Menu (which is the key that opens the context menu).
RCtrl::RWin	Makes the right Ctrl become the right Win.
Ctrl::Alt	Makes both Ctrl behave like Alt. However, see alt-tab issues .
^x...^c	Makes Ctrl+X produce Ctrl+C. It also makes Ctrl+Alt+X produce Ctrl+Alt+C, etc.
RWin::Return	Disables the right Win by having it simply return .

You can try out any of these examples by copying them into a new text file such as "Remap.ahk", then launching the file.

See the [Key List](#) for a complete list of key and mouse button names.

Remarks

The directives [#IfWinActive/Exist](#) can be used to make selected remappings active only in the windows you specify. For example:

```
#IfWinActive ahk_class Notepad  
a::b ; Makes the 'a' key send a 'b' key, but only in Notepad.  
#IfWinActive ; This puts subsequent remappings and hotkeys in effect for all windows.
```

Remapping a key or button is "complete" in the following respects:

- Holding down a modifier such as `Ctrl` or `Shift` while typing the origin key will put that modifier into effect for the destination key. For example, `b::a` would produce `Ctrl+A` if you press `Ctrl+B`.
- `CapsLock` generally affects remapped keys in the same way as normal keys.
- The destination key or button is held down for as long as you continue to hold down the origin key. However, some games do not support remapping; in such cases, the keyboard and mouse will behave as though not remapped.
- Remapped keys will auto-repeat while being held down (except keys remapped to become mouse buttons).

When a script is launched, each remapping is translated into a pair of [hotkeys](#). For example, a script containing `a::b` actually contains the following two hotkeys instead:

```
*a::  
SetKeyDelay 1 ; If the destination key is a mouse button, SetMouseDelay is used instead.  
Send {Blind}{b DownR} ; DownR is like Down except that other Send commands in the script won't assume "b" should stay down during their Send.  
return  
  
*a up::  
SetKeyDelay 1 ; See note below for why press duration is not specified with either of these SetKeyDelays.  
Send {Blind}{b up}  
return
```

Since remappings are translated into hotkeys as described above, the [Suspend](#) command affects them. Similarly, the [Hotkey](#) command can disable or modify a remapping. For example, the following two commands would disable the remapping `a::b`.

```
Hotkey, *a, Off  
Hotkey, *a up, Off
```

~~In addition to the keys and mouse buttons on the Key List page, the source key may also be a virtual key (VKnn) or scan code (SCnnn) as described on the special keys page. The same is true for the destination key except that it may optionally specify a scan code after the virtual key. For example, sc01e...vk42sc030 is equivalent to a...b on most keyboard layouts.~~

To disable a key rather than remapping it, make it a hotkey that simply [returns](#). For example, `F1::return` would disable `F1`.

Moving the Mouse Cursor via the Keyboard

The keyboard can be used to move the mouse cursor as demonstrated by the fully-featured [Keyboard-To-Mouse script](#). Since that script offers smooth cursor movement, acceleration, and other features, it is the recommended approach if you plan to do a lot of mousing with the keyboard. By contrast, the following example is a simpler demonstration:

```
*#up::MouseMove, 0, -10, 0, R ; Win+UpArrow hotkey => Move cursor upward  
*#Down::MouseMove, 0, 10, 0, R ; Win+DownArrow => Move cursor downward  
*#Left::MouseMove, -10, 0, 0, R ; Win+LeftArrow => Move cursor to the left  
*#Right::MouseMove, 10, 0, 0, R ; Win+RightArrow => Move cursor to the right
```

```
*<#RCtrl:: ; LeftWin + RightControl => Left-click (hold down Control/Shift to Control-Click or Shift-Click).
SendEvent {Blind}{LButton down}
KeyWait RCtrl ; Prevents keyboard auto-repeat from repeating the mouse click.
SendEvent {Blind}{LButton up}
return

*<#AppsKey:: ; LeftWin + AppsKey => Right-click
SendEvent {Blind}{RButton down}
KeyWait AppsKey ; Prevents keyboard auto-repeat from repeating the mouse click.
SendEvent {Blind}{RButton up}
return
```

#

Seek --by Phi

Navigating the Start Menu can be a hassle, especially if you have installed many programs over time. 'Seek' lets you specify a case-insensitive key word/phrase that it will use to filter only the matching programs and directories from the Start Menu, so that you can easily open your target program from a handful of matched entries. This eliminates the drudgery of searching and traversing the Start Menu.

[Download This Script](#) | [Other Sample Scripts](#) | [Home](#)

```
+*****+
+ Program : Seek
+ Coder : Phi
+ Updated : Sat Oct 23 01:54:50 2004
+
+ What do you seek, my friend?
+
+*****+
+
+ I have a lot of fun coding this, and hope you will
+ enjoy using it too. Feel free to drop me an email with
+ your comments and feedback at: phi1618 (*a.t*) gmail
+ :DOT: com.
+
+ Options:
+   cache Use the cached directory listing if available
+           (this is the default mode when no option is specified)
+   scan Force a directory scan to retrieve the latest
+         directory listing
+   seek Scan & exit (this is useful for scheduling the
+             potentially time consuming directory scanning as
```

```
a background job)
help Show this help

#####
#
# HOW TO 'SEEK'!
#
# 1. 'Seek' is an AutoHotkey script. You can either run it
#    as Seek.ahk (original script) or Seek.exe (compiled
#    executable).
#
# To obtain Seek.exe, you can download Seek.zip (includes
# both the source code and the compiled binary) from
# http://home.ripway.com/2004/10/100509/
# Otherwise, you can compile Seek.ahk on your own by
# using AutoHotkey's Ahk2Exe.exe compiler, or you can
# ask me for a copy via email. The filesize is small at
# about 100 kbytes. I can be reached at: phil610 (*a.t*)
# gmail :DOT: com.
#
# To use Seek.ahk, first, you'll need to install
# AutoHotkey v1.0.21 or higher on your PC (download from
# http://www.autohotkey.com). Next, run the command:
#
# X:\myTools\AutoHotkey\AutoHotkey.exe Y:\myAHK\Seek.ahk
#
# Remember to replace X:\myTools and Y:\myAHK with
# the proper directory names on your PC.
#
# 2. You can place the executable Seek.exe anywhere you
# want. There is no installation required, it doesn't
# write anything to your registry, and it doesn't
# access the Internet at all (no phoning home). To
# uninstall, simply delete Seek.exe.
#
# The only 2 files 'Seek' creates are placed in your
# TMP directory:
#
#   a. _Seek.key (cache file for last query string)
#   b. _Seek.list (cache file for directory listing)
#
# If you're a purist, you can delete them manually
# when you decide to remove 'Seek' from your system.
#
# 3. The most convenient way to run 'Seek' is via a
# shortcut/hotkey. If you are not already using any
# hotkey management program on your PC, I highly
# recommend AutoHotkey. If you don't intend to install
# any hotkey management program at the moment, you can
# make use of Windows shortcut feature and bind a
# shortcut key (e.g. ALT F1) to launch 'Seek'. This is
```

important so that you can run 'Seek' at anytime and anywhere.

4. When you run 'Seek' for the first time, it'll scan your Start Menu, and save the directory listing into a cache file.

The following directories are included in the scanning:

 %USERPROFILE%\Start Menu
 ...\\All Users\\Start Menu

By default, subsequent runs will read from the cache file so as to reduce the loading time. For more info on options, run 'Seek.exe help'. If you think your Start Menu doesn't contain too many programs, you can choose not to use the cache and instruct 'Seek' to always do a directory scan (via option scan). That way, you will always get the latest listing.

5. When you run 'Seek', a window will appear, waiting for you to enter a key word/phrase. After you have entered a query string, a list of matching records will be displayed. Next, you need to highlight an entry and press <Enter> or click on the 'Open' button to run the selected program or open the selected directory.

TECHNICAL NOTES:

'Seek' requires Chris Mallett's AutoHotkey v1.0.21 or higher version (<http://www.autohotkey.com/>).
Thanks to Chris for his great work on AutoHotkey. :)

The following environment variables must be valid:

a. %USERPROFILE%
b. %COMSPEC%
c. %TMP%

IMPLEMENTED SUGGESTIONS:

Highlight 1st matching record by default so that user can just hit <Enter> to run it.
(Suggested by Yih Yeong)

Enable double click on the listing of the search results to launch the program.

(Suggested by Yih Yeong & Jack)
 Auto real time incremental search.
 (Suggested by Rajat)

 SUGGESTED FEATURES (MAY OR MAY NOT BE IMPLEMENTED):
 Log the launch history. List the most frequently used programs at the top of the search results.
 (Suggested by Yih Yeong)
 Instead of using list box, can it display a series of application icons so that hovering the cursor over the icon will display a tooltip containing the program information (path, etc).
 (Suggested by Yih Yeong)
 Instead of matching text in the middle, match only these program/directory names that begin with the query string.
 (Suggested by Stefan)
 Add favorites management. Launch group of programs in a single run.
 (Suggested by Atomhrt)
 Integrate Seek into the Windows taskbar/toolbar so that it is always available and there is no need to bind a hotkey to launch Seek.
 (Suggested by Deniz Akay)

 CHANGE HISTORY:
 * v1.1.0
 Initial release.
 * v1.1.1
 Removed maximise window option since some programs don't function well with it.
 Added double click detection to trigger 'Open' function.
 * v2.0.0
 Integrated the 'Seek' popup window into the output screen so that user can re enter the query string to search for something else without having to exit and run Seek again.
 Added 'Scan Start Menu' button.
 Added real time incremental search which will auto

```
filter for matching records while you type away,  
without waiting for you to press <Enter>.  
Added internal switch to track search string (ON/OFF)  
Added internal switch to show filename in tooltip (ON/OFF)  
  
* v2.0.1  
Added horizontal scrollbar to ListBox so that very  
long records will not be cut off in the middle.  
  
* v2.0.2  
Allowed user to add their own customised list of directories  
to be included in the scanning. User just needs to create a  
text file 'Seek.dir' in the same directory as Seek.exe or  
Seek.ahk, and specify the full path of the directory to be  
added, one directory per line. Do not enclose the path in  
quotes or double quotes.  
*****  
*****  
< BEGIN OF PROGRAM >  
*****  
  
Your Customisation  
  
Specify which program to use when opening a directory.  
If the program cannot be found or is not specified  
(i.e. variable is unassigned or assigned a null value),  
the default Explorer will be used.  
dirExplorer = E:\util\xplorer2_lite\xplorer2.exe  
  
User's customised list of additional directories to  
be included in the scanning. If this file is missing,  
only the default directories will be scanned.  
seekMyDir = %A_ScriptDir%\Seek.dir  
  
Specify the filename and directory location to save  
the cached directory/program listing. There is no  
need to change this unless you want to.  
listListing = %temp%\_Seek.list  
  
Specify the filename and directory location to save  
the cached key word/phrase of last search. There is  
no need to change this unless you want to.  
keyPhrase = %temp%\_Seek.key  
  
Track search string (ON/OFF)  
If ON, the last used query string will be re used as  
the default query string the next time you run Seek.  
If OFF, the last used query string will not be tracked  
and there will not be a default query string value the
```

```
next time you run Seek.
TrackKeyPhrase = ON

.....
INIT
+NOTrayIcon
StringCaseSense, Off
version Seek v2.0.2

SHOW FILENAME IN TOOLTIP (ON/OFF)
WITH THE ADDITION OF THE HORIZONTAL SCROLLBAR IN THE
LISTBOX, THERE IS NO NEED TO TURN ON THIS ANYMORE.
KEEP IT AS OFF.
ToolTipFilename OFF

DISPLAY HELP INSTRUCTIONS
++ i in help, help,/h, h,/?, ?

+
MsgBox,, %version%, Navigating the Start Menu can be a hassle, especially if you have installed many programs over time. 'Seek' lets you specify a case insensitive key word/phrase that it will use to filter only the matching programs and directories from the Start Menu, so that you can easily open your target program from a handful of matched entries. This eliminates the drudgery of searching and traversing the Start Menu. I have a lot of fun coding this, and hope you will enjoy using it too. Feel free to drop me an email with your comments and feedback at: phil610 (*at*) gmail .DOT. com. `n`nOptions: `n`nUse the cached directory listing (this is the default mode when no option is specified)`n`n -scanForce a directory scan to retrieve the latest directory listing`n`n seek tScan & exit (this is useful for scheduling the potentially time consuming directory scanning as a background job)`n`n -help`nShow this help
Goto QuitNoSave
+

CHECK THAT THE MANDATORY ENVIRONMENT VARIABLES EXIST AND ARE VALID
+USERPROFILE
+NotExist, %USERPROFILE% ; PATH DOES NOT EXIST
+
MsgBox This mandatory environment variable is either not defined or invalid.`n`n %USERPROFILE% - %USERPROFILE%`n`nPlease fix it before running Seek.
Goto QuitNoSave
+
+COMSPEC
+NotExist, %COMSPEC% ; COMSPEC EXECUTABLE FILE DOES NOT EXIST
+
MsgBox This mandatory environment variable is either not defined or invalid.`n`n %COMSPEC% - %COMSPEC%`n`nPlease fix it before running Seek.
Goto QuitNoSave
+
+TMP+
+NotExist, %TMP% ; PATH DOES NOT EXIST
+
MsgBox This mandatory environment variable is either not defined or invalid.`n`n %TMP% - %TMP%`n`nPlease fix it before running Seek.
Goto QuitNoSave
+

IF NOT SCAN AND EXIT
+NotEqual 1, seek
+
RETRIEVE THE LAST USED KEY PHRASE FROM CACHE FILE
TO BE USED AS THE DEFAULT QUERY STRING
```

```
++ TrackKeyPhrase ON
    FileReadLine, PrevKeyPhrase, %keyPhrase%, 1
NewKeyPhrase = %PrevKeyPhrase%

; ADD THE TEXT BOX FOR USER TO ENTER THE QUERY STRING
Gui, 1:Add, Edit, vFilename W600, %PrevKeyPhrase%

; ADD MY FAV TAGLINE
Gui, 1:Add, Text, X625 Y10, What do you seek, my friend?

; ADD THE SELECTION LISTBOX FOR DISPLAYING SEARCH RESULTS
Gui, 1:Add, ListBox, vOpenTarget gTargetSelection X10 Y33 R30 W764 Hscroll, %List%
GuiControl, 1:Disable, OpenTarget

; ADD THE EXIT BUTTON. THIS ALSO SERVES TO FORCE THE GUI
; TO THE PROPER SIZE SO THAT SUBSEQUENT SHOW COMMAND WILL
; NOT NEED TO DO ANY RESIZING. ALSO, THIS SOLVES THE
; MISSING BUTTONS PROBLEM ON SOME PCS.
Gui, 1:Add, Button, gButtonEXIT X743 Y446, Exit

; POP UP THE QUERY WINDOW
Gui, 1>Show, Center, %version%
+ 

; ENABLE RE SCANNING OF LATEST DIRECTORY LISTING
If 1 In Scan, seek
    reScan = Y
; CHECK WHETHER THE DIRECTORY LISTING CACHE FILE ALREADY EXISTS. IF NOT, DO A RE SCAN.
Else IfNotExist, %diListing%
    reScan = Y

If reScan = Y, DO A RE SCAN
+
; DISPLAY FEEDBACK TOOLTIP UNLESS USER SPECIFIES SCAN AND EXIT OPTION
IfNotEqual 1, seek, Tooltip, Retrieving Directory Listing..., 0,503

; SCAN START MENU AND STORE DIRECTORY/PROGRAM LISTINGS IN CACHE FILE
Gosub ScanStartMenu

; QUIT IF USER SPECIFIES SCAN AND EXIT OPTION
IfEqual 1, seek, Goto, QuitNoSave
+
; RETRIEVE THE MATCHING LIST FOR THE LAST USED KEY PHRASE
Gosub SilentFindMatches

; DIRECTORY LISTING IS NOW LOADED. ADD THE OTHER BUTTONS TO WINDOW.
; THESE BUTTONS ARE NOT ADDED EARLIER BECAUSE THEY SHOULD NOT BE
; FUNCTIONAL UNTIL THIS PART OF THE SCRIPT.
Gui, 1:Add, Button, gButtonOPEN Default X10 Y446, Open
Gui, 1:Add, Button, gButtonOPENDIR X50 Y446, Open Directory
```

```
eui, 1:Add, Button, gButtonSCANSTARTMENU X340 Y416, Scan Start Menu  
  
    ; TURN ON INCREMENTAL SEARCH  
SetTimer, tIncrementalSearch, 500  
  
    ; REFRESH THE GUI  
Gosub EnterQuery  
  
Return  
  
*****  
-- END OF MAIN PROGRAM --  
*****  
  
-----  
-- BEGIN ButtonSCANSTARTMENU EVENT -----  
  
ButtonSCANSTARTMENU:  
  
eui, 1:Submit, NoHide  
  
    ; DISABLE LISTBOX WHILE SCANNING IS IN PROGRESS  
EuiControl, 1:Disable, OpenTarget  
  
    ; DO THE SCANNING  
Gosub ScanStartMenu  
  
    ; INFORM USER THAT SCANNING HAS COMPLETED  
##Filename  
+  
    MsgBox, 0102, %version%, Scan completed.  
Gosub EnterQuery  
+  
Else  
+  
    ; FILTER FOR SEARCH STRING WITH THE NEW LISTING  
MsgBox, 0102, %version%, Scan completed. Press OK to proceed to search for "%Filename%".  
NewKeyPhrase  
Gosub FindMatches  
+  
Return  
  
---- END ButtonSCANSTARTMENU EVENT -----  
  
-----  
-- BEGIN ScanStartMenu SUBROUTINE -----  
-- SCAN THE START MENU AND STORE THE DIRECTORY/PROGRAM  
-- LISTINGS IN A CACHE FILE  
ScanStartMenu:  
  
    ; DEFINE THE DIRECTORY PATHS TO RETRIEVE
```

```
scanPath = "%USERPROFILE%\Start Menu"
splitPath, USERPROFILE, name, dir, ext, name_no_ext, drive
scanPath = %scanPath%\%dir%\All Users\Start Menu"

; INCLUDE ADDITIONAL USER DEFINED PATHS FOR SCANNING
; fExist, %SeekMyDir%
+
+ Loop, read, %SeekMyDir%
+
+ IFNotExist, %A_LoopReadLine%
    MsgBox, 0x1000, %version%, Processing your customised directory list...`n`n%A_LoopReadLine% does not exist and will be excluded from the scanning.`nPlease update [ %SeekMyDir% ].
Else
    scanPath = %scanPath%\%A_LoopReadLine%
+
+
; DELETED EXISTING FILE BEFORE CREATING A NEW VERSION
FileDelete, %dirListing%

; ACTUAL RETRIEVAL OF DIRECTORY LISTING IS DONE VIA THE 'DIR' COMMAND
RunWait, %COMSPEC% /c dir /s /b %scanPath% > %dirListing%, Hide

; HIDE THE FEEDBACK TOOLTIP WHEN SCANNING IS DONE
Tooltip
Return

;... END ScanStartMenu SUBROUTINE .....
```



```
;----- BEGIN FindMatches SUBROUTINE
;----- SEARCH AND DISPLAY ALL MATCHING RECORDS IN THE LISTBOX
FindMatches:
eai, 1:Submit, NoHide
Tooltip

; CHECK FOR EMPTY QUERY STRING
; If filename
+
+ MsgBox, 0x1000, %version%, Please enter the key word/phrase to search for.
Goto EnterQuery
+
; tIncrementalSearch IS BEING INTERRUPTED. LET IT FINISHES.
; If NewKeyPhrase <> %filename%
+
+ ; INFORM USER THAT PATIENCE IS A VIRTUE
Tooltip, Seeking for matching records..., 0, 500
ResumeFindMatches = TRUE
Return
```

```
if List = +
+
    NOT EVEN A SINGLE MATCHING RECORD IS FOUND.
    LET USER MODIFY THE QUERY STRING AND TRY AGAIN.
    MsgBox, 0102, iversion$, The query string "%filename%" does not match any record. Try again.
    GuiControl, 1:Disable, OpenTarget
    Goto EnterQuery
+
else
+
    SELECT THE FIRST RECORD IF NO OTHER RECORD HAS BEEN SELECTED
    Gui, 1:Submit, NoHide
    GuiControl, 1:Enable, OpenTarget
    GuiControl, Focus, OpenTarget
    if OpenTarget
        GuiControl, 1:Choose, OpenTarget, +1
+
REFRESH GUI
Gui, 1>Show, Center, iversion$
Return
.... END FindMatches SUBROUTINE .....
```

```
----- BEGIN SilentFindMatches SUBROUTINE -----
SilentFindMatches:
    Gui, 1:Submit, NoHide
    sfmFilename = %filename%
    FILTER MATCHING RECORDS BASED ON USER QUERY STRING
list =
    sfmFilename =>
+
    loop, read, %dirListing%
+
        Gui, 1:Submit, NoHide
        if sfmFilename <> %filename%
+
            USER HAS CHANGED THE SEARCH STRING. THERE IS NO POINT
            TO CONTINUE SEARCHING USING THE OLD STRING, SO ABORT.
            Return
+
    else
+
        APPEND MATCHING RECORDS INTO THE LIST
SplitPath, A_LoopReadLine, name, dir, ext, name_no_ext, drive
```

```
    ;>FINISTERE, name, %$fmFilename%
    ;>List ->%List%_LoopReadLine++
```

+
+

```
    ;>REFRESH LIST WITH SEARCH RESULTS
    GuiControl, 1:, OpenTarget, %List%
```

+
+
+ List +
+
 ;> NO MATCHING RECORD IS FOUND
 ;> DISABLE LISTBOX
 GuiControl, 1:Disable, OpenTarget

+
+ Else
+
 ;> MATCHING RECORDS ARE FOUND
 ;> ENABLE LISTBOX
 GuiControl, 1:Enable, OpenTarget

+
+
 ;>REFRESH GUI
 Gui, 1>Show, Center, %version%

Return

```
.... END SilentFindMatches SUBROUTINE .....
```

+
+
+ BEGIN EnterQuery SUBROUTINE
+>REFRESH GUI AND LET USER ENTERS SEARCH STRING
EnterQuery:
 GuiControl, Focus, Filename
 Gui, 1>Show, Center, %version%
Return

```
.... END EnterQuery SUBROUTINE .....
```

+
+
+ BEGIN TargetSelection EVENT

```
TargetSelection:
    Gui, 1:Submit, NoHide
```

+>DOUBLE CLICK DETECTION TO LAUNCH PROGRAM
+ If A_GuiControlEvent DoubleClick
+
 Goseub ButtonOPEN

+
+ Else
+

```
    ; DISPLAY THE TOOLTIP FOR THE CURRENTLY HIGHLIGHTED RECORD.  
    ; THIS IS USEFUL IF THE RECORD IS TOO LONG AND COULDN'T FIT IN THE WINDOW.  
    ; THE TOOLTIP WILL SHOW YOU THE COMPLETE PATH AND FILE NAME.  
    ;  
    ;if A_GuiControlEvent ==Normal  
    +  
        ;if ToolTipFilename == ON  
        +  
            Tooltip, %OpenTarget%, 0,500  
            SetTimer, tRemoveTip, 6000 ; REMOVE TOOLTIP AFTER 6 SECS  
        +  
    +  
Return  
  
.... END TargetSelection EVENT .....  
---- BEGIN ButtonOPEN EVENT ----  
  
    ; USER CLICKED ON 'OPEN' BUTTON OR PRESSED <ENTER>  
    ;  
    ;if Gui, 1:Submit, NoHide  
    ;  
    ; FIND OUT WHERE THE KEYBOARD FOCUS WAS. IF IT'S THE  
    ; TEXT FIELD, RUN THE QUERY TO FIND MATCHES. ELSE, IT  
    ; MUST BE FROM THE LISTBOX.  
    ;  
    ;if GuiControlGet, focusControl, 1:Focus  
    ;if focusControl == Edit1  
    +  
        GuiControl, Focus, OpenTarget  
        GuiControl, 1:Disable, OpenTarget  
        Goto FindMatches  
    +  
    ; NO RECORD FROM THE LISTBOX IS SELECTED  
    ;if OpenTarget ==  
    +  
        MsgBox, 8192, %version%, Please make a selection before hitting <Enter>. `nPress <Esc> to exit.  
        Goto EnterQuery  
    +  
    ; SELECTED RECORD DOES NOT EXIST (FILE OR DIRECTORY NOT FOUND)  
    ;if NotExist, %OpenTarget%  
    +  
        MsgBox, 8192, %version%, %OpenTarget% does not exist. This means that the directory cache is outdated. You may click on the 'Scan Start Menu' button below to update the directory cache with your latest directory listing now.  
        Goto EnterQuery  
    +  
    ; CHECK WHETHER THE SELECTED RECORD IS A FILE OR DIRECTORY  
    ;FileGetAttrib, fileAttrib, %OpenTarget%
```

```
+  
+      IfInString, fileAttrib, D , IS DIRECTORY  
+  
+          Gosub sOpenDir  
+  
+      Else If fileAttrib <> ; IS FILE  
+  
+          Run, tOpenTarget  
+  
+      Else  
+  
+          MsgBox %OpenTarget% is neither a DIRECTORY or a FILE. This shouldn't happen. Seek cannot proceed. Quitting...  
+  
+  
Goto Quit  
  
.... END ButtonOPEN EVENT .....  
  
---- BEGIN ButtonOPENDIR EVENT ----  
  
USER CLICKED ON 'OPEN DIRECTORY' BUTTON  
ButtonOPENDIR:  
Gui, l:Submit, NoHide  
  
CHECK THAT USER HAS SELECTED A RECORD ALREADY  
If OpenTarget  
+  
+            MsgBox, 0102, iVersion!, Please make a selection first.  
            Goto EnterQuery  
+  
+  
RUN SUBROUTINE TO OPEN A DIRECTORY  
Gosub sOpenDir  
  
Goto Quit  
  
.... END ButtonOPENDIR EVENT .....  
  
---- BEGIN sOpenDir SUBROUTINE ----  
  
OpenDir:  
  
IF USER SELECTED A FILE RECORD INSTEAD OF A DIRECTORY RECORD,  
EXTRACT THE DIRECTORY PATH. (I'M USING DriveGet INSTEAD OF  
FileGetAttrib TO ALLOW THE SCENARIO WHEREBY OpenTarget IS  
INVALID BUT THE DIRECTORY PATH OF OpenTarget IS VALID.  
DriveGet, status, status, tOpenTarget  
If status <> Ready ; NOT A DIRECTORY  
+  
+            SplitPath, OpenTarget, name, dir, ext, name_no_ext, drive
```

```
    OpenTarget ->dir?
+
+-----+
    ✓ CHECK WHETHER DIRECTORY EXISTS
    IfNotExist, %OpenTarget%
+
+-----+
        MsgBox, 0102, tversion, %OpenTarget% does not exist. This means that the directory cache is outdated. You may click on the 'Scan Start Menu' button below to update the directory cache with your latest directory listing now.
        Goto EnterQuery
+
+-----+
    ✓ OPEN THE DIRECTORY
    IfExist, %dirExplorer%
+
+-----+
        Run, "%dirExplorer%" "%OpenTarget%", , Max , OPEN WITH CUSTOMISED FILE EXPLORER
+
Else
+
        Run, %OpenTarget%, , Max , OPEN WITH DEFAULT WINDOWS FILE EXPLORER
+
Return
+
.... END subOpenDir SUBROUTINE .....
```



```
    ✓ BEGIN tIncrementalSearch EVENT
    ✓ AUTOMATICALLY CONDUCT REAL TIME INCREMENTAL SEARCH
    ✓ TO FIND MATCHING RECORDS WITHOUT WAITING FOR USER
    ✓ TO PRESS <ENTER>
    IncrementalSearch:
+
Loop
    ✓ REPEAT SEARCHING UNTIL USER HAS STOPPED CHANGING THE QUERY STRING
+
    Gui, 1,Submit, NoHide
    CurFilename = %filename%
    If NewKeyPhrase <> %CurFilename%
    +
        OpenTarget -
        Scanb SilentFindMatches
        NewKeyPhrase = %CurFilename%
        Sleep, 100 ; DON'T HOG THE CPU!
    +
Else
+
    ; QUERY STRING HAS STOPPED CHANGING
    Break
+
+
    ✓ USER HAS HIT <ENTER> TO LOOK FOR MATCHING RECORDS.
    ✓ RUN FindMatches NOW
```

```
++ ResumeFindMatches TRUE
+
+      ResumeFindMatches FALSE
    EndSub FindMatches
+
+
/* CONTINUE MONITORING FOR CHANGES
SetTimer, tIncrementalSearch, 500
Return
.... END tIncrementalSearch EVENT .....
```



```
/* BEGIN tRemoveTip EVENT */
-
- REMOVE TOOLTIP
tRemoveTip:
SetTimer, tRemoveTip, Off
ToolTip
Return
.... END tRemoveTip EVENT .....
```



```
/* BEGIN Quit SUBROUTINE */
-
exit:
ButtonEXIT:
GuiClose:
GuiEscape:
-
Gui, 1, Submit, NoHide
-
/* SAVE THE KEY WORD/PHRASE FOR NEXT RUN IF IT HAS CHANGED
++ TrackKeyPhrase ON
+
+      ++ PrevKeyPhrase <> %Filename%
+
+          FileDelete, %keyPhrase%
FileAppend, %Filename%, %keyPhrase%
+
+
quitNoSave:
ExitApp, JOB DONE. G'DAY!
.... END Quit SUBROUTINE .....
```



```
*****
```

< END OF PROGRAM >

/* vim: set noexpandtab shiftwidth=4: */

#

Threads

The *current thread* is defined as the flow of execution invoked by the most recent [hotkey](#), [timed subroutine](#), or [custom menu item](#). That thread can be executing commands within its own subroutine or within other subroutines called by that subroutine.

Although AutoHotkey doesn't actually use multiple threads, it simulates that behavior: If a second thread is started -- such as by pressing another hotkey while the previous is still running -- the *current thread* will be interrupted (temporarily halted) to allow the new thread to become *current*. If third thread is started while the second is still running, both the second and first will be in a dormant state, and so on.

When the *current thread* finishes, the one most recently interrupted will be resumed, and so on, until all the threads finally finish. When resumed, a thread's settings for things such as [ErrorLevel](#) and [SetKeyDelay](#) are automatically restored to what they were just prior to its interruption; in other words, a thread will experience no side-effects from having been interrupted (except for a possible change in the [active window](#)).

Note: The [KeyHistory](#) command/menu-item shows how many threads are in an interrupted state and the [ListHotkeys](#) command/menu-item shows which hotkeys have threads.

Related to the above: A single script can have multiple simultaneous [MsgBox](#), [InputBox](#), [FileSelectFile](#), and [FileSelectFolder](#) dialogs. This is achieved by launching a new thread (via [hotkey](#), [timed subroutine](#), or [custom menu item](#)) while a prior thread already has a dialog displayed.

By default, a given [hotkey](#) or [hotstring](#) subroutine cannot be run a second time if it is already running. Use [#MaxThreadsPerHotkey](#) to change this behavior.

Thread Priority

A [hotkey](#), [timed subroutine](#), or [custom menu item](#) with a priority lower than that of the *current thread* cannot interrupt it. During that time, such timers will not run, and pressing such a hotkey or selecting such a custom menu item will have no effect (nor will it be buffered). Because of this, it is usually best to design high priority threads to finish quickly, or avoid making them high priority.

The default priority is 0. All threads use the default priority unless changed by one of the following methods:

- 1) A timed subroutine is given a specific priority via [SetTimer](#).
- 2) A hotkey is given a specific priority via the [Hotkey](#) command.
- 3) A [hotstring](#) is given a specific priority when it is defined, or via the [#Hotstring](#) directive.
- 4) A custom menu item is given a specific priority via the [Menu](#) command.
- 5) The *current thread* sets its own priority via the [Thread](#) command.

The [OnExit](#) subroutine (if any) will always run when called for, regardless of the *current thread*'s priority.

#

ToolTip Mouse Menu (requires XP/2k/NT) — by Rajat

This script displays a popup menu in response to briefly holding down the middle mouse button. Select a menu item by left clicking it. Cancel the menu by left clicking outside of it. A recent improvement is that the contents of the menu can change depending on which type of window is active (Notepad and Word are used as examples here).

~~/* You can set any title here for the menu.~~

~~MenuTitle~~

~~/* This is how long the mouse button must be held to cause the menu to appear.~~

~~menuDelay 20~~

~~setFormat, float, 0.0~~

~~setBatchLines, 10ms~~

~~SetTitleMatchMode, 2~~

~~#SingleInstance~~

~~/* Menu Definitions~~

~~/* Create / Edit Menu Items here.~~

~~/* You can't use spaces in keys/values/section names.~~

~~/* Don't worry about the order, the menu will be sorted.~~

~~MenuItemList = Notepad/Calculator/Section 3/Section 4/Section 5~~

~~/* Dynamic menuitems here~~

~~/* Syntax:~~

~~/* Dyn# MenuItem|Window title~~

~~Dyn1 = MS Word| Microsoft Word~~

~~Dyn2 = Notepad II| Notepad~~

~~/*~~

~~Exit~~

~~/* Menu Sections~~

~~/* Create / Edit Menu Sections here.~~

~~Notepad:~~

~~Run, Notepad.exe~~

~~Return~~

~~Calculator:~~

~~Run, Calc~~

```
Return

Section3:
MsgBox, You selected 3
Return

Section4:
MsgBox, You selected 4
Return

Section5:
MsgBox, You selected 5
Return

Howword:
msgbox, this is a dynamic entry (word)
Return

NotepadII:
msgbox, this is a dynamic entry (notepad)
Return
```

```
Hotkey Section

---


MButton::
HowLong 0
Loop
+
    HowLong ++
    Sleep, 10
    GetKeyState, MButton, MButton, P
    IfEqual, MButton, 0, Break
+
    IfLess, HowLong, %UMDelay%, Return
```

```
prepares dynamic menu
DynMenu
Loop
+
    IfEqual, Dyn%a_index%, Break

    StringGetPos, ppos, dyn%a_index%,_
    StringLeft, item, dyn%a_index%, %ppos%
    ppos += 2
    StringMid, win, dyn%a_index%, %ppos%, 1000

    IfWinActive, win%
    DynMenu = %DynMenu%/%item%
```

```
+  
  
    Joins sorted main menu and dynamic menu  
Sort, MenuItem*, D/  
TempMenu = @MenuItem**DynMenu*  
  
    Clears earlier entries  
Loop  
+  
    #IfEqual, MenuItem@_index%, , Break  
    MenuItem@_index%  
+  
    Creates new entries  
Loop, Parse, TempMenu, /  
+  
    MenuItem@_index% = a_loopefield%  
+  
    Creates the menu  
Menu = @MenuItem*  
Loop  
+  
    #IfEqual, MenuItem@_index%, , Break  
    numItems++  
    StringTrimLeft, MenuText, MenuItem@_index%, 0  
    Menu = @Menu@`n@MenuText%  
+  
MouseGetPos, mx, my  
HotKey, LButton, MenuClick  
HotKey, LButton, On  
ToolTip, @Menu%, @mx%, @my%  
WinActivate, @MenuItem%  
Return  
  
MenuClick:  
HotKey, -LButton, Off  
#IfWinNotActive, @MenuItem%  
+  
    ToolTip  
    Return  
+  
MouseGetPos, mx, my  
ToolTip  
my = 3           ;space after which first line starts  
my / 12          ;space taken by each line
```

```
#fless, mY, 1, Return  
#fGreater, mY, %numItems%, Return  
StringTrimLeft, TargetSection, MenuItem%mY%, 0  
StringReplace, TargetSection, TargetSection, %a_space%, ,  
Eesub, %TargetSection%  
Return
```

#

AutoHotkey Tutorial—Launch a program or document (continued)

To have a program or document start off maximized, minimized, or hidden, consider the following hotkey subroutine, which assigns the Win+Z hotkey to launch two instances of Notepad, the 1st maximized and the 2nd minimized:

```
#z::  
Run, Notepad, , max  
Run, Notepad, , min  
return
```

To have a program use a specific folder as its working directory, consider this Win+C hotkey which creates a command prompt window in the specified directory:

```
#c::Run, %comspec% /k, C:\My Documents
```

In the above example, the %comspec% environment variable might resolve to C:\Windows\system32\cmd.exe on a typical system.

To pass parameters, add them immediately after the name of the program or document as in these examples:

```
Run, %comspec% /k dir, C:\My Documents  
Run, Notepad.exe "C:\My Documents\Address List.txt"  
Run, %ProgramFiles%\AutoHotkey\AutoHotkey.exe "C:\Scripts\Test Script.ahk" param1 "param2 with spaces" param3
```

In the 2nd and 3rd examples, parameters with spaces in them are enclosed in double quotes, which as a rule is the safest practice. By contrast, the working directory should not be enclosed in double quotes even if it contains spaces, such as in the 1st example above.

Certain special words known as *system verbs* are also supported. The first example below opens the Explorer's properties dialog for the indicated file. The second example prints the specified document:

```
Run, properties "C:\Address List.txt"  
Run, print C:\Address List.txt
```

RunWait sets the built-in ErrorLevel variable to the exit code of the program it ran. For example, the following will display a non-zero ErrorLevel because the comspec program is indicating that a problem occurred:

```
RunWait, %comspec% /c dir c:\NonExistent.txt, , hide  
MsgBox, %ErrorLevel%
```

[Return to the Tutorial Table of Contents](#)

#

Volume On-Screen Display (OSD) — by Rajat

This script assigns hotkeys of your choice to raise and lower the master and/or wave volume. Both volumes are displayed as different color bar graphs.

[Download This Script](#) | [Other Sample Scripts](#) | [Home](#)

User Settings

~~Make customisation only in this area or hotkey area only!!~~

~~The percentage by which to raise or lower the volume each time.~~

~~vol_Step = 4~~

~~How long to display the volume level bar graphs.~~

~~vol_DisplayTime = 2000~~

~~Master Volume Bar color (see the help file to use more precise shades).~~

~~vol_CDM = Red~~

~~Wave Volume Bar color~~

~~vol_CBW = Blue~~

~~Background color~~

~~vol_CW = Silver~~

~~Bar's screen position. Use 1 to center the bar in that dimension.~~

~~vol_PosX = 1~~

~~vol_PosY = 1~~

~~vol_Width = 150 ; width of bar~~

~~vol_Thick = 12 ; thickness of bar~~

~~If your keyboard has multimedia buttons for Volume, you can try changing the below hotkeys to use them by specifying Volume_Up, ^Volume_Up, Volume_Down, and ^Volume_Down.~~

~~HotKey, #Up, vol_MasterUp ; Win+UpArrow~~

~~HotKey, #Down, vol_MasterDown~~

~~HotKey, !#Up, vol_WaveUp ; Shift+Win+UpArrow~~

~~HotKey, !#Down, vol_WaveDown~~

Auto Execute Section

~~DON'T CHANGE ANYTHING HERE (unless you know what you're doing).~~

~~vol_BarOptionsMaster = 1:B ZH%vol_Thickness% ZY0 ZY0 W%vol_Width% CB%vol_CDM% CW%vol_CW%~~

~~vol_BarOptionsWave = 2:B ZH%vol_Thickness% ZY0 ZY0 W%vol_Width% CB%vol_CBW% CW%vol_CW%~~

~~If the X position has been specified, add it to the options.~~

~~Otherwise, omit it to center the bar horizontally.~~

~~If vol_PosX > 0~~

~~vol_BarOptionsMaster = %vol_BarOptionsMaster% X%vol_PosX%~~

~~vol_BarOptionsWave = %vol_BarOptionsWave% X%vol_PosX%~~

```
+  
+ If the Y position has been specified, add it to the options.  
+ Otherwise, omit it to have it calculated later.  
if vol_PosY >= 0  
+  
    vol_BarOptionsMaster = vol_BarOptionsMaster || vol_PosY  
    vol_PosY_wave = vol_PosY  
    vol_PosY_wave + vol_Thickness  
    vol_BarOptionsWave = vol_BarOptionsWave || vol_PosY_wave  
+  
#SingleInstance  
SetBatchLines, 10ms  
Return  
  
=====  
  
vol_WaveUp:  
soundSet, !vol_Step!, Wave  
Gosub, vol_ShowBars  
Return  
  
vol_WaveDown:  
soundSet, !vol_Step!, Wave  
Gosub, vol_ShowBars  
Return  
  
vol_MasterUp:  
soundSet, !vol_Step!  
Gosub, vol_ShowBars  
Return  
  
vol_MasterDown:  
soundSet, !vol_Step!  
Gosub, vol_ShowBars  
Return  
  
vol_ShowBars:  
+ To prevent the "flashing" effect, only create the bar window if it  
+ doesn't already exist:  
++WinNotExist, vol_Wave  
    Progress, !vol_BarOptionsWave!, , , vol_Wave  
++WinNotExist, vol_Master  
+  
+ Calculate position here in case screen resolution changes while  
+ the script is running:  
if vol_PosY < 0  
+  
    + Create the Wave bar just above the Master bar:  
    vol_PosY = vol_PosY - vol_Thickness
```

```
    winSetPos, , vol_Wave_Posy, , , vol_Wave
    vol_Wave_Posy = &vol_Thick%
    Progress, &vol_BarOptionsMaster% Yvol_Wave_Posy%, , , vol_Master
+
else
    Progress, &vol_BarOptionsMaster%, , , vol_Master
+
/* Get both volumes in case the user or an external program changed them.
SoundSet, vol_Master, Master
SoundSet, vol_Wave, Wave
Progress, 1:vol_Master%
Progress, 2:vol_Wave%
SetTimer, vol_BarOff, &vol_DisplayTime%
return

vol_BarOff:
SetTimer, vol_BarOff, off
Progress, 1:off
Progress, 2:off
return
```

#

Automating Winamp

This section demonstrates how to control Winamp via hotkey even when it is minimized or inactive. This information has been tested with Winamp 2.78c and should work for Winamp 2.x and possibly other major releases as well. Please post changes and improvements in the forum or contact the author.

This example makes the Ctrl+Alt+P hotkey equivalent to pressing Winamp's pause/unpause button:

```
^p..  
SetTitleMatchMode, 2  
IfWinNotExist, Winamp  
    IfWinNotExist, Winamp 2. , Adjust this if your Winamp is not 2.x  
        return  
    ; Otherwise, the above has set the "last found" window for us.  
ControlSend, ahk_parent, c , Pause/Unpause  
return
```

Here are some of the keyboard shortcuts available in Winamp 2.x (may work in other versions too). The above example can be revised to use any of these keys:

Key to send	Effect
e	Pause/UnPause
x	Play/Restart/UnPause
v	Stop
^v	Stop with Fadeout
^v	Stop after the current track
b	Next Track
z	Previous Track
{left}	Rewind 5 seconds
{right}	Fast forward 5 seconds
{up}	Turn Volume Up
{down}	Turn Volume Down

-

→ This next example asks Winamp which track number is currently active.

```
SetTitleMatchMode, 2  
SendMessage, 1024, 0, 120, Winamp  
If ErrorLevel <> FAIL  
+
```

ErrorLevel1, Winamp's count starts at 0, so adjust by 1.
MsgBox, Track %ErrorLevel% is active or playing.

+

#

Window Shading (roll up a window to its title bar) -- by Rajat

This script reduces a window to its title bar and then back to its original size by pressing a single hotkey. Any number of windows can be reduced in this fashion (the script remembers each). If the script exits for any reason, all "rolled up" windows will be automatically restored to their original heights.

[Download This Script](#) | [Other Sample Scripts](#) | [Home](#)

```
;  
; Set the height of a rolled up window here. The operating system  
; probably won't allow the title bar to be hidden regardless of  
; how low this number is.  
ws_MinHeight 25  
  
;  
; This line will unroll any rolled up windows if the script exits  
; for any reason.  
onExit, ExitSub  
return ; End of auto execute section  
  
+:: ; Change this line to pick a different hotkey.  
; Below this point, no changes should be made unless you want to  
; alter the script's basic functionality.  
; Uncomment this next line if this subroutine is to be converted  
; into a custom menu item rather than a hotkey. The delay allows  
; the active window that was deactivated by the displayed menu to  
; become active again.  
;sleep, 200  
WinGet, ws_ID, ID,  
Loop, Parse, ws_IDList,  
+  
    ifEqual, A_LoopField, %ws_ID%  
    +  
        ; Match found, so this window should be restored (unrolled).  
        StringTrimRight, ws_Height, ws_Window%ws_ID%, 0  
        WinMove, ahk_id %ws_ID%,,,,, %ws_Height%  
        StringReplace, ws_IDList, ws_IDList, |%ws_ID%  
        return  
    +  
WinGetPos, , ws_Height, A  
ws_Window%ws_ID% = %ws_Height%  
WinMove, ahk_id %ws_ID%,,,,, %ws_MinHeight%  
ws_IDList = %ws_IDList%|%ws_ID%
```

```
return

ExitSub:
Loop, Parse, ws_IDList, +
+
    if A_LoopField = ; First field in list is normally blank.
        continue ; So skip it.
    StringTrimRight, ws_Height, ws_Window%A_LoopField%, 0
    WinMove, ahk_id %A_LoopField%,,,, %ws_Height%
+
ExitApp ; Must do this for the OnExit subroutine to actually Exit the script.
```