



AutoHotkey

Automation. Hotkeys. Scripting.

Version 1.1.28.02

<https://autohotkey.com>

©2003-2014 Chris Mallett, portions ©[AutoIt Team](#) and the AHK community

Software License: [GNU General Public License](#)

Quick Reference

Getting started:

- [How to use the program](#)
- [Tutorial for beginners](#)
- [Text editors with AutoHotkey support](#)
- [Frequently asked questions](#)

Scripts:

- [Concepts and conventions](#): explanations of various things you need to know.
- [Scripting language](#): how to write scripts.
- [Miscellaneous topics](#)
- [Command and function list](#)
- [Variables and expressions](#)
- [Functions](#)
- [Objects](#)
- [Interactive debugging](#)

Keyboard and mouse:

- [Hotkeys \(mouse, joystick and keyboard shortcuts\)](#)
- [Hotstrings and auto-replace](#)
- [Remapping keys and buttons](#)
- [List of keys, mouse buttons and joystick controls](#)

Other:

- [DllCall](#)
- [RegEx quick reference](#)
- [Script compatibility](#) - AutoHotkey 1.0 users read this

Acknowledgements

A special thanks to Jonathan Bennett, whose generosity in releasing AutoIt v2 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 AutoIt v2 command set, as well as the Window Spy and the old script compiler, were adapted directly from the AutoIt v3 source code. So thanks to Jon and the other AutoIt authors for those as well.

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

~ Chris Mallett

Using the Program

AutoHotkey doesn't do anything on its own; it needs a script to tell it what to do. A script is simply a plain text file with the `.ahk` filename extension containing instructions for the program, like a configuration file, but much more powerful. A script can do as little as performing a single action and then exiting, but most scripts define a number of [hotkeys](#), with each hotkey followed by one or more actions to take when the hotkey is pressed.

```
#z::Run https://autohotkey.com ; Win+Z

^!n:: ; Ctrl+Alt+N
if WinExist("Untitled - Notepad")
    WinActivate
else
    Run Notepad
return
```

Tip: If your browser supports it, you can download any code block (such as the one above) as a script file by clicking the button which appears in the top-right of the code block when you hover your mouse over it.

Table of Contents

- [Create a Script](#)
- [Edit a Script](#)
- [Run a Script](#)
- [Tray Icon](#)
- [Main Window](#)
- [Command Line Usage](#)
- [Portability of AutoHotkey.exe](#)
- [Installer Options](#)
 - [Run with UI Access](#)

Create a Script

There are a couple of common ways to create a script file:

- In Notepad (or a [text editor](#) of your choice), save a file with the `.ahk` filename extension. On some systems you may need to enclose the name in quotes to ensure the editor does not add another extension (such as `.txt`).

Be sure to save the file as UTF-8 with BOM if it will contain non-ASCII characters. For details, see the [FAQ](#).

- In Explorer, right-click in empty space in the folder where you want to save the script, then select **New** and **AutoHotkey Script**. You can then type a name for the script (taking care not to erase the `.ahk` extension if it is visible).

See [Scripting Language](#) for details about how to write a script.

Edit a Script

To open a script for editing, right-click on the script file and select **Edit Script**. If the script is already running, you can use the [Edit](#) command or right-click the script's [tray icon](#) and select **Edit This Script**. By default this will open Notepad, but that can be changed by writing to the registry as shown [here](#). Of course, you can always open your text editor first and then open the script as you would any other text file.

After editing a script, you must run or [reload](#) the script for the changes to take effect. A running script can usually be reloaded via its [tray menu](#).

Run a Script

With AutoHotkey installed, there are several ways to run a script:

- Double-click a script file (or shortcut to a script file) in Explorer.
- Call AutoHotkey.exe on the command line and pass the script's filename as a [command-line parameter](#).
- After creating [the default script](#), launch AutoHotkey via the shortcut in the Start menu to run it.
- If AutoHotkey is pinned to the taskbar or Start menu on Windows 7 or later, recent or pinned scripts can be launched via the program's Jump List.

Most scripts have an effect only while they are running. Use the [tray menu](#) or the [ExitApp](#) command to exit a script. Scripts are also forced to exit when Windows shuts down. To configure a script to start automatically after the user logs in, the easiest way is to place a shortcut to the script file in the [Startup](#) folder.

Scripts can also be [compiled](#); that is, combined together with an AutoHotkey binary file to form a self-contained executable (.exe) file.

Tray Icon

By default, each script adds its own icon to the taskbar notification area (commonly known as the tray).

The tray icon usually looks like this (but the color or letter changes when the script is [paused](#) or [suspended](#)): 

Right-click the tray icon to show the tray menu, which has the following options by default:

- Open - Open the script's [main window](#).
- Help - Open the AutoHotkey offline help file.
- Window Spy - Displays various information about a window.
- Reload This Script - See [Reload](#).
- Edit This Script - See [Edit](#).
- Suspend Hotkeys - [Suspend](#) or unsuspend hotkeys.
- Pause Script - [Pause](#) or unpause the script.
- Exit - Exit the script.

By default, double-clicking the tray icon shows the script's [main window](#).

The [Menu](#) command can be used to customise the tray icon and menu.

The [#NoTrayIcon](#) directive can be used to hide the tray icon.

Main Window

The script's main window is usually hidden, but can be shown via the [tray icon](#) or one of the commands listed below to gain access to information useful for debugging the script. Items under the **View** menu control what the main window displays:

- Lines most recently executed - See [ListLines](#).
- Variables and their contents - See [ListVars](#).
- Hotkeys and their methods - See [ListHotkeys](#).
- Key history and script info - See [KeyHistory](#).

Known issue: Keyboard shortcuts for menu items do not work while the script is displaying a [MsgBox](#) or other dialog.

The built-in variable [A_ScriptHwnd](#) contains the unique ID (HWND) of the script's main window.

Closing this window with [WinClose](#) (even from another script) causes the script to exit, but most other methods just hide the window and leave the script running.

Command Line Usage

See [Passing Command Line Parameters to a Script](#) for command line usage, including a list of command line switches which affect the program's behavior.

Portability of AutoHotkey.exe

The file AutoHotkey.exe is all that is needed to launch any .ahk script.

[AHK_L 51+]: Renaming AutoHotkey.exe also changes which script it runs [by default](#), which can be an alternative to compiling a script for use on a computer without AutoHotkey installed. For instance, *MyScript.exe* automatically runs *MyScript.ahk* if a filename is not supplied, but is also capable of running other scripts.

Installer Options

To silently install AutoHotkey into the default directory (which is the same directory displayed by non-silent mode), pass the parameter `/S` to the installer. For example:

```
AutoHotkey110800_Install.exe /S
```

A directory other than the default may be specified via the `/D` parameter (in the absence of `/S`, this changes the default directory displayed by the installer). For example:

```
AutoHotkey110800_Install.exe /S /D=C:\Program  
Files\AutoHotkey
```

Version: If AutoHotkey was previously installed, the installer automatically detects which version of AutoHotkey.exe to set as the default. Otherwise, the default is Unicode 32-bit or Unicode 64-bit depending on whether the OS is 64-bit. To override which version of AutoHotkey.exe is set as the default, pass one of the following switches:

- `/A32` or `/ANSI` : ANSI 32-bit.
- `/U64` or `/x64` : Unicode 64-bit (only valid on 64-bit systems).
- `/U32` : Unicode 32-bit.

For example, the following installs silently and sets ANSI 32-bit as the default:

```
AutoHotkey110800_Install.exe /S /A32
```

Uninstall: To silently uninstall AutoHotkey, pass the `/Uninstall` parameter to `Installer.ahk`. For example:

```
"C:\Program Files\AutoHotkey\AutoHotkey.exe"  
"C:\Program Files\AutoHotkey\Installer.ahk"
```

```
/Uninstall
```

For AutoHotkey versions older than 1.1.08.00, use `uninst.exe /S`. For example:

```
"C:\Program Files\AutoHotkey\uninst.exe" /S
```

Note: Installer.ahk must be run as admin to work correctly.

Extract: Later versions of the installer include a link in the bottom-right corner to extract setup files without installing. If this function is present, the `/E` switch can be used to invoke it from the command line. For example:

```
AutoHotkey110903_Install.exe /D=F:\AutoHotkey /E
```

Restart scripts [v1.1.19.02+]: In silent install/uninstall mode, running scripts are closed automatically, where necessary. Pass the `/R` switch to automatically reload these scripts using whichever EXE they were running on, **without** command line args. Setup will attempt to launch the scripts via Explorer, so they do not run as administrator if UAC is enabled.

Taskbar buttons [v1.1.08+]: On Windows 7 and later, taskbar buttons for multiple scripts are automatically grouped together or combined into one button by default. The *Separate taskbar buttons* option disables this by registering each AutoHotkey executable as a [host app \(IsHostApp\)](#).

[v1.1.24.02+]: For command-line installations, specify `/IsHostApp` or `/IsHostApp=1` to enable the option and `/IsHostApp=0` to disable it.

Run with UI Access [v1.1.24.02+]

The installer GUI has an option "Add 'Run with UI Access' to context menus". This context menu option provides a workaround for common [UAC-related issues](#) by allowing the script to automate administrative programs - without the script running as admin. To achieve this, the installer does the following:

- Copies AutoHotkeyA32.exe, AutoHotkeyU32.exe and (if present) AutoHotkeyU64.exe to AutoHotkey*_UIA.exe.
- Sets the [uiAccess attribute](#) in each UIA file's embedded manifest.
- Creates a self-signed digital certificate named "AutoHotkey" and signs each UIA file.
- Registers the context menu option to run the appropriate exe file.

If any these UIA files are present before installation, the installer will automatically update them even if the UI Access option is not enabled.

For command-line installations, specify `/uiAccess` or `/uiAccess=1` to enable the option and `/uiAccess=0` to disable it. By default, the installer will enable the option if UAC is enabled and the UI Access context menu option was present before installation.

Scripts which need to run other scripts with UI access can simply [Run](#) the appropriate UIA.exe file with the normal [command line parameters](#).

Known limitations:

- UIA is only effective if the file is in a trusted location; i.e. a Program Files sub-directory.
- UIA.exe files created on one computer cannot run on other computers without first installing the digital certificate which was used to sign them.
- UIA.exe files cannot be started via CreateProcess due to security restrictions. ShellExecute can be used instead. [Run](#) tries both.
- UIA.exe files cannot be modified, as it would invalidate the file's digital signature.
- Because UIA programs run at a different "integrity level" than other programs, they can only access objects registered by other UIA programs. For example, `ComObjActive("word.Application")` will fail because Word is not marked for UI Access.
- The script's own windows can't be automated by non-UIA programs/scripts for security reasons.
- Running a non-UIA script which uses a mouse hook (even as simple as `#InstallMouseHook`) may prevent all mouse hotkeys from working when the mouse is pointing at a window owned by a UIA script, even hotkeys implemented by the UIA script itself. A workaround is to ensure UIA scripts are loaded last.

For more details, see [Enable interaction with administrative programs](#) on the archive forum.

Concepts and Conventions

This document covers some general concepts and conventions used by AutoHotkey, with focus on explanation rather than code. The reader is not assumed to have any prior knowledge of scripting or programming, but should be prepared to learn new terminology.

For more specific details about syntax, see [Scripting Language](#).

Table of Contents

- Values
 - Strings
 - Numbers
 - Boolean
 - Nothing
 - Objects
 - Object Protocol
- Variables
 - Uninitialised Variables
 - Built-in Variables
 - Environment Variables
 - Caching
- Functions/Commands
- Control Flow
- Details
 - String Encoding
 - Pure Numbers
 - Names
 - Variable References vs Values
 - References to Objects

Values

A *value* is simply a piece of information within a program. For example, the name of a key to send or a program to run, the number of times a hotkey has been pressed, the title of a window to activate, or whatever else has some meaning within the program or script.

AutoHotkey supports these types of values:

- [Strings](#) (text)
- [Numbers](#) (integers and floating-point numbers)
- [Objects](#)

Some other related concepts:

- [Boolean](#)
- [Nothing](#)

Strings

A *string* is simply text. Each string is actually a sequence or *string* of characters, but can be treated as a single entity. The *length* of a string is the number of characters in the sequence, while the *position* of a character in the string is merely that character's sequential number. By convention in AutoHotkey, the first character is at position 1.

A string of digits (or any other supported [number format](#)) is automatically interpreted as a number when a math operation or comparison requires it.

How literal text should be written within the script depends on the context. For details, see [Legacy Syntax](#) and [Strings \(in expressions\)](#).

For a more detailed explanation of how strings work, see [String Encoding](#).

Numbers

AutoHotkey supports these number formats:

- Decimal integers, such as `123`, `00123` or `-1`.
- Hexadecimal integers, such as `0x7B`, `0x007B` or `-0x1`.
- Decimal floating-point numbers, such as `3.14159`.

Hexadecimal numbers must use the `0x` or `0X` prefix, except where noted in the documentation. This prefix must be written after the `+` or `-` sign, if present, and before any leading zeroes. For example, `0x001` is valid, but `000x1` is not.

Numbers written with a decimal point are always considered to be floating-point, even if the fractional part is zero. For example, `42` and `42.0` are usually interchangeable, but not always. Scientific notation is also recognized, but only if a decimal point is present (e.g. `1.0e4` and `-2.1E-4`).

The decimal separator is always a dot, even if the user's regional settings specify a comma.

When a number is converted to a string, it is formatted according to the current [integer or float format](#). Although the [SetFormat](#) command can be used to change the current format, it is usually better to use the [Format](#) function to format a string. Floating-point numbers can also be formatted by using the [Round](#) function.

For details about the range and accuracy of numeric values, see [Pure Numbers](#).

Boolean

A *boolean* value can be either *true* or *false*. Boolean values are used to represent anything that has exactly two possible states, such as the *truth* of an expression. For example, the expression `(x <= y)` is *true* when `x` has lesser or equal value to `y`. A boolean value could also represent *yes* or *no*, *on* or *off*, *down* or *up* (such as for [GetKeyState](#)) and so on.

AutoHotkey does not have a specific boolean type, so it uses the integer value `0` to represent false and `1` to represent true. When a value is required to be either true or false, a blank or zero value is considered false and all other values are considered true. (Objects are always considered true.)

The words `true` and `false` are [built-in variables](#) containing 1 and 0. They

can be used to make a script more readable.

Nothing

AutoHotkey does not have a value which uniquely represents *nothing*, *null*, *nil* or *undefined*, as seen in other languages. Instead, an empty string (a string of zero length) often has this meaning.

If a [variable](#) or parameter is said to be "empty" or "blank", that usually means an empty string (a string of zero length).

Objects

There are generally two ways of viewing objects:

- An object contains a group of values, allowing the group itself to be treated as one value. For example, an object could contain an *array* or sequence of items, or a set of related values, such as the X and Y coordinates of a position on the screen. Objects can be used to build complex structures by combining them with other objects.
- An object can represent a *thing*, a *service*, or something else, and can provide ways for the script to interact with this thing or service. For example, a *BankAccount* object might have properties such as the account number, current balance and the owner of the account, and methods to withdraw or deposit an amount.

The proper use of objects (and in particular, [classes](#)) can result in code which is *modular* and *reusable*. Modular code is usually easier to test, understand and maintain. For instance, one can improve or modify one section of code without having to know the details of other sections, and without having to make corresponding changes to those sections. Reusable code saves time, by avoiding the need to write and test code for the same or similar tasks over and over.

When you assign an object to a [variable](#), as in `myObj := {}`, what you store is not the object itself, but a [reference](#) to the object. Copying that variable, as in `yourObj := myObj`, creates a new reference to the *same* object. A change such as `myObj.ans := 42` would be reflected by both `myObj.ans` and `yourObj.ans`, since they both refer to the same object. However, `myObj := Object()` only affects the variable *myObj*, not the variable *yourObj*.

which still refers to the original object.

Object Protocol

This section builds on these concepts which are covered in later sections:
[variables](#), [functions](#)

Objects work through the principle of *message passing*. You don't know where an object's code or variables actually reside, so you must pass a message to the object, like "give me *foo*" or "go do *bar*", and rely on the object to respond to the message. Objects in AutoHotkey support the following basic messages:

- **Get** a value.
- **Set** a value, denoted by `:=`.
- **Call** a method, denoted by `()`.

Each message can optionally have one or more [parameters](#) (and, for **Set**, the value). Usually there is at least one parameter, and it is interpreted as the name of a property or method, a key or an array index, depending on the object and how you're using it. The parameters of a message are specified using three different patterns: `.Name`, `[Parameters]` and `(Parameters)`, where *Name* is a literal [name or identifier](#), and *Parameters* is a list of parameters (as sub-expressions), which can be empty/blank (`[]` or `()`).

For **Get** and **Set**, `.Name` and `[Parameters]` can be used interchangeably, or in combination:

```
myObj[arg1, arg2, ..., argN]
myObj.name
myObj.name[arg2, ..., argN]
```

For **Call**, `.Name` and `[Parameter]` can be used interchangeably, and must always be followed by `(Parameters)`:

```
myObj.name(arg2, ..., argN)
myObj[arg1](arg2, ..., argN)
```

Notice that if `name` is present, it becomes the first parameter. `myObj.name` is equivalent to `myObj["name"]`, while `myObj.123` is equivalent to `myObj[123]`. This is true for every type of object, so it is always possible to compute the name of a property or method at runtime, rather than hard-coding it into the script.

Although *name* or *arg1* is considered to be the first parameter, remember that these are just *messages*, and the object is free to handle them in any way. In a method call such as those shown above, usually the object uses *name* or *arg1* to identify which method should be called, and then only *arg2* and beyond are passed to the method. In effect, *arg2* becomes the method's first apparent parameter.

Generally, **Set** has the same meaning as an assignment, so it uses the same operator:

```
myObj[arg1, arg2, ..., argN] := value
myObj.name := value
myObj.name[arg2, ..., argN] := value
```

Currently there is also a "hybrid" syntax allowed with **Set**, but it is best not to use it:

```
myObj.name(arg2, ..., argN) := value
```

Technically, `value` is passed as the last the parameter of the *Set* message; however, this detail is almost never relevant to script authors. Generally one can simply think of it as "the value being assigned".

Variables

A variable allows you to use a name as a placeholder for a value. Which value that is could change repeatedly during the time your script is running. For example, a hotkey could use a variable `press_count` to count the number of times it is pressed, and send a different key whenever `press_count` is a multiple of 3 (every third press). Even a variable which is only assigned a value once can be useful. For example, a `WebBrowserTitle` variable could be used to make your code easier to update when and if you were to change your preferred web browser, or if the `title` or `window class` changes due to a software update.

In AutoHotkey, variables are created simply by using them. Each variable is *not* permanently restricted to a single [data type](#), but can instead hold a value of any type: string, number or object. Each variable starts off empty/blank; in other words, each newly created variable contains an empty string until it is assigned some other value.

A variable has three main aspects:

- The variable's *name*.
- The variable itself.
- The variable's *value*.

Certain restrictions apply to variable names - see [Names](#) for details. In short, it is safest to stick to names consisting of ASCII letters (which are case insensitive), digits and underscore, and to avoid using names that start with a digit.

A variable name has *scope*, which defines where in the code that name can be used to refer to that particular variable; in other words, where the variable is *visible*. If a variable is not visible within a given scope, the same name can refer to a different variable. Both variables might exist at the same time, but only one is visible to each part of the script. [Global variables](#) are visible in the "global scope" (that is, outside of functions), but must usually be [declared](#) to make them visible inside a function. [Local variables](#) are visible only inside the function which created them.

A variable can be thought of as a container or storage location for a value, so

you'll often find the documentation refers to a variable's value as *the contents of the variable*. For a variable `x := 42`, we can also say that the variable `x` has the number 42 as its value, or that the value of `x` is 42.

It is important to note that a variable and its value are not the same thing. For instance, we might say "`myArray` is an array", but what we really mean is that `myArray` is a variable containing a reference to an array. We're taking a shortcut by using the name of the variable to refer to its value, but "`myArray`" is really just the name of the variable; the array object doesn't know that it has a name, and could be referred to by many different variables (and therefore many names).

Uninitialised Variables

To *initialise* a variable is to assign it a starting value. Although the program automatically initialises all variables (an empty string being the default value), it is good practice for a script to always initialise its variables before use. That way, anyone reading the script can see what variables the script will be using and what starting values they are expected to have.

It is usually necessary for the script to initialise any variable which is expected to hold a number. For example, `x := x + 1` will not work if `x` has never been assigned a value, since the *empty string* is considered to be non-numeric. The script should have assigned a starting value, such as `x := 0`. There are some cases where empty values *are* assumed to be 0, but it is best not to rely on this.

Script authors can use the `#Warn` directive to help find instances where a variable is used without having been initialised by the script.

Built-in Variables

A number of useful variables are built into the program and can be referenced by any script. With the exception of `Clipboard`, `ErrorLevel`, and `command line parameters`, these variables are read-only; that is, their contents cannot be directly altered by the script. By convention, most of these variables start with the prefix `A_`, so it is best to avoid using this prefix for your own variables.

Some variables such as `A_KeyDelay` and `A_TitleMatchMode` represent settings that control the script's behavior, and retain separate values for each `thread`. This

allows subroutines launched by new threads (such as for hotkeys, menus, timers and such) to change settings without affecting other threads.

Some special variables are not updated periodically, but rather their value is retrieved or calculated whenever the script references the variable. For example, [Clipboard](#) retrieves the current contents of the clipboard as text, and [A_TimeSinceThisHotkey](#) calculates the number of milliseconds that have elapsed since the hotkey was pressed.

Related: [list of built-in variables](#).

Environment Variables

Environment variables are maintained by the operating system. You can see a list of them at the command prompt by typing SET then pressing Enter.

A script may create a new environment variable or change the contents of an existing one with [EnvSet](#). Such additions and changes are not seen by the rest of the system. However, any programs or scripts which the script launches by calling [Run](#) or [RunWait](#) usually inherit a copy of the parent script's environment variables.

It is recommended that all new scripts retrieve environment variables such as Path via [EnvGet](#):

```
EnvGet, OutputVar, Path ; For explanation, see #NoEnv.
```

If a script lacks the [#NoEnv](#) directive, reading an empty variable will instead return the value of the environment variable with that name, if there is one. This can cause confusion, so it is recommended that all new scripts use [#NoEnv](#).

Caching

Although a variable is typically thought of as holding a single value, and that value having a distinct type (string, number or object), AutoHotkey automatically converts between numbers and strings in cases like `myString + 1` and `MsgBox %myNumber%`. As these conversions can happen very

frequently, whenever a variable is converted, the result is *cached* in the variable.

In effect, a variable can contain both a string and a number simultaneously. Usually this just improves the script's performance with no down-sides, but if a variable contains both a number and a string, is it number, or is it a string? This ambiguity causes unexpected behavior in at least two cases:

1. COM objects. In order to pass parameters to a COM object, the program must convert the variable's content to either a number *or* a string. Some COM objects throw an exception if the wrong type of value is passed. If a variable has both, the number is used. Usually this gets the right result, but sometimes it doesn't.
2. Objects don't have the capability to store both a number and a string as a key or value. Since numbers are more memory-efficient, if a variable has both, the number is used (except for floating-point values used as keys).

[SetFormat](#)'s slow mode forces the assignment of a pure number to immediately convert that number to a string. For integers, the number is also stored, so this doesn't have adverse effects other than to performance. For floats, the number is not stored, since SetFormat affects the precision of the value, possibly even truncating all decimal places. In other words, SetFormat's slow mode prevents pure floats from being stored in variables.

Taking the address of a variable effectively converts the variable's value to a string, disabling caching until the variable's address changes (this happens when its capacity changes). This is both for backward-compatibility and because the script could change the value indirectly via its address at any time, making the cache inaccurate.

Related

- [Variables](#): basic usage and examples.
- [Variable Capacity and Memory](#): details about limitations.

Functions/Commands

A *function* or *command* is the basic means by which a script *does something*.

In essence, functions and commands are the same thing, so the concepts explained here apply to both. However, the long history of AutoHotkey v1 and an emphasis on backward-compatibility have resulted in a divide between *commands*, which require legacy syntax, and *functions*, which require expression syntax.

Commands and functions can have many different purposes. Some functions might do no more than perform a simple calculation, while others have immediately visible effects, such as moving a window. One of AutoHotkey's strengths is the ease with which scripts can automate other programs and perform many other common tasks by simply calling a few functions. See the [command and function list](#) for examples.

Throughout this documentation, some common words are used in ways that might not be obvious to someone without prior experience. Below are several such words/phrases which are used frequently in relation to functions and commands:

Call a function or command

Calling a function or command causes the program to invoke, execute or evaluate it. In other words, a *function call* temporarily transfers control from the script to the function. When the function has completed its purpose, it *returns* control to the script. In other words, any code following the function call does not execute until after the function completes.

However, sometimes a function or command completes before its effects can be seen by the user. For example, the [Send](#) command *sends* keystrokes, but may return before the keystrokes reach their destination and cause their intended effect.

Parameters

Usually a command or function accepts *parameters* which tell it how to

operate or what to operate on. Each parameter is a **value**, such as a string or number. For example, **WinMove** moves a window, so its parameters tell it which window to move and where to move it to. Parameters can also be called *arguments*. Common abbreviations include *param* and *arg*.

Pass parameters

Parameters are *passed* to a function or command, meaning that a value is specified for each parameter of the function or command when it is called. For example, one can *pass* the name of a key to **GetKeyState** to determine whether that key is being held down.

Return a value

Functions *return* a value, so the result of the function is often called a *return value*. For example, **StrLen** returns the number of characters in a string. Commands do not return a result directly; instead, they store the result in a **variable**. Functions may also do this, such as when there is more than one result.

Functions and commands usually expect parameters to be written in a specific order, so the meaning of each parameter value depends on its position in the comma-delimited list of parameters. Some parameters can be omitted, in which case the parameter can be left blank, but the comma following it can only be omitted if all remaining parameters are also omitted. For example, the syntax for **ControlSend** is:

```
ControlSend [, Control, Keys, WinTitle, WinText,  
ExcludeTitle, ExcludeText]
```

Square brackets signify that the enclosed parameters are optional (the brackets themselves should not appear in the actual code). However, **ControlSend** isn't useful unless *Keys* are specified, and usually one must also specify the target window. For example:

```
ControlSend, Edit1, ^{Home}, A ; Correct.  
Control is specified.  
ControlSend, ^{Home}, A ; Incorrect:
```

```
Parameters are mismatched.  
ControlSend,, ^{Home}, A ; Correct.  
Control is omitted.
```

Methods

Methods are functions which operate on a particular [object](#). While there can only be one function named `Send` (for example), there can be as many methods named `Send` as there are objects, as each object (or class of objects) can respond in a different way. For this reason, the target object (which may be a variable or sub-expression) is specified to left of the method name instead of inside the parameter list. For details, see [Object Protocol](#).

Control Flow

Control flow is the order in which individual statements are executed. Normally statements are executed sequentially from top to bottom, but a control flow statement can override this, such as by specifying that statements should be executed repeatedly, or only if a certain condition is met.

Statement

A *statement* is simply the smallest standalone element of the language that expresses some action to be carried out. In AutoHotkey, statements include commands, assignments, function calls and other expressions. However, directives, labels (including hotkeys and hotstrings), and declarations without assignments are not statements; they are processed when the program first starts up, before the script *executes*.

Execute

Carry out, perform, evaluate, put into effect, etc. *Execute* basically has the same meaning as in non-programming speak.

Body

The *body* of a control flow statement is the statement or group of statements to which it applies. For example, the body of an *if statement* is executed only if a specific condition is met.

For example, consider this simple set of instructions:

1. Open Notepad
2. Wait for Notepad to appear on the screen
3. Type "Hello, world!"

We take one step at a time, and when that step is finished, we move on to the next step. In the same way, control in a program or script usually flows from one statement to the next statement. But what if we want to type into an existing Notepad window? Consider this revised set of instructions:

1. If Notepad is not running:
 1. Open Notepad

2. Wait for Notepad to appear on the screen
2. Otherwise:
 1. Activate Notepad
3. Type "Hello, world!"

So we either open Notepad or activate Notepad depending on whether it is already running. #1 is a *conditional statement*, also known as an *if statement*; that is, we execute its *body* (#1.1 - #1.2) only if a condition is met. #2 is an *else statement*; we execute its body (#2.1) only if the condition of a previous *if statement* (#1) is not met. Depending on the condition, control *flows* one of two ways: #1 (if true) → #1.1 → #1.2 → #3; or #1 (if false) → #2 (else) → #2.1 → #3.

The instructions above can be translated into the code below:

```
if (not WinExist("ahk_class Notepad"))
{
    Run Notepad
    WinWait ahk_class Notepad
}
else
    WinActivate ahk_class Notepad
Send Hello`, world!
```

In our written instructions, we used indentation and numbering to group the statements. Scripts work a little differently. Although indentation makes code easier to read, in AutoHotkey it does not affect the grouping of statements. Instead, statements are grouped by enclosing them in braces, as shown above. This is called a *block*.

For details about syntax - that is, how to write or recognise control flow statements in AutoHotkey - see [Control Flow](#).

Details

String Encoding

Each character in the string is represented by a number, called its *ordinal number*, or *character code*. For example, the value "Abc" would be represented as follows:

A	b	c
65	98	99

The *encoding* of a string is the way in which numbers are mapped to symbols. There are many different encodings, but as all of those supported by AutoHotkey include ASCII as a subset, character codes 0 to 127 always have the same meaning. For example, 'A' always has the character code 65.

Although AutoHotkey provides ways to work with text in various encodings, the built-in commands and functions--and to some degree the language itself--all assume string values to be in one particular encoding. This is referred to as the *native* encoding. The native encoding depends on the version of AutoHotkey:

- Unicode versions of AutoHotkey use UTF-16. The smallest element in a UTF-16 string is two bytes (16 bits). Unicode characters in the range 0 to 65535 (U+FFFF) are represented by a single 16-bit code unit of the same value, while characters in the range 65536 (U+10000) to 1114111 (U+10FFFF) are represented by a *surrogate pair*; that is, exactly two 16-bit code units between 0xD800 and 0xDFFF. (For further explanation of surrogate pairs and methods of encoding or decoding them, search the Internet.)
- ANSI versions of AutoHotkey use the system default ANSI code page, which depends on the system locale or "language for non-Unicode programs" system setting. The smallest element of an ANSI string is one byte. However, some code pages contain characters which are represented by sequences of multiple bytes (these are always non-ASCII characters).

Generally, other parts of this documentation use the term "character" to mean a string's smallest unit; bytes for ANSI strings and 16-bit code units for Unicode strings. For practical reasons, the length of a string and positions within a string are measured by counting these fixed-size units, even though they may not be complete Unicode characters.

[FileRead](#), [FileAppend](#), [FileOpen](#) and the [File object](#) provide ways of reading and writing text in files with a specific encoding.

The functions [StrGet](#) and [StrPut](#) can be used to convert strings between the native encoding and some other specified encoding. However, these are usually only useful in combination with data structures and the [DllCall](#) function. Strings which are passed directly to or from [DllCall](#) can be converted to ANSI or UTF-16 by using the `AStr` or `WStr` parameter types.

Techniques for dealing with the differences between ANSI and Unicode versions of AutoHotkey can be found under [Unicode vs ANSI](#).

Pure Numbers

A *pure* or *binary* number is one which is stored in memory in a format that the computer's CPU can directly work with, such as to perform math. In most cases, AutoHotkey automatically converts between numeric strings and pure numbers as needed, and rarely differentiates between the two types. AutoHotkey primarily uses two data types for pure numbers:

- 64-bit signed integers (*int64*).
- 64-bit binary floating-point numbers (the *double* or *binary64* format of the IEEE 754 international standard).

In other words, scripts are affected by the following limitations:

- Integers must be within the range -9223372036854775808 (-0x8000000000000000, or -2^{63}) to 9223372036854775807 (0x7FFFFFFFFFFFFFFF, or $2^{63}-1$). Although larger values can be contained within a string, any attempt to convert the string to a number (such as by using it in a math operation) might yield inconsistent results.
- Floating-point numbers generally support 15 digits of precision. However,

converting a floating-point number to a string causes the number to be rounded according to the current [float format](#), which defaults to 6 decimal places. If the "slow" mode of [SetFormat](#) is present anywhere in the script, numbers are *always* converted to strings when assigned to a [variable](#).

Note: There are some decimal fractions which the binary floating-point format cannot precisely represent, so a number is rounded to the closest representable number. This may lead to unexpected results. For example:

```
SetFormat FloatFast, 0.17 ; Show "over-full"
precision
MsgBox % 0.1 + 0          ; 0.100000000000000001
MsgBox % 0.1 + 0.2        ; 0.300000000000000004
MsgBox % 0.3 + 0          ; 0.299999999999999999
MsgBox % 0.1 + 0.2 = 0.3  ; 0 (not equal)
```

One strategy for dealing with this is to avoid direct comparison, instead comparing the difference. For example:

```
MsgBox % Abs((0.1 + 0.2) - (0.3)) <
0.000000000000000001
```

Another strategy is to always convert to string (thereby applying rounding) before comparison. There are generally two ways to do this while specifying the precision, and both are shown below:

```
MsgBox % Round(0.1 + 0.2, 15) = Format("
{:.15f}", 0.3)
```

Names

AutoHotkey uses the same set of rules for naming various things, including variables, functions, [window groups](#), [GUIs](#), classes and methods:

- **Case sensitivity:** None for ASCII characters. For example,

`CurrentDate` is the same as `currentdate`. However, uppercase non-ASCII characters such as 'Ä' are *not* considered equal to their lowercase counterparts, regardless of the current user's locale. This helps the script to behave consistently across multiple locales.

- **Maximum length:** 253 characters.
- **Allowed characters:** Letters, numbers, non-ASCII characters, and the following symbols: `_ # @ $`

Due to style conventions, it is generally better to name your variables using only letters, numbers, and the underscore character (for example: `CursorPosition`, `Total_Items`, and `entry_is_valid`). This style allows people familiar with other computer languages to understand your scripts more easily.

Although a [variable](#) name may consist entirely of digits, this is generally used only for [incoming command line parameters](#). Such numeric names cannot be used in expressions because they would be seen as numbers rather than variables. It is best to avoid starting a name with a digit, as such names are confusing and will be considered invalid in AutoHotkey v2.

As the following characters may be reserved for other purposes in AutoHotkey v2, it is recommended to avoid using them: `# @ $`

Property names in classes have the same rules and restrictions as variable names, except that the three characters listed above (`# @ $`) are not allowed. Although they can be used in a method definition, calling such a method requires using square brackets. For example, `myObject.@home()` is not valid, but `myObject["@home"]()` is acceptable.

Variable References vs Values

Variables have certain attributes that blur the line between a variable and its value, but it is important to make the distinction. In particular, consider [objects](#) and `ByRef` parameters.

Although we may say the variable `myArray` *contains* an array (which is a type of object), what the variable contains isn't the array itself but rather a *reference or pointer* to the array. Any number of variables can contain a reference to the same object. It may be helpful to think of a variable as just a name in that case. For instance, giving a person a nickname doesn't cause a clone of that person to

come into existence.

By default, variables are passed to user-defined functions *by value*. In other words, the value contained by the variable is copied into the variable which corresponds to the function's parameter. **ByRef** parameters allow you to pass a variable *by reference*, or in other words, to make one parameter of the function an *alias* for your variable, allowing the function to assign a new value to your variable.

Because a variable only ever contains a *reference* to an object and not the object itself, when you pass such a variable to a non-ByRef parameter, what the function receives is a reference to the same object. This allows the function to modify the object, but it does not allow the function to modify the *variable* which the function's caller used, because the function only has a reference to the object, not the variable.

References to Objects

Scripts interact with an object only indirectly, through a *reference* to the object. When you create an object, the object is created at some location you don't control, and you're given a reference. Passing this reference to a function or storing it in a variable or another object creates a new reference to the *same* object. You release a reference by simply using an assignment to replace it with any other value. An object is deleted only after all references have been released; you cannot delete an object explicitly, and should not try.

```
ref1 := Object() ; Create an object and store
first reference
ref2 := ref1     ; Create a new reference to
the same object
ref1 := ""      ; Release the first reference
ref2 := ""      ; Release the second
reference; object is deleted
```

If that's difficult to understand, try thinking of an object as a rental unit. When you rent a unit, you're given a key which you can use to access the unit. You can get more keys and use them to access the same unit, but when you're finished with the unit, you must hand all keys back to the rental agent. Usually a unit

wouldn't be *deleted*, but maybe the agent will have any junk you left behind removed; just as any values you stored in an object are freed when the object is deleted.

Scripting Language

An AutoHotkey script is basically a set of instructions for the program to follow, written in a custom language exclusive to AutoHotkey. This language bears some similarities to several other scripting languages, but also has its own unique strengths and pitfalls. This document describes the language and also tries to point out common pitfalls.

See [Concepts and Conventions](#) for more general explanation of various concepts utilised by AutoHotkey.

There are two distinct styles of syntax used in AutoHotkey: [legacy syntax](#) and [expressions](#).

Table of Contents

- General Conventions
- Comments
- Expressions
 - Strings / Text
 - Variables
 - Operators
 - Function Calls
 - Operators for Objects
 - Expression Statements
- Legacy Syntax
- Commands
 - OutputVar and InputVar Parameters
 - Text Parameters
 - Numeric Parameters
 - % Expression
 - Documentation Conventions
 - Optional Parameters
- Expressions vs Legacy Syntax
 - Different Equals
 - Commands vs Functions
- Control Flow Statements
 - Control Flow vs Commands
 - If Statement
 - Loop Statement
 - Not Control Flow
- Structure of a Script
 - Auto-execute Section
 - Subroutines
 - User-Defined Functions
 - #Include
- Miscellaneous
 - Dynamic Variables
 - Pseudo-arrays
 - Associative pseudo-arrays
 - Commands which create pseudo-arrays

- Labels

General Conventions

Names: Variable and function names are not case sensitive (for example, `CurrentDate` is the same as `currentdate`). For details such as maximum length and usable characters, see [Names](#).

No typed variables: Variables have no explicitly defined type; instead, a value of any type can be stored in any variable (excluding built-in variables). Numbers may be automatically converted to strings (text) and vice versa, depending on the situation.

Declarations are optional: Except where noted on the [functions page](#), variables do not need to be declared; they come into existence simply by using them (and each variable starts off empty/blank).

Spaces are mostly ignored: Indentation (leading space) is important for writing readable code, but is not required by the program and is generally ignored. Spaces and tabs are *generally* ignored at the end of a line, within an expression (except between quotes), and before and after command parameters. However, spaces are significant in some cases, including:

- [Function](#) and method calls require there to be no space between the function/method name and `(`.
- Spaces are required when performing concatenation.
- Spaces may be required between two operators, to remove ambiguity.
- Single-line [comments](#) require a leading space if they are not at the start of the line.

Line breaks are meaningful: Line breaks generally act as a statement separator, terminating the previous command or expression. (A *statement* is simply the smallest standalone element of the language that expresses some action to be carried out.) The exception to this is line continuation (see below).

Line continuation: Long lines can be divided up into a collection of smaller ones to improve readability and maintainability. This is achieved by preprocessing, so is not part of the language as such. There are two methods:

- [Line continuation](#), where lines that begin with an [expression operator](#)

(except ++ and --) are merged with the previous line. Lines are merged regardless of whether the line actually contains an expression.

- **Continuation sections**, where multiple lines are merged with the line above the section. The start and end of a continuation section are marked with (and) (both symbols must appear at the beginning of a line, excluding whitespace).

Comments

Comments are portions of text within the script which are ignored by the program. They are typically used to add explanation or disable parts of the code.

Scripts can be commented by using a semicolon at the beginning of a line. For example:

```
; This entire line is a comment.
```

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

```
Run 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).  
MsgBox, Common mistake: */ this does not end the  
comment.  
MsgBox, This line is commented out.  
*/
```

Since comments are ignored when a script is launched, they do not impact performance or memory utilization.

The default comment character (semicolon) can be changed to some other character or string via [#CommentFlag](#).

Expressions

Expressions are combinations of one or more [values](#), [variables](#), [operators](#) and [function calls](#). For example, `10`, `1+1` and `MyVar` are valid expressions. Usually, an expression takes one or more values as input, performs one or more operations, and produces a value as the result. The process of finding out the value of an expression is called *evaluation*. For example, the expression `1+1` *evaluates* to the number 2.

Commands are designed to take a list of parameters and perform only a single action per line, whereas simple expressions can be pieced together to form increasingly more complex expressions. For example, if `Discount/100` converts a discount percentage to a fraction, `1 - Discount/100` calculates a fraction representing the remaining amount, and `Price * (1 - Discount/100)` applies it to produce the net price.

Values are [numbers](#), [objects](#) or [strings](#). A *literal* value is one written physically in the script; one that you can see when you look at the code.

Strings / Text

For a more general explanation of strings, see [Strings](#).

A *string*, or *string of characters*, is just a text value. In an expression, literal text must be enclosed in quotation marks to differentiate it from a variable name or some other expression. This is often referred to as a *quoted literal string*, or just *quoted string*. For example, `"This is a quoted string."`.

To include an *actual* quote character inside a quoted string, specify two consecutive quotes as shown twice in this example: `"She said, ""An apple a day."""`.

Quoted strings can contain [escape sequences](#) such as `\t` (tab), `\n` (linefeed), and `\r` (carriage return). Unlike [unquoted text](#), it is not necessary to escape commas or percent signs, as quoted strings cannot contain variables. The use of the `\"` escape sequence to produce a literal quote-character is currently not

supported; instead, use two consecutive quotes as shown above.

Variables

For a basic explanation and general details about variables, see [Variables](#).

Variables can be used in an expression simply by writing the variable's name. For example, `A_ScreenWidth/2`. However, variables cannot be used inside a quoted string. Instead, variables and other values can be combined with text through a process called *concatenation*. There are two ways to *concatenate* values in an expression:

- Implicit concatenation: `"The value is " MyVar`
- Explicit concatenation: `"The value is " . MyVar`

Implicit concatenation is also known as *auto-concat*. In both cases, the spaces preceding the variable and dot are mandatory.

The [Format](#) function can also be used for this purpose. For example:

```
MsgBox % Format("You are using AutoHotkey v{1}
{2}-bit.", A_AhkVersion, A_PtrSize*8)
```

To assign a value to a variable, use the `:=` [assignment operator](#), as in `MyVar := "Some text"`.

Percent signs within an expression are used to create [dynamic variable references](#) and [dynamic function calls](#). Most of the time these constructs are not needed, so in general, variable names should not be enclosed in percent signs within an expression.

Operators

Operators take the form of a symbol or group of symbols such as `+` or `:=`, or one of the words `and`, `or`, `not` or `new`. They take one, two or three values as input and return a value as the result. A value or sub-expression used as input for an operator is called an *operand*.

- *Unary* operators are written either before or after a single operand, depending on the operator. For example, `-x` or `not keyIsDown`.
- *Binary* operators are written in between their two operands. For example, `1+1` or `2 * 5`.
- AutoHotkey has only one *ternary* operator, which takes the form `condition ? valueIfTrue : valueIfFalse`.

Some unary and binary operators share the same symbols, in which case the meaning of the operator depends on whether it is written before, after or in between two values. For example, `x-y` performs subtraction while `-x` inverts the sign of `x` (producing a positive value from a negative value and vice versa).

Operators of equal precedence such as multiply (`*`) and divide (`/`) are evaluated in left-to-right order unless otherwise specified in the [operator table](#). By contrast, an operator of lower precedence such as add (`+`) is evaluated after a higher one such as multiply (`*`). For example, `3 + 2 * 2` is evaluated as `3 + (2 * 2)`. Parentheses may be used to override precedence as in this example: `(3 + 2) * 2`

Function Calls

For a general explanation of functions and related terminology, see [Functions/Commands](#).

Functions take a varying number of inputs, perform some action or calculation, and then *return* a result. The inputs of a function are called *parameters* or *arguments*. A function is *called* simply by writing its name followed by its parameters enclosed in parentheses. For example,

`GetKeyState("Shift")` returns (evaluates to) 1 if the  being held down or 0 otherwise.

Note: There must not be any space between the function name and open parenthesis.

When compared to [commands](#), the requirement for parentheses may seem cryptic or verbose at first, but they are what allows a function call to be combined with other operations. For example, the expression

`GetKeyState("Shift", "P")` and `GetKeyState("Ctrl", "P")` returns 1 only if both keys are being physically held down.

Function names are always global, and are separate to variable names. For example, `Round` can be both a variable name and a function name, and `Round := 1` will not affect `Round(n)` in any way.

Operators for Objects

There are other symbols used in expressions which don't quite fit into any of the categories defined above, or that affect the meaning of other parts of the expression, as described below. These all relate to *objects* in some way. Providing a full explanation of what each construct does would require introducing more concepts which are outside the scope of this section.

`Alpha.Beta` is often called *member access*. *Alpha* is an ordinary variable, and could be replaced with a function call or some other sub-expression which returns an object. When evaluated, the object is sent a request "give me the value of property *Beta*", "store this value in property *Beta*" or "call the method named *Beta*". In other words, *Beta* is a name which has meaning to the object; it is not a local or global variable.

`Alpha.Beta()` is a *method call*, as described above.

`Alpha.Beta[Param]` is a specialised form of member access which includes additional parameters in the request. While *Beta* is a simple name, *Param* is an ordinary variable or sub-expression, or a list of sub-expressions separated by commas (the same as in a function's parameter list).

`Alpha[Index]` has a similar function to `Alpha.Beta`, but each part is interpreted in a more standard way. That is, both *Alpha* and *Index* are variables in this case, and could be replaced with virtually any sub-expression. This syntax is usually used to retrieve an element of an [array](#) or [associative array](#).

`new ClassName()` is used to instantiate a class, or create an object derived from another object. Although this looks like a function call, *ClassName* is actually an ordinary variable. Similarly, `new Alpha.Beta()` would create an object derived from the object returned by `Alpha.Beta`; *Beta* is neither a function nor a method. If the optional parentheses are present, they may contain

parameters for the object's `__New` method.

`[A, B, C]` creates an [array](#) with the initial contents A, B and C (all variables in this case), where A is element 1.

`{Key1: Value1, Key2: Value2}` creates an [associative array](#) from a list of key-value pairs. A value can later be retrieved by its associated key. Writing a plain word (consisting of alphanumeric characters, underscore and non-ASCII characters) to the left of `:` is equivalent to enclosing that word in quotation marks. For example, `{A: B}` is equivalent to `{"A": B}`. However, `{(A): B}` uses the contents of the variable `A` as the key.

`MyFunc(Params*)` is a [variadic function call](#). The asterisk must immediately precede the closing parenthesis at the end of the function's parameter list. *Params* must be a variable or sub-expression which returns an array object. Although it isn't valid to use `Params*` just anywhere, it can be used in an array literal (`[A, B, C, ArrayToAppend*]`) or indexer (`Alpha[Params*]`).

Expression Statements

Not all expressions can be used alone on a line. For example, a line consisting of just `21*2` or `"Some text"` wouldn't make any sense. An expression *statement* is an expression used on its own, typically for its side-effects. Most expressions with side-effects can be used this way, so it is generally not necessary to memorise the details of this section.

The following types of expressions can be used as statements:

Assignments, as in `x := y`, compound assignments such as `x += y`, and increment/decrement operators such as `++x` and `x--`. However, in AutoHotkey v1, `++`, `--`, `+=`, `-=`, `*=` and `/=` have slightly different behavior when used alone on a line, as they are actually equivalent to `EnvAdd`, `EnvSub`, `EnvMult` or `EnvDiv`. For details, see "Known limitations" under [Assign](#) in the table of operators.

Function calls such as `MyFunc(Params)`. However, a standalone function call cannot be followed by an open brace `{` (at the end of the line or on the next line), because it would be confused with a function declaration.

Method calls such as `MyObj.MyMethod()`.

Member access using square brackets, such as `MyObj[Index]`, which can have side-effects like a function call.

Expressions starting with the `new` operator, as in `new ClassName`, because sometimes a class can be instantiated just for its side-effects.

Ternary expressions such as `x ? CallIfTrue() : CallIfFalse()`. However, in AutoHotkey v1, command names take precedence. For example, `MsgBox ? 1 : 0` shows a message box.

Expressions starting with `(`. However, there usually must be a matching `)` on the same line, otherwise the line would be interpreted as the start of a [continuation section](#).

Expressions that start with any of those described above (but not those described below) are also allowed, for simplicity. For example, `MyFunc()+1` is currently allowed, although the `+1` has no effect and its result is discarded. Such expressions might become invalid in the future due to enhanced error-checking.

Member access using a dot (once or in a series), such as `ExcelApp.Quit` or `x.y.z`. However, unless parentheses are used (as in a method call), this cannot be the prefix of a larger expression. For example, `ExcelApp.Quit, xxx` is prohibited due to the apparent similarity to command syntax.

Legacy Syntax

Legacy or **command** syntax generally only allows a single action per line, but uses fewer characters to perform simple tasks such as [sending keystrokes](#) or [running a program](#). The syntax consists of command and variable names, *unquoted text* and a few symbols such as `,` `=` and `%`.

Unquoted text is simply text, not enclosed in quote marks, just straight up. Since the text has no explicit start and end marks, it ends at the end of the line or the end of the parameter. Leading and trailing spaces and tabs are ignored. Within unquoted text, the following characters have special meaning:

- `%`: Enclose a variable name in percent signs to include the contents of that variable. For example, `The year is %A_Year%`.

Note: Variable names are not *always* enclosed in percent signs; they are required only within unquoted text. Percent signs should not be used anywhere else, except to create a [dynamic variable reference](#) or [dynamic function call](#).

Note: Only a plain variable name can be used. [Array elements](#), [properties](#) and other [expressions](#) are not supported.

- `,`: Comma is used to delimit (separate) the parameters of a command, with [some exceptions](#). It has no special meaning when used in an assignment or comparison, so is interpreted literally in such cases.
- ```: An [escape character](#) is usually used to indicate that the character immediately following it should be interpreted differently than it normally would. For example, ``%`` produces a literal percent sign and ``,`` produces a literal comma. Some other common escape sequences produce special characters, such as ``t`` (tab), ``n`` (linefeed), and ``r`` (carriage return).

Commands accept a mixture of [unquoted text](#), [variable names](#) and [numeric expressions](#).

```
Send, The time is %A_Hour% o'clock.
```

[Legacy assignment](#) assigns [unquoted text](#) to a variable.

```
Clipboard = This text is copied to the  
clipboard.
```

[If statements](#) perform an action only if the specified condition is met.

```
If Var = Text value
```

There are also several other [control flow statements](#) (such as loops) which use legacy syntax similar to commands.

Commands

A *command* is an instruction to carry out a specific predefined action.

"Command" may also refer to a specific predefined action, such as [MsgBox](#). The set of available [commands](#) is predefined and cannot be changed by the script.

A command is *called* simply by writing its name at the beginning of a line, optionally followed by parameters. For example:

```
MsgBox, The time is %A_Hour% o'clock.
```

The comma separating the command name from its parameters is optional, except in the following cases:

- When it's necessary to prevent the line from being interpreted as a [legacy assignment](#) or [assignment expression](#).

```
MsgBox, := This would be an assignment  
without the comma.
```

- When the first parameter is blank.

```
MsgBox,, Second, Third
```

- When the command is alone at the top of a [continuation section](#).

Each parameter of a command may accept different syntax, depending on the command. There are four types of parameters:

- OutputVar
- InputVar
- Text
- Number

In most cases the [percent prefix](#) can be used to pass an expression.

OutputVar and InputVar Parameters

OutputVar and *InputVar* parameters require a variable name or [dynamic variable reference](#). For example:

```
; Replace all spaces with pluses:  
StringReplace, NewStr, OldStr, %A_Space%, +, All
```

This command reads the value from *OldStr* (the *InputVar*) and stores the result in *NewStr* (the *OutputVar*).

Note: Only a plain variable can be used as an *OutputVar*. Array elements, properties and other expressions are not supported.

InputVar parameters can accept an expression only when the [percent prefix](#) is used. However, the prefix is not supported in the *Var* parameters of [legacy If commands](#), so [If \(expression\)](#) should be used instead.

Text Parameters

Text parameters accept [unquoted text](#). For example:

```
MsgBox, The time is %A_Hour% o'clock.
```

Since commas and percent signs have special meaning, use the [escape sequence](#) `` ,` to specify a literal comma and `` %` to specify a literal percent sign. For clarity, it is best to always escape any comma which is intended to be literal, but escaping comma is optional in the following cases:

- In the last parameter of any command.
- In the *Text* parameter of `MsgBox`, which has smart comma handling.

To include a leading or trailing space or tab, use the built-in variables [%A_Space%](#) and [%A_Tab%](#) or a forced expression such as `% " x "`.

[v1.1.06+]: Whitespace can also be preserved by preceding the space or tab with an [escape character](#), except for whitespace at the end of a line.

Text parameters can also accept a [forced expression](#).

Numeric Parameters

Numeric parameters accept a literal number or an [expression](#), and can be identified by phrasing like "This parameter can be an expression."

For historical reasons, simple variable references alone or combined with digits are not interpreted as expressions. For example:

```
Sleep %n%000 ; Sleep for n seconds.  
Sleep %m%    ; Sleep for m milliseconds.
```

To perform a [double-deref](#) in such cases, enclose the expression in parentheses:
`Sleep (%m%)`

Note that mixed-type parameters such as [SetTimer's](#) second parameter, which sometimes accepts a number and sometimes accepts a string such as `On` or `Off`, are actually Text parameters, and as such, they do not accept expressions unless the [percent prefix](#) is used.

Numeric parameters allow and ignore the [percent prefix](#).

% Expression

Although purely numeric parameters accept an expression by default, all other parameters of commands do not. Specify a percent sign followed by a space or tab to force a parameter to accept an [expression](#). For example, all of the following are effectively identical because [Sleep's](#) first parameter is expression-capable:

```
Sleep MillisecondsToWait  
Sleep %MillisecondsToWait%  
Sleep % MillisecondsToWait
```

Note: Using the percent-space prefix in a [numeric parameter](#) does not

necessarily force it to be an expression.

All parameters support the percent-space prefix except for:

- The *Var* parameter of any **legacy If** command. Users can avoid confusion by always using **if (expression)**.
- *OutputVar* parameters, which accept a variable reference using the same syntax as expressions.

Some users may find it easier to always force an expression, keeping to one consistent syntax (expression syntax) as much as possible.

Documentation Conventions

At the top of each page which documents a command, there is usually a block showing syntax, like this:

```
Run, Target [, WorkingDir,  
Max|Min|Hide|UseErrorLevel, OutputVarPID]
```

The square brackets denote optional parameters; the brackets themselves must be omitted from the actual code.

Sometimes the value or values a parameter accepts are written directly in the syntax block. For example, the third parameter of **Run** shown above accepts one or more of the words shown, as text. The pipe symbol denotes alternatives; the symbol itself should be omitted from the actual code. The exact syntax of the parameter is described in the *Parameters* table, and varies between commands.

Optional Parameters

Optional parameters can simply be left blank. The comma preceding an optional parameter can also be omitted if all subsequent parameters are omitted. For example, the **Run** command can accept between one and four parameters. All of the following are valid:

```
Run, notepad.exe, C:\
Run, notepad.exe,, Min
Run notepad.exe, , , notepadPID
```

Expressions vs Legacy Syntax

Many command parameters do not accept expressions by default. Use the [percent-space prefix](#) at the beginning of a parameter to evaluate that parameter as an expression. In the following examples, the expression is shown on the first line (beginning *after* the percent sign), with pure legacy syntax shown on the second line.

```
MsgBox % 1+1 ; Shows "2"  
MsgBox 1+1 ; Shows "1+1"
```

Literal text in an expression is always enclosed in quote marks. These are called *quoted strings*.

```
MsgBox % "This is text."  
MsgBox This is text.
```

Variables in an expression are never enclosed in percent signs, except to create a [double reference](#).

```
MsgBox % A_AhkVersion  
MsgBox %A_AhkVersion%
```

Variables cannot be used inside a quoted string.

```
MsgBox % "Hello %A_UserName%." ; Shows  
"%A_UserName%"  
MsgBox Hello %A_UserName%. ; Shows your  
username.
```

Instead, values are [concatenated](#) by writing them in sequence, separated by a space or tab, or a dot surrounded by spaces.

```
MsgBox % "Hello " . A_UserName . "." ; Shows  
your username.
```

One alternative is to use the [Format](#) function, which can also format the parameter value in various ways.

```
MsgBox % Format("Hello {1}.", A_UserName) ; {}  
also works in place of {1}.
```

A value is assigned to a variable with `:=` instead of `=`:

```
MyVar := "This is text."  
MyVar = This is text.
```

Comparisons are performed using the same symbols as [legacy If](#): `=`, `<>` or `!=", >, >=, <` and `<=`.

```
if (Var1 = Var2)  
if Var1 = %Var2%
```

In an expression, both values can be simple values or complex sub-expressions. A comparison can also be combined with other conditions using [operators](#) such as `and` and `or` (which are equivalent to `&&` and `||`).

```
if (Var1 >= Low and Var1 <= High)  
if Var1 between %Low% and %High%
```

Different Equals

One common mistake is to write `=` where `:=` is needed. For example:

```
Total = A + B ; Assigns the literal text "A +  
B"
```

This can be difficult to avoid (at least until such time as the legacy assignment syntax is removed), but it may help to always use `:=` where an assignment is intended.

The equal sign (when not used with another symbol, such as `<=`) has the following meanings:

- **Legacy assignment:** `Var = Value`
- **Legacy If equal:** `if Var = Value`
- **Case-insensitive equal:** `if (Expr1 = Expr2)` (also valid in other expressions, not just `if`)
- **Assign after comma:** `x:=1, y=2, a=b=c` (all are assignments due to a special rule)
- **Declare and initialize:** `local x = Expr` (always accepts an expression)
- **Set parameter default value:** `MyFunc(Param="Default value")`
{ ...

The first two cases can be avoided by always using the `:=` [assignment operator](#) and [if \(expression\)](#).

For the last three cases, `:=` should have been used instead of `=`.

Commands vs Functions

In AutoHotkey v1, it is currently not possible to call a command from an expression, or to call a function using the *command syntax*. However, several commands have a function replacement.

Command	Replacement
FileAppend	FileOpen and File.Write
FileGetAttrib	FileExist
FileRead	FileOpen and File.Read
GetKeyState	GetKeyState (the function returns 0 or 1, not "U" or "D")

IfExist	FileExist
IfInString	InStr
IfWinActive	WinActive
IfWinExist	WinExist
StringGetPos	InStr
StringLen	StrLen
StringReplace	StrReplace
StringSplit	StrSplit
StringLower StringUpper	<code>Format("{:L}", input)</code> , <code>Format("{:U}", input)</code> or <code>Format("{:T}", input)</code>
StringLeft StringMid StringRight StringTrimLeft StringTrimRight	SubStr

Control Flow Statements

For a general explanation of control flow, see [Control Flow](#).

Statements are grouped together into a *block* by enclosing them in braces `{ }`, as in C, JavaScript and similar languages, but usually the braces must appear at the start of a line. Control flow statements can be applied to an entire block or just a single statement.

The **body** of a control flow statement is always a single *group* of statements. A block counts as a single group of statements, as does a control flow statement and its body. The following related statements are also grouped with each other, along with their bodies: **If** with **Else**; **Loop / For** with **Until**; **Try** with **Catch** and/or **Finally**. In other words, when a group of these statements is used as a whole, it does not always need to be enclosed in braces (however, some coding styles always include the braces, for clarity).

Control flow statements which have a body and therefore must always be followed by a related statement or group of statements: **If**, **Else**, **Loop**, **While**, **For**, **Try**, **Catch** and **Finally**.

The following control flow statements exist:

- A **block** (denoted by a pair of braces) groups zero or more statements to act as a single statement.
- An **If statement** causes its body to be executed or not depending on a condition. It can be followed by an **Else** statement, which executes only if the condition was not met.
- **Goto** jumps to the specified label and continues execution.
- **Gosub** calls a **subroutine**.
- **Return** returns from a **subroutine** or function.
- A **Loop statement** (**Loop**, **While** or **For**) executes its body repeatedly.
 - **Break** exits (terminates) a loop.
 - **Continue** skips the rest of the current loop iteration and begins a new one.
 - **Until** causes a loop to terminate when an expression evaluates to true. The expression is evaluated after each iteration.

- Exception handling:
 - **Try** guards its body against runtime errors and exceptions thrown by the throw command.
 - **Catch** executes its body after an exception is thrown within a try statement (and only if an exception is thrown).
 - **Finally** executes its body when control is being transferred out of a try or catch statement's body.
 - **Throw** throws an exception to be handled by try/catch or display an error dialog.

Control Flow vs Commands

Control flow statements have syntax resembling **commands**, and are often referred to as such, but some differ from commands:

- There are several types of **If statement**, with each having different syntax.
- **For** and several types of **If statement** use keywords or an operator instead of commas to separate some of their parameters.
- The opening brace of a **block** can be written at the end of the same line as an **If (expression)**, **Else**, **Loop Count**, **While**, **For**, **Try**, **Catch** or **Finally** statement (basically any control flow statement which does not use **legacy syntax**). This is referred to as the One True Brace (OTB) style. It is not supported by the other Loop sub-commands or **legacy If statement**, as the brace would be interpreted as a literal `{` character.
- **Else**, **Try** and **Finally** allow any valid statement to their right, as they require a **body** but have no parameters.
- **If (expression)** and **While** allow an open parenthesis to be used immediately after the name. For example, `if(expression)`.
- **For**, **While**, **Until** and **Throw** always accept expressions. They treat `%var%`, `%var%000` and similar as expressions, whereas **numeric parameters** of other commands do not. The requirement for backward-compatibility does not apply to these control flow statements as they are relatively new.

If Statement

If (expression) evaluates an expression and executes the following statement only if the result is true.

Common cause of confusion: There are several other types of If statements, some of which look very similar to *If (expression)*. These should be avoided in new scripts. If in doubt, it is best to always begin the expression with an open-parenthesis. The "legacy" If statements are as follows:

- [If Var](#)
- [If Var op Value](#), where *op* is one of the following operators: `=`, `<>`, `!=`, `>`, `>=`, `<`, `<=`.
- [If Var \[not\] between Lower and Upper](#)
- [If Var \[not\] in/contains MatchList](#)
- [If Var is \[not\] Type](#)

Any If statement which does not match one of the usages shown above is interpreted as [If \(expression\)](#).

These are some common points of confusion related to legacy If statements:

- Variable names must be enclosed in percent signs *only* on the right-hand side of the operator.
- `between`, `in`, `contains` and `is` are only valid in this context; they cannot be used in [expressions](#).
- Multiple conditions cannot be written on the same line (such as with the `and` operator).
- None of the parameters are expressions.

The following "legacy" named If statements also exist:

- [IfEqual](#), [IfNotEqual](#), [IfLess](#), [IfLessOrEqual](#), [IfGreater](#) and [IfGreaterOrEqual](#)
- [If\[Not\]Exist](#)
- [If\[Not\]InString](#)
- [If\[Not\]WinActive](#)
- [If\[Not\]WinExist](#)
- [IfMsgBox](#)

With the exception of `IfMsgBox`, these are all obsolete and generally should be avoided in new scripts.

Named If statements allow a [command](#) to be written on the same line, but misspelled command names are treated as literal text. Such errors may be difficult

to detect.

Loop Statement

There are several types of loop statements:

- **Loop Count** executes a statement repeatedly: either the specified number of times or until break is encountered.
- **Loop Reg** retrieves the contents of the specified registry subkey, one item at a time.
- **Loop Files** retrieves the specified files or folders, one at a time.
- **Loop Parse** retrieves substrings (fields) from a string, one at a time.
- **Loop Read** retrieves the lines in a text file, one at a time.
- **While** executes a statement repeatedly until the specified expression evaluates to false. The expression is evaluated before each iteration.
- **For** executes a statement once for each value or pair of values returned by an enumerator, such as each key-value pair in an object.

Break exits (terminates) a loop, effectively jumping to the next line after the loop's body.

Continue skips the rest of the current loop iteration and begins a new one.

Until causes a loop to terminate when an expression evaluates to true. The expression is evaluated after each iteration.

A **label** can be used to "name" a loop for **Continue** and **Break**. This allows the script to easily continue or break out of any number of nested loops without using **Goto**.

The built-in variable **A_Index** contains the number of the current loop iteration. It contains 1 the first time the loop's body is executed. For the second time, it contains 2; and so on. If an inner loop is enclosed by an outer loop, the inner loop takes precedence. **A_Index** works inside all types of loops, but contains 0 outside of a loop.

For some loop types, other built-in variables return information about the current loop item (registry key/value, file, substring or line of text). These variables have names beginning with **A_Loop**, such as **A_LoopFileName** and

A_LoopReadLine. Their values always correspond to the most recently started (but not yet stopped) loop of the appropriate type. For example, A_LoopField returns the current substring in the innermost parsing loop, even if it is used inside a file or registry loop.

```
t := "column 1`column 2`nvalue 1`tvalue 2"
Loop Parse, t, `n
{
    rowtext := A_LoopField
    rownum := A_Index ; Save this for use in
the second loop, below.
    Loop Parse, rowtext, `t
    {
        MsgBox %rownum%:%A_Index% =
%A_LoopField%
    }
}
```

Loop variables can also be used outside the body of a loop, such as in a function or subroutine which is called from within a loop.

Not Control Flow

As directives, labels (including hotkeys and hotstrings), and declarations without assignments are processed when the script is loaded from file, they are not subject to control flow. In other words, they take effect unconditionally, before the script ever executes any control flow statements. Similarly, the #If directives such as #IfWinActive cannot affect control flow; they merely set the criteria for any hotkey labels and hotstrings specified in the code. A hotkey's criteria is evaluated each time it is pressed, not when the #If directive is encountered in the code.

Structure of a Script

Auto-execute Section

After the script has been loaded, it begins executing at the top line, continuing until a [Return](#), [Exit](#), the script's first [hotkey/hotstring label](#), or the physical end of the script is encountered (whichever comes first). This top portion of the script is referred to as the *auto-execute section*, but it is really just a [subroutine](#) which is called after program startup.

Note: While the script's *first* hotkey/hotstring label has the same effect as [return](#), other hotkeys and labels do not.

The auto-execute section is often used to configure settings which apply to every newly launched [thread](#). For details, see [The Top of the Script](#).

Subroutines

A *subroutine* (or sub) is a reusable block of code which can be *called* to perform some task.

Scripts use subroutines to define what should happen when a particular hotkey is pressed or some other event occurs. Scripts can also call subroutines directly, by using [Gosub](#).

Any [label](#) can be used as the starting point of a subroutine. A subroutine has no explicitly marked ending point, but instead ends if and when control is returned to the subroutine's caller by [Return](#) or when the thread is exited. For example:

```
gosub Label1

Label1:
MsgBox %A_ThisLabel%
return
```

Note that as labels have no effect when reached during normal execution, in this example a MsgBox would be shown twice: once while the subroutine is running and again after it returns. One important consequence is that you cannot define one subroutine inside another subroutine, because the "body" of the inner subroutine would execute automatically and then *return*, effectively terminating the outer subroutine.

Subroutines should typically be defined separately to any other block of code, but can also be [defined inside a function](#), allowing the subroutine access to that function's static variables (and local variables, but only while the function is running).

Note: Subroutines defined inside a function have certain limitations regarding the use of local variables and dynamic variable references, including Gui control variables. For details, see [Using Subroutines Within a Function](#).

User-Defined Functions

Generally speaking, a [function](#) is a kind of subroutine. However, within the AutoHotkey documentation, "subroutine" typically refers to the kind of subroutine defined by a label (described above).

User-defined functions differ from subroutines in that they can *accept parameters* and *return a value*, and they can have [local variables](#). They can be called either by a [function call](#) within the script or by the program itself, such as if a function was passed to the [Hotkey](#) or [SetTimer](#) commands.

Functions are defined using syntax resembling a function call followed by a block of code enclosed in braces:

```
MyFunction(FirstParameter, Second, ByRef Third,
Fourth:= "")
{
    ...
    return "a value"
}
```

As with function calls, there must be no space between the function name and open-parenthesis.

The line break between the close-parenthesis and open-brace is optional. There can be any amount of whitespace or comments between the two.

ByRef indicates that the parameter accepts a variable reference, making that parameter an alias for whichever variable the caller passes. If the caller does not pass a variable, the parameter acts as a normal local variable. ByRef parameters can also be optional.

Optional parameters are specified by following the parameter name with `:=` or `=` and a default value, which must be a literal quoted string, a number, `true` or `false`. The operators `:=` and `=` are interchangeable for historical reasons, but it is best to use `:=` for consistency with assignment in expressions.

The function can **return a value**. If it does not, the default return value is an empty string.

A function cannot be defined inside another function. Otherwise, the position of a function definition does not matter; any function defined within the script can be called from anywhere else.

See [Functions](#) for much more detail.

#Include

The **#Include** directive causes the script to behave as though the specified file's contents are present at this exact position. This is often used to organise code into separate files, or to make use of script libraries written by other users.

Note: The following paragraphs detail some common points of confusion.

When using **#Include**, it is important to consider what effect the file's contents would have if placed at that position, since **#Include** will have the same effect. For instance:

- **#Include** generally should not be used in the middle of a subroutine or

function.

- The use of `#Include` in the script's [auto-execute section](#) requires special consideration, because the auto-execute section is essentially just a subroutine. Execution of a subroutine halts if it reaches a `return`, regardless of which file that `return` is in. Similarly, if the file contains a hotkey/hotstring, it may be considered the script's *first* hotkey/hotstring, which would act like `return`.
- The script only has one [auto-execute section](#), not one per file.

`#Include` can be safely used within the [auto-execute section](#) to include files which contain only function definitions, since function definitions (but not function calls) are skipped over during execution. If a file contains other code, one can avoid breaking the auto-execute section by skipping over the file's contents with [Goto](#).

Unlike in C/C++, `#Include` does nothing if the file has already been included by a previous directive. To include the contents of the same file multiple times, use [#IncludeAgain](#).

Script files containing functions can be *automatically* included without having to use `#Include`, if they are saved in a standard location and named appropriately. The effect is similar to using `#Include` at the end of the main script file. For details, see [Libraries of Functions](#).

Miscellaneous

Dynamic Variables

A *dynamic variable reference* takes a text value and interprets it as the name of a variable.

The most common form of dynamic variable reference is called a *double reference* or *double-deref*. Before performing a double reference, the name of the target variable is stored in a second variable. This second variable can then be used to assign a value to the target variable indirectly, using a double reference. For example:

```
target := 42
second := "target"
MsgBox %second% ; Normal (single) variable
reference in text => target
MsgBox % second ; Normal (single) variable
reference in an expression => target
MsgBox % %second% ; Double-deref in an
expression => 42
```

At first, it would appear that percent signs have a different meaning depending on whether they are used in text or in an expression. However, it may make more sense to think of `%second%` as being replaced with the contents of the variable `second` in *both* cases:

- `MsgBox %second%` → `MsgBox target` : Shows "target".
- `MsgBox % %second%` → `MsgBox % target` : Shows the contents of `target`, i.e. "42".

Currently, `second` must always contain a variable name in the second case; arbitrary expressions are not supported.

A dynamic variable reference can also take one or more pieces of literal text and the content of one or more variables, and join them together to form a single

variable name. This is done simply by writing the pieces of the name and percent-enclosed variables in sequence, without any spaces. For example, `MyArray%A_Index%` or `MyGrid%X%_%Y%`. This is used to access *pseudo-arrays*, described below.

For a description of how dynamic variable references inside functions are resolved, see [Functions: More about locals and globals](#).

Pseudo-arrays

A *pseudo-array* is actually just a bunch of discrete variables, but with a naming pattern which allows them to be used like elements of an array. For example:

```
MyArray1 = A
MyArray2 = B
MyArray3 = C
Loop 3
    MsgBox % MyArray%A_Index% ; Shows A, then
    B, then C.
```

As the individual elements are just normal variables, one can assign or retrieve a value, but cannot *remove* or *insert* elements. Because the pseudo-array itself doesn't really exist, it can't be passed to or returned from a function, or copied as a whole. For these reasons, it is generally recommended to use [normal arrays](#) instead, where possible.

Associative pseudo-arrays

The "index" used to form the final variable name does not have to be numeric; it could instead be a letter or keyword, making the pseudo-array similar to an [associative array](#) or an [object](#). The following example creates a pseudo-array with elements "Left", "Top", "Right" and "Bottom":

```
SysGet, WA, MonitorWorkArea
MsgBox, Left: %WALeft% -- Top: %WATop% -- Right:
%WARight% -- Bottom: %WABottom%.
```

Commands which create pseudo-arrays

There are several commands which create associative pseudo-arrays:

- [GuiControlGet Pos](#).
- [RegExMatch](#), except when given the `0`) option, which causes it to output a single object containing all match information.
- [SysGet Monitor/MonitorWorkArea](#), as demonstrated above.
- [StringSplit](#). New scripts should use [StrSplit\(\)](#) instead, as it creates a [normal array](#).
- [WinGet List](#).

Caution: These commands do not follow the same rules as *dynamic variable references*. If used within a function, the resulting pseudo-array is either entirely global or entirely local, depending only on the first element (or base name) of the array. Some of the variables in the pseudo-array may be inaccessible if they are not individually declared. For details, see [Functions: More about locals and globals](#).

AutoHotkey also creates one global pseudo-array to contain any [command line parameters](#) that were passed to the script.

Labels

A label identifies a line of code, and can be used as a [Goto](#) target or to form a [subroutine](#). There are three kinds of label: normal named labels, [hotkey](#) labels and [hotstring](#) labels.

Normal labels consist of a name followed by a colon.

```
this_is_a_label:
```

Hotkey labels consist of a hotkey followed by double-colon.

```
^a::
```

Hotstring labels consist of a colon, zero or more [options](#), another colon, an abbreviation and double-colon.

```
:*:btw::
```

Generally, aside from whitespace and comments, no other code can be written on the same line as a label. However:

- A hotkey label can be directly followed by a command or other statement to create a *one-line* hotkey. In other words, if a command, assignment or expression is present on the same line as a hotkey label, it acts as though followed by `return`.
- A hotkey with a [key name](#) written to the right of the double-colon is actually a *remapping*, which is shorthand for [a pair of hotkeys](#). For example, `a::b` creates hotkeys and labels for `*a` and `*a Up`, and does not create a label named `a`.
- A hotstring with text written to the right of the final double-colon is an *auto-replace* hotstring. Auto-replace hotstrings do not act as labels.

For more details, see [Labels](#).

Hotkeys (Mouse, Joystick and Keyboard Shortcuts)

Table of Contents

- [Introduction and Simple Examples](#)
- [Table of Hotkey Prefix Symbols \(Modifiers\)](#)
- [Context-sensitive Hotkeys](#)
- [Custom Combinations](#)
- [Other Features](#)
- [Mouse Wheel Hotkeys](#)
- [Hotkey Tips and Remarks](#)
- [Alt-Tab Hotkeys](#)
- [Function Hotkeys](#)

Introduction and Simple Examples

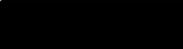
Hotkeys are sometimes referred to as shortcut keys because of their ability to easily trigger an action (such as launching a program or [keyboard macro](#)). In the following example, the hotkey   is configured to launch Notepad.

The pound sign [#] stands for the  which is known as a *modifier*:

```
#n::  
Run Notepad  
return
```

In the final line above, `return` serves to finish the hotkey. However, if a hotkey needs to execute only a single line, that line can be listed to the right of the double-colon. In other words, the `return` is implicit:

```
#n::Run Notepad
```

To use more than one modifier with a hotkey, list them consecutively (the order does not matter). The following example uses `^!s` to indicate  

```
^!s::  
Send Sincerely,{enter}John Smith ; This line  
sends keystrokes to the active (foremost)  
window.  
return
```

You can use the following modifier symbols to define hotkeys:

Symbol	Description

#	<p>Win [redacted] (Windows logo key).</p> <p>[v1.0.48.01+]: For Windows Vista and later, hotkeys that include the Win [redacted] (e.g. #a) will wait for the Win [redacted] to be released before sending any text containing an L [redacted] stroke. This prevents usages of Send within such a hotkey from locking the PC. This behavior applies to all sending modes except SendPlay (which doesn't need it) and blind mode.</p> <p>Note: Pressing a hotkey which includes the Win [redacted] key may result in extra simulated keystrokes (Ctrl [redacted] by default). See #MenuMaskKey.</p>
!	<p>Alt [redacted]</p> <p>Note: Pressing a hotkey which includes the Alt [redacted] key may result in extra simulated keystrokes (Ctrl [redacted] by default). See #MenuMaskKey.</p>
^	<p>Control [redacted]</p>
+	<p>Shift [redacted]</p>
&	<p>An ampersand may be used between any two keys or mouse buttons to combine them into a custom hotkey. See below for details.</p>
<	<p>Use the left key of the pair. e.g. <!a is the same as !a except that only the left Alt [redacted] will trigger it.</p>
>	<p>Use the right key of the pair.</p>
	<p>AltGr [redacted] (alternating/gravings). If your keyboard layout has</p>

AltGr [redacted] a right **Alt** [redacted] this series of symbols can usually be used to stand for **AltGr** [redacted] pple:

<^>!

```
<^>!m::MsgBox You pressed AltGr+m.  
<^<!m::MsgBox You pressed  
LeftControl+LeftAlt+m.
```

Alternatively, to make **AltGr** [redacted] a hotkey, use the following hotkey (without any hotkeys like the above present):

```
LControl & RAlt::MsgBox You pressed  
AltGr itself.
```

Wildcard: Fire the hotkey even if extra modifiers are being held down. This is often used in conjunction with [remapping](#) keys or buttons. For example:

*

```
*#c::Run Calc.exe ; Win+C,  
Shift+Win+C, Ctrl+Win+C, etc. will all  
trigger this hotkey.  
*ScrollLock::Run Notepad ; Pressing  
ScrollLock will trigger this hotkey  
even when modifier key(s) are down.
```

Wildcard hotkeys always use the keyboard hook, as do any hotkeys eclipsed by a wildcard hotkey. For example, the presence of ***a::** would cause **^a::** to always use the hook.

When the hotkey fires, its key's native function will not be blocked (hidden from the system). In both of the below examples, the user's click of the mouse button will be sent to the active window:

```
~RButton::MsgBox You clicked the right
mouse button.
~RButton & C::MsgBox You pressed C
while holding down the right mouse
button.
```

Unlike the other prefix symbols, the tilde prefix is allowed to be present on some of a hotkey's [variants](#) but absent on others. However, if a tilde is applied to the [prefix key](#) of any custom combination which has not been turned off or suspended, it affects the behavior of that prefix key for *all* combinations.

Special hotkeys that are substitutes for [alt-tab](#) always ignore the tilde prefix.

[v1.1.14+]: If the tilde prefix is applied to a custom modifier key ([prefix key](#)) which is also used as its own hotkey, that hotkey will fire when the key is pressed instead of being delayed until the key is released. For example, the `~RButton` hotkey above is fired as soon as the button is pressed. Prior to [v1.1.14] (or without the tilde prefix), it was fired when the button was released, but only if the `RButton & C` combination was not activated.

If the tilde prefix is applied only to the custom combination and not the non-combination hotkey, the key's native function will still be blocked. For example, in the script below, holding  will show the ToolTip and will not trigger a context menu:

```
AppsKey::ToolTip Press < or > to cycle
through windows.
AppsKey Up::ToolTip
~AppsKey & <::Send !+{Esc}
~AppsKey & >::Send !{Esc}
```

If at least one variant of a keyboard hotkey has the tilde modifier, that hotkey always uses the keyboard hook.

\$

This is usually only necessary if the script uses the [Send](#) command to send the keys that comprise the hotkey itself, which might otherwise cause it to trigger itself. 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](#) somewhere above the definition of this hotkey.

The \$ prefix has no effect for mouse hotkeys, since they always use the mouse hook. It also has no effect for hotkeys which already require the keyboard hook, including any keyboard hotkeys with the [tilde \(~\)](#) or [wildcard \(*\)](#) modifiers, key-up hotkeys and custom combinations. To determine whether a particular hotkey uses the keyboard hook, use [ListHotkeys](#).

[v1.1.06+]: [#InputLevel](#) and [SendLevel](#) provide additional control over which hotkeys and hotstrings are triggered by the Send command.

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  become the left .

```
*LWin::Send {LControl down}
*LWin Up::Send {LControl up}
```

"Up" can also be used with normal hotkeys as in this example:
[^!r Up::MsgBox You pressed and released Ctrl+Alt+R](#) . It also works with [combination hotkeys](#) (e.g. [F1 & e Up::](#))

Limitations: 1) "Up" does not work with [joystick buttons](#); and 2) An "Up" hotkey without a normal/down counterpart hotkey will

UP

completely take over that key to prevent it from getting stuck down. One way to prevent this is to add a [tilde prefix](#) (e.g. `~LControl up::`)

"Up" hotkeys and their key-down counterparts (if any) always use the keyboard hook.

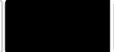
On a related note, a technique similar to the above is to make a hotkey into a prefix key. The advantage is that although the hotkey will fire upon release, it will do so only if you did not press any other key while it was held down. For example:

```
LControl & F1::return ; Make left-  
control a prefix by using it in front  
of "&" at least once.  
LControl::MsgBox You released LControl  
without having used it to modify any  
other key.
```

Note: See the [Key List](#) for a complete list of keyboard keys and mouse/joystick buttons.

Multiple hotkeys can be stacked vertically to have them perform the same action. For example:

```
^Numpad0::  
^Numpad1::  
MsgBox Pressing either Control+Numpad0 or  
Control+Numpad1 will display this message.  
return
```

A key or key-combination can be disabled for the entire system by having it do nothing. The following example disables the right-side  

```
Rwin::return
```

Context-sensitive Hotkeys

The directives `#IfWinActive/Exist` and `#If` can be used to make a hotkey perform a different action (or none at all) depending on a specific condition. For example:

```
#IfWinActive, ahk_class Notepad
^a::MsgBox You pressed Ctrl-A while Notepad is
active. Pressing Ctrl-A in any other window will
pass the Ctrl-A keystroke to that window.
#c::MsgBox You pressed Win-C while Notepad is
active.

#IfWinActive
#c::MsgBox You pressed Win-C while any window
except Notepad is active.

#If MouseIsOver("ahk_class Shell_TrayWnd")
WheelUp::Send {Volume_Up} ; Wheel over
taskbar: increase/decrease volume.
WheelDown::Send {Volume_Down} ;
```

Custom Combinations

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

```
Numpad0 & Numpad1::MsgBox You pressed Numpad1
while holding down Numpad0.
Numpad0 & Numpad2::Run Notepad
```

The prefix key loses its native function: In the above example, Numpad0 becomes a *prefix key*; but this also causes Numpad0 to lose its original/native function when it is pressed by itself. To avoid this, a script may configure Numpad0 to perform a new action such as one of the following:

```
Numpad0::WinMaximize A ; Maximize the
active/foreground window.
Numpad0::Send {Numpad0} ; Make the release of
Numpad0 produce a Numpad0 keystroke. See comment
below.
```

Fire on release: The presence of one of the above custom combination hotkeys causes the *release* of Numpad0 to perform the indicated action, but only if you did not press any other keys while Numpad0 was being held down. [v1.1.14+]: This behaviour can be avoided by applying the [tilde prefix](#) to either hotkey.

Modifiers: Unlike a normal hotkey, custom combinations act as though they have the [wildcard \(*\)](#) modifier by default. For example, `1 & 2::` will activate even if `Ctrl` `Alt` down when `1` `2` pressed, whereas `^1::` would be activated only by `Ctrl` `1` not `Ctrl` `Alt` `1`

For standard modifier keys, normal hotkeys typically work as well or better than "custom" combinations. For example, `<+s::` is recommended over `LShift`

`& s::.`

Combinations of three or more keys are not supported. Combinations which your keyboard hardware supports can usually be detected by using `#If` and `GetKeyState`, but the results may be inconsistent. For example:

```
; Press AppsKey and Alt in any order, then slash (/).
#if GetKeyState("AppsKey", "P")
Alt & /::MsgBox Hotkey activated.

; If the keys are swapped, Alt must be pressed first (use one at a time):
#if GetKeyState("Alt", "P")
AppsKey & /::MsgBox Hotkey activated.

; [ & ] & \::
#if GetKeyState("[") && GetKeyState("]")
\::MsgBox
```

Keyboard hook: Custom combinations involving keyboard keys always use the keyboard hook, as do any hotkeys which use the prefix key as a suffix. For example, `a & b::` causes `^a::` to always use the hook.

Other Features

NumLock, CapsLock, and ScrollLock: These keys may be forced to be "AlwaysOn" or "AlwaysOff". For example: `SetNumLockState AlwaysOn`.

Overriding Explorer's hotkeys: Windows' built-in hotkeys such as `Win` `[]` and `Win` `[R]` can be individually overridden simply by assigning them to an action in the script. See the [override page](#) for details.

Substitutes for Alt-Tab: Hotkeys can provide an alternate means of alt-tabbing. For example, the following two hotkeys allow you to alt-tab with your right hand:

```
RControl & RShift::AltTab ; Hold down right-control then press right-shift repeatedly to move forward.  
RControl & Enter::ShiftAltTab ; Without even having to release right-control, press Enter to reverse direction.
```

For more details, see [Alt-Tab](#).

Mouse Wheel Hotkeys

Hotkeys that fire upon turning the mouse wheel are supported via the key names WheelDown and WheelUp. [v1.0.48+]: WheelLeft and WheelRight are also supported, but have no effect on operating systems older than Windows Vista. Here are some examples of mouse wheel hotkeys:

```
MButton & WheelDown::MsgBox You turned the mouse
wheel down while holding down the middle button.
^!WheelUp::MsgBox You rotated the wheel up while
holding down Control+Alt.
```

[v1.0.43.03+]: The built-in variable **A_EventInfo** contains the amount by which the wheel was turned, which is typically 1. However, A_EventInfo can be greater or less than 1 under the following circumstances:

- If the mouse hardware reports distances of less than one notch, A_EventInfo may contain 0;
- If the wheel is being turned quickly (depending on type of mouse), A_EventInfo may be greater than 1. A hotkey like the following can help analyze your mouse: `~WheelDown::ToolTip %A_EventInfo%`.

Some of the most useful hotkeys for the mouse wheel involve alternate modes of scrolling a window's text. For example, the following pair of hotkeys scrolls horizontally instead of vertically when you turn the wheel while holding down the left

Control	
---------	---

```
~LControl & WheelUp:: ; Scroll left.
ControlGetFocus, fcontrol, A
Loop 2 ; <-- Increase this value to scroll
faster.
    SendMessage, 0x114, 0, 0, %fcontrol%, A ;
0x114 is WM_HSCROLL and the 0 after it is
SB_LINELEFT.
return
```

```
~LControl & WheelDown:: ; Scroll right.  
ControlGetFocus, fcontrol, A  
Loop 2 ; <-- Increase this value to scroll  
faster.  
    SendMessage, 0x114, 1, 0, %fcontrol%, A ;  
0x114 is WM_HSCROLL and the 1 after it is  
SB_LINERIGHT.  
return
```

Finally, since mouse wheel hotkeys generate only down-events (never up-events), they cannot be used as [key-up hotkeys](#).

Hotkey Tips and Remarks

Each numpad key can be made to launch two different hotkey subroutines depending on the state of `NumLock` .y, a numpad key can be made to launch the same subroutine regardless of the `NumLock`  example:

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

If the [tilde \(~\) operator](#) is used with a [prefix key](#) even once, it changes the behavior of that prefix key for all combinations. For example, in both of the below hotkeys, the active window will receive all right-clicks 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.
```

The [Suspend](#) command can temporarily disable all hotkeys except for ones you make exempt. For greater selectivity, use [#IfWinActive/Exist](#).

By means of the [Hotkey](#) command, hotkeys can be created dynamically while the script is running. The Hotkey command can also modify, disable, or enable the script's existing hotkeys individually.

Joystick hotkeys do not currently support modifier prefixes such as `^ (` `Control`  `Win` .ever, you can use [GetKeyState](#) to mimic this effect as shown in the following example:

```

Joy2::
if not GetKeyState("Control") ; Neither the
left nor right Control key is down.
    return ; i.e. Do nothing.
MsgBox You pressed the first joystick's second
button while holding down the Control key.
return

```

There may be times when a hotkey should wait for its own modifier keys to be released before continuing. Consider the following example:

```
^!s::Send {Delete}
```

Pressing **Control** **Alt** **S** would cause the system to behave as though you pressed **Control** **Alt** **Delete** system's aggressive detection of **Ctrl** **Alt** **Delete** around this, use [KeyWait](#) to wait for the keys to be released; for example:

```

^!s::
KeyWait Control
KeyWait Alt
Send {Delete}
return

```

If a hotkey label like `#Z::` produces an error like "Invalid Hotkey", your system's keyboard layout/language might not have the specified character ("Z" in this case). Try using a different character that you know exists in your keyboard layout.

A hotkey label can be used as the target of a [Gosub](#) or [Goto](#). For example:
`Gosub ^!s .`

One common use for hotkeys is to start and stop a repeating action, such as a series of keystrokes or mouse clicks. For an example of this, see [this FAQ topic](#).

Finally, each script 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.

Alt-Tab Hotkeys

Each Alt-Tab hotkey must be a combination of two keys, which is typically achieved via the ampersand symbol (&). In the following example, you would hold down the right  and press   to navigate the alt-tab menu:

```
RAlt & j::AltTab
RAlt & k::ShiftAltTab
```

AltTab and *ShiftAltTab* are two of the special commands that are only recognized when used on the same line as a hotkey. Here is the complete list:

AltTab: If the alt-tab menu is visible, move forward in it. Otherwise, display the menu (only if the hotkey is an "&" combination of two keys; otherwise, it does nothing).

ShiftAltTab: Same as above except move backward in the menu.

AltTabAndMenu: If the alt-tab menu is visible, move forward in it. Otherwise, display the menu.

AltTabMenuDismiss: Close the Alt-tab menu.

To illustrate the above, the mouse wheel can be made into an entire substitute for Alt-tab. With the following hotkeys in effect, clicking the middle button displays the menu and turning the wheel navigates through it:

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

To cancel a hotkey-invoked Alt-tab menu without activating the selected window, use a hotkey such as the following. It might require adjustment depending on: 1) the means by which the alt-tab menu was originally displayed;

and 2) whether the script has the [keyboard hook](#) installed.

```
LCtrl & CapsLock::AltTab
!MButton:: ; Middle mouse button. The ! prefix
makes it fire while the Alt key is down (which
it is if the alt-tab menu is visible).
IfWinExist ahk_class #32771 ; Indicates that
the alt-tab menu is present on the screen.
    Send !{Escape}{Alt up}
return
```

Currently, all special Alt-tab actions must be assigned directly to a hotkey as in the examples above (i.e. they cannot be used as though they were commands). They are **not affected by #IfWin or #If**.

Custom alt-tab actions can also be created via hotkeys. In the following example, you would press **F1** [redacted] display the menu and advance forward in it. Then you would press **F2** [redacted] activate the selected window (or press **Escape** [redacted] cancel):

```
*F1::Send {Alt down}{tab} ; Asterisk is required
in this case.
!F2::Send {Alt up} ; Release the Alt key, which
activates the selected window.
~*Escape::
IfWinExist ahk_class #32771
    Send {Escape}{Alt up} ; Cancel the menu
without activating the selected window.
return
```

Function Hotkeys [v1.1.20+]

One or more hotkeys can be assigned a [function](#) by simply defining it immediately after the hotkey label as in this example:

```
; Ctrl+Shift+O to open containing folder in
Explorer.
; Ctrl+Shift+E to open folder with current file
selected.
; Supports SciTE and Notepad++.
^+o::
^+e::
    editor_open_folder() {
        WinSetTitle, path, A
        if RegExMatch(path, "\*?\K(.*)\\\[^\]"+(?
= [-*] )" , path)
            if (FileExist(path) && A_ThisHotkey
= "^+e")
                Run explorer.exe
            /select `,"%path%"
            else
                Run explorer.exe "%path1%"
    }
```

[v1.1.28+]: Hotstrings can also be defined this way. Multiple hotkeys or hotstrings can be stacked together to call the same function.

There must only be whitespace, comments or directives between the hotkey/hotstring labels or label and the function. Hotkey/hotstring labels defined this way are not visible to [IsLabel](#), [Gosub](#) or other commands; however, the [auto-execute section](#) ends at the first hotkey/hotstring even if it is assigned a function.

The main benefit of using a function is that local variables can be used, which avoids conflicts when two or more hotkeys use the same variable names for different purposes. It also encourages self-documenting hotkeys, like in the code

above where the function name describes the hotkey.

The [Hotkey](#) command can also be used to assign a function or function object to a hotkey.

Hotstrings and Auto-replace

Introduction and Simple Examples

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).

To define a hotstring, enclose the triggering abbreviation between pairs of 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 you type it (however, by default you must type an [ending character](#) after typing btw, such as  

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 the following examples. Note that the commands must appear beneath the hotstring:

```
::btw::  
MsgBox You typed "btw".  
return  
  
*:]d:: ; This hotstring replaces "]d" with the  
current date and time via the commands below.  
FormatTime, CurrentDateTime,, M/d/yyyy h:mm tt  
; It will look like 9/1/2005 3:53 PM  
SendInput %CurrentDateTime%  
return
```

Even though the two examples above are not auto-replace hotstrings, the abbreviation you type is erased by default. This is done via automatic backspacing, which can be disabled via the [b0 option](#).

Ending Characters

Unless the [asterisk option](#) is in effect, you must type an *ending character* after a hotstring's abbreviation to trigger it. Ending characters initially consist of the following: `-()[]{}';"/\, .?!`n`t` (note that ``n` is   here is a plain space between ``n` and ``t`). This set of characters can be changed by editing the following example, which sets the new ending characters for all hotstrings, not just the ones beneath it:

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

[v1.1.28+]: The ending characters can be changed while the script is running by calling the [Hotstring](#) function as demonstrated below:

```
Hotstring("EndChars", "-()[]{}';")
```

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. The following example puts the C and R options into effect: `#Hotstring c r`.
2. Putting options inside a hotstring's first pair of colons. The following example puts the C and * options (case sensitive and "ending character not required") into effect for a single hotstring:
`:c*:j@::john@somedomain.com`.

The list below describes each option. When specifying more than one option using the methods above, spaces optionally may be included between them.

* (asterisk): An ending character (e.g. `Space`  `Enter`  required to trigger the hotstring. For example:

```
:*:j@::jsmith@somedomain.com
```

The example above would send its replacement the moment you type the @ character. When using the `#Hotstring directive`, use `*0` to turn this option back off.

? (question mark): 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.

B0 (B followed by a zero): Automatic backspacing is not done to erase the abbreviation you type. Use a plain **B** to turn backspacing back on after it was previously turned off. A script may also do its own backspacing via `{bs 5}`, which sends 5 `Backspace`  hilarly, it may send `←`  keystrokes via `{left 5}`. For example, the following hotstring produces "``" and moves the caret 5 places to the left (so that it's between the tags):

```
:*b0:<em>::</em>{left 5}
```

C: Case sensitive: When you type an abbreviation, it must exactly match the case defined in the script. Use **C0** to turn case sensitivity back off.

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 of the characters you actually type. Case-conforming hotstrings (which are the default) produce their replacement text in all caps if you type the abbreviation in all caps. If you type the first letter in caps, the first letter of the replacement will also be capitalized (if it is a letter). If you type the case in any other way, the replacement is sent exactly as defined. When using the [#Hotstring directive](#), **C0** can be used to turn this option back off, which makes hotstrings conform again.

Kn: Key-delay: This rarely-used option sets the delay between keystrokes produced by auto-backspacing or [auto-replacement](#). Specify the new delay for **n**; for example, specify k10 to have a 10ms delay and k-1 to have no delay. The exact behavior of this option depends on which [sending mode](#) is in effect:

- **SI (SendInput):** Key-delay is ignored because a delay is not possible in this mode. The exception to this is when SendInput is [unavailable](#), in which case hotstrings revert to SendPlay mode below (which does obey key-delay).
- **SP (SendPlay):** A delay of length zero is the default, which for SendPlay is the same as -1 (no delay). In this mode, the delay is actually a [PressDuration](#) rather than a delay between keystrokes.
- **SE (SendEvent):** A delay of length zero is the default. Zero is recommended for most purposes since it is fast but still cooperates well with other processes (due to internally doing a [Sleep 0](#)). Specify k-1 to have no delay at all, which is useful to make auto-replacements faster if your CPU is frequently under heavy load. When set to -1, a script's process-priority becomes an important factor in how fast it can send keystrokes. To raise a script's priority, use [Process, Priority,, High](#).

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 the spacebar will produce "aristocrat" with no trailing space, which allows you to make the word plural or possessive without having to press **Backspace** (letter O followed by a zero) to turn this option back off.

Pn: The [priority](#) of the hotstring (e.g. P1). This rarely-used option has no effect on [auto-replace hotstrings](#).

R: Send the replacement text [raw](#); that is, without translating {Enter} to **Enter** **Control** **C**. This option is put into effect automatically for hotstrings that have a [continuation section](#). Use **R0** to turn this option back off.

Note: [Text mode](#) may be more reliable. The R and T options are mutually exclusive.

SI or **SP** or **SE** [v1.0.43+]: Sets the method by which [auto-replace hotstrings](#) send their keystrokes. These options are mutually exclusive: only one can be in effect at a time. The following describes each option:

- **SI** stands for [SendInput](#), which typically has superior speed and reliability than the other modes. Another benefit is that like [SendPlay](#) below, [SendInput](#) postpones anything you type during a hotstring's [auto-replacement text](#). This prevents your keystrokes from being interspersed with those of the replacement. When [SendInput](#) is [unavailable](#), hotstrings automatically use [SendPlay](#) instead.
- **SP** stands for [SendPlay](#), which may allow hotstrings to work in a broader variety of games.
- **SE** stands for [SendEvent](#), which is the default in versions older than 1.0.43.

If none of the above options are used, the default mode in [v1.0.43] and later is [SendInput](#). However, unlike the **SI** option, [SendEvent](#) is used instead of [SendPlay](#) when [SendInput](#) is unavailable.

T [v1.1.27+]: Send the replacement text [raw](#), without translating each character to a keystroke. For details, see [Text mode](#). Use **T0** or **R0** to turn this option back off, or override it with **R**.

X [v1.1.28+]: Execute. Instead of replacement text, the hotstring accepts a command or expression to execute. For example, `:X:~mb::MsgBox` would cause a message box to be displayed when the user types "~mb" instead of auto-replacing it with the word "MsgBox". This is most useful when defining a large number of hotstrings which call functions, as it would otherwise require three lines per hotstring.

When used with the [Hotstring](#) function, the X option causes the *Replacement* parameter to be interpreted as a label or function name instead of replacement text. However, the X option has this effect only if it is specified each time the function is called.

Z: This rarely-used option resets the hotstring recognizer after each triggering of the hotstring. In other words, the script will begin waiting for an entirely new hotstring, eliminating from consideration anything you previously typed. This can prevent unwanted triggerings of hotstrings. To illustrate, consider the following hotstring:

```
:b0*?:11::  
SendInput xx  
return
```

Since the above lacks the Z option, typing 111 (three consecutive 1's) would trigger the hotstring twice because the middle 1 is the *last* character of the first triggering but also the *first* character of the second triggering. By adding the letter Z in front of b0, you would have to type four 1's instead of three to trigger the hotstring twice. Use **Z0** to turn this option back off.

Long Replacements

Hotstrings that produce a large amount of replacement text can be made more readable and maintainable by using a [continuation section](#). For example:

```
::text1::  
(  
Any text between the top and bottom parentheses  
is treated literally, including commas and  
percent signs.  
By default, the hard carriage return (Enter)  
between the previous line and this one is also  
preserved.  
    By default, the indentation (tab) to the  
left of this line is preserved.  
  
See continuation section for how to change these  
default behaviors.  
)
```

The presence of a continuation section also causes the hotstring to default to [raw mode](#). The only way to override this special default is to specify the [r0 option](#) in each hotstring that has a continuation section (e.g. `:r0:text1::`).

Context-sensitive Hotstrings

The directives `#IfWinActive/Exist` can be used to make selected hotstrings context sensitive. Such hotstrings send a different replacement, perform a different action, or do nothing at all depending on the type of window that is active or exists. For example:

```
#IfWinActive ahk_class Notepad
::btw::This replacement text will appear only in
Notepad.
#IfWinActive
::btw::This replacement text appears in windows
other than Notepad.
```

AutoCorrect

The following script uses hotstrings to correct about 4700 common English misspellings on-the-fly. It also includes a   key to make it easy to add more misspellings:

Download: [AutoCorrect.ahk](#) (127 KB)

Author: [Jim Biancolo](#) and [Wikipedia's Lists of Common Misspellings](#)

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 [SendInput](#) command beneath the abbreviation, followed by a line containing only the word Return.

To send an extra space or tab after a replacement, include the space or tab at the end of the replacement but make the last character an accent/backtick (```). For example:

```
:*:btw::By the way `
```

By default, any click of the left or right mouse button will reset the hotstring recognizer. In other words, the script will begin waiting for an entirely new hotstring, eliminating from consideration anything you previously typed (if this is undesirable, specify the line `#Hotstring NOMouse` anywhere in the script). This "reset upon mouse click" behavior is the default because each click typically moves the text insertion point (caret) or sets keyboard focus to a new control/field. In such cases, it is usually desirable to: 1) fire a hotstring even if it lacks the [question mark option](#); 2) prevent a firing when something you type after clicking the mouse accidentally forms a valid abbreviation with what you typed before.

The hotstring recognizer checks the active window each time a character is typed, and resets if a different window is active than before. If the active window changes but reverts before any characters are typed, the change is not detected (but the hotstring recognizer may be reset for some other reason). The hotstring recognizer can also be reset by calling `Hotstring("Reset")`.

The built-in variable `A_EndChar` contains the ending character that you typed to trigger the most recent non-auto-replace hotstring. If no ending character was required (due to the [* option](#)), it will be blank. `A_EndChar` is useful when making hotstrings that use the `Send` command or whose behavior should vary depending on which ending character you typed. 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](#), backticks 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 special characters such as `{Enter}` are 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` ([escape sequences](#) for a complete list). For example, the hotstring `:* :ab` t::` would be triggered when you type "ab" followed by a tab.

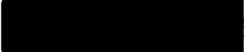
Spaces and tabs are treated literally within hotstring definitions. For example, the following would produce two different results: `::btw::` by the way and `::btw:: by the way`.

Each hotstring abbreviation can be no more than 40 characters long. The program will warn you if this length is exceeded. By contrast, the length of hotstring's replacement text is limited to about 5000 characters when the [sending mode](#) is at its default of `SendInput`. That limit can be increased to 16,383 characters by switching to one of the other [sending modes](#). Furthermore, an unlimited amount of text can be sent by using `SendPlay %MyVariable%` in the body of the hotstring.

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 you type, only the one listed first in the script will take effect. Related topic: [context-sensitive hotstrings](#).

Any backspacing you do is taken into account for the purpose of detecting hotstrings. However, the use of `↑` `←` `→` `PageUp` `PageDown` `Home` `End` `↩` gate 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 your keystrokes. In other words, the hotstring will still fire even though the triggering

abbreviation is never visible. In addition, you may still press 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. For example: `Gosub ::xyz`. However, jumping to a [single-line \(auto-replace\) hotstring](#) will do nothing other than execute a [return](#).

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.

By default, hotstrings are never triggered by keystrokes produced by any AutoHotkey script. This avoids the possibility of an infinite loop where hotstrings trigger each other over and over. In [v1.1.06] and later, this behaviour can be controlled with [#InputLevel](#) and [SendLevel](#). However, auto-replace hotstrings always use send level 0 and therefore never trigger [hook hotkeys](#) or hotstrings.

[v1.1.28+]: Hotstrings can be created dynamically by means of the [Hotstring](#) function, which can also modify, disable, or enable the script's existing hotstrings individually.

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

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](#).

Known limitation: On some systems in Java applications, hotstrings might interfere with the user's ability to type diacritical letters (via dead keys). To work around this, [Suspend](#) can be turned on temporarily (which disables all hotstrings).

Function Hotstrings [v1.1.28+]

One or more hotstrings can be assigned a [function](#) by simply defining it immediately after the hotstring label, as in this example:

```
; This example also demonstrates one way to
implement case conformity in a script.
:C:BTW:: ; Typed in all-caps.
:C:Btw:: ; Typed with only the first letter
upper-case.
: :btw:: ; Typed in any other combination.
    case_conform_btw() {
        hs := A_ThisHotkey ; For convenience
and in case we're interrupted.
        if (hs == ":C:BTW")
            Send BY THE WAY
        else if (hs == ":C:Btw")
            Send By the way
        else
            Send by the way
    }
```

For additional details, see [Function Hotkeys](#).

The [Hotstring](#) function can also be used to assign a function or function object to a hotstring.

Hotstring Helper

Andreas Borutta suggested the following script, which might be useful if you are a heavy user of hotstrings. By pressing   (another hotkey of your choice), the currently selected text can be turned into a hotstring. For example, if you have "by the way" selected in a word processor, pressing   will prompt you for its abbreviation (e.g. btw) and then add the new hotstring to the script. It will then reload the script to activate the hotstring.

Note: The [Hotstring](#) function can be used to create new hotstrings without reloading. Take a look at the [first example](#) in the example section of the function's page to see how this could be done.

```
#h:: ; Win+H hotkey
; Get the text currently selected. The clipboard
is used instead of
; "ControlGet Selected" because it works in a
greater variety of editors
; (namely word processors). Save the current
clipboard contents to be
; restored later. Although this handles only plain
text, it seems better
; than nothing:
AutoTrim Off ; Retain any leading and trailing
whitespace on the clipboard.
ClipboardOld = %ClipboardAll%
Clipboard = ; Must start off blank for detection
to work.
Send ^c
ClipWait 1
if ErrorLevel ; ClipWait timed out.
    return
; Replace CRLF and/or LF with `n for use in a
"send-raw" hotstring:
```

```

; The same is done for any other characters that
might otherwise
; be a problem in raw mode:
StringReplace, Hotstring, Clipboard, ``, ```, All
; Do this replacement first to avoid interfering
with the others below.
StringReplace, Hotstring, Hotstring, `r`n, ``r,
All ; Using `r works better than `n in MS Word,
etc.
StringReplace, Hotstring, Hotstring, `n, ``r, All
StringReplace, Hotstring, Hotstring, %A_Tab%, ``t,
All
StringReplace, Hotstring, Hotstring, `;, ```;, All
Clipboard = %ClipboardOld% ; Restore previous
contents of clipboard.
; This will move the InputBox's caret to a more
friendly position:
SetTimer, MoveCaret, 10
; Show the InputBox, providing the default
hotstring:
InputBox, Hotstring, New Hotstring, Type your
abbreviation at the indicated insertion point. You
can also edit the replacement text if you
wish.`n`nExample entry: :R:btw`:::by the
way,,,,,,,, :R:`:::%Hotstring%
if ErrorLevel ; The user pressed Cancel.
    return
IfInString, Hotstring, :R`:::
{
    MsgBox You didn't provide an abbreviation. The
hotstring has not been added.
    return
}
; Otherwise, add the hotstring and reload the
script:
FileAppend, `n%Hotstring%, %A_ScriptFullPath% ;
Put a `n at the beginning in case file lacks a

```

blank line at its end.

Reload

Sleep 200 ; If successful, the reload will close this instance during the Sleep, so the line below will never be reached.

MsgBox, 4,, The hotstring just added appears to be improperly formatted. Would you like to open the script for editing? Note that the bad hotstring is at the bottom of the script.

IfMsgBox, Yes, Edit

return

MoveCaret:

IfWinNotActive, New Hotstring

return

; Otherwise, move the InputBox's insertion point to where the user will type the abbreviation.

Send {Home}{Right 3}

SetTimer, MoveCaret, Off

return

Remapping Keys and Buttons

Introduction

Limitation: AutoHotkey's remapping feature described below is generally not as pure and effective as remapping directly via the Windows registry. For the advantages and disadvantages of each approach, see [registry remapping](#).

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 the `A` behave like the `B`:

```
a::b
```

The above example does not alter the `B` itself. The `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` and `Shift` are 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 not held down):

```
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:

MButton::Shift	Makes the middle button behave like the Shift key.
XButton1::LButton	Makes the fourth mouse button behave like the left mouse button.
RAlt::RButton	Makes the right Alt have like the right mouse button.

Other useful remappings:

CapsLock::Ctrl	<p>Makes CapsLock Control</p> <p>To retain the ability to turn CapsLock add the remapping <code>+CapsLock::CapsLock</code> first. This toggles CapsLock when you hold down the Shift press CapsLock</p> <p>Because both remappings allow additional modifier keys to be held down, the more specific <code>+CapsLock::CapsLock</code> remapping must be placed first for it to work.</p>
XButton2::^LButton	Makes the fifth mouse button (XButton2) produce a control-click.
RAlt::AppsKey	Makes the right Alt come the Menu (which is the key that opens the context menu).
RCtrl::RWin	Makes the right Control the right Win
Ctrl::Alt	Makes both Control like an Alt key. However, see alt-tab issues .
	Makes Control X Control

^x::^c	
RWin::Return	Disables the right  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 **Control** **Shift** holding the origin key will put that modifier into effect for the destination key. For example, **b::a** would produce **Control** **A** press **Control** **B**
- **CapsLock** acts 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).

Although a remapped key can trigger normal hotkeys, by default it cannot trigger mouse hotkeys or [hook hotkeys](#) (use [ListHotkeys](#) to discover which hotkeys are "hook"). For example, if the remapping **a::b** is in effect, pressing

Ctrl **Alt** **A** would trigger the **^!b** hotkey only if **^!b** is not a hook hotkey. If **^!b** is a hook hotkey, you can define **^!a** as a hotkey if you want **Ctrl** **Alt** **A** perform the same action as **Ctrl** **Alt** **B** for example:

```
a::b
^!a::
^!b::
ToolTip You pressed %A_ThisHotkey%.
return
```

Alternatively, in [v1.1.06] and later, [#InputLevel](#) can be used to override the default behaviour. For example:

```
#InputLevel 1
a::b

#InputLevel 0
^!b::
ToolTip You pressed %A_ThisHotkey%.
return
```

If [SendMode](#) is used in the auto-execute section (top part of the script), it affects all remappings. However, since remapping uses [Send {Blind}](#) and since the [SendPlay mode](#) does not fully support {Blind}, some remappings might not function properly in SendPlay mode (especially [Control](#) [Shift](#) [Alt](#) [Win](#)). Work around this, avoid SendPlay in auto-execute section when you have remappings; then use the command [SendPlay](#) vs. [Send](#) in other places throughout the script. Alternatively, you could translate your remappings into hotkeys (as described below) that explicitly call [SendEvent](#) vs. [Send](#).

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
```

However, the above hotkeys vary under the following circumstances:

1. When the source key is the left **Control** [redacted] destination key is an **Alt** [redacted] the line `Send {Blind}{LAlt DownR}` is replaced by `Send {Blind}{LCtrl up}{LAlt DownR}`. The same is true if the source is the right **Control** [redacted] that `{RCtrl up}` is used.
2. When a keyboard key is being remapped to become a mouse button (e.g. `RCtrl::RButton`), the hotkeys above use `SetMouseDelay` in place of `SetKeyDelay`. In addition, the first hotkey above is replaced by the following, which prevents the keyboard's auto-repeat feature from generating repeated mouse clicks:

```
*RCtrl::
```

```
SetMouseDelay -1
```

```
if not GetKeyState("RButton") ; i.e. the
right mouse button isn't down yet.
```

```
Send {Blind}{RButton DownR}
```

```
return
```

3. When the source is a [custom combination](#) in [v1.1.27.01+], the wildcard modifier (*) is omitted to allow the hotkeys to work.

Prior to [v1.1.27], [DownTemp](#) was used instead of [DownR](#).

Note that `SetKeyDelay`'s second parameter ([press duration](#)) is omitted in the hotkeys above. This is because `press-duration` does not apply to down-only or up-only events such as `{b down}` and `{b up}`. However, it does apply to changes in the state of the `Shift` `Ctrl` `Alt` `Win` `████` which affects remappings such as `a::B` or `a::^b`. Consequently, any `press-duration` a script puts into effect via its [auto-execute section](#) will apply to all such remappings.

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
```

Alt-tab issues: If you remap a key or mouse button to become an `Alt` `████` that key will probably not be able to alt-tab properly. A possible work-around is to add the hotkey `*Tab::Send {Blind}{Tab}` -- but be aware that it will likely interfere with using the real `Alt` `████` alt-tab. Therefore, it should be used only when you alt-tab solely by means of remapped keys and/or [alt-tab hotkeys](#).

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 the `F1` `████`

The following keys are not supported by the built-in remapping method:

- The mouse wheel (WheelUp/Down/Left/Right).
- Pause and Break as destination keys (since they match the names of commands).

- Curly braces {} as destination keys. Instead use the [VK/SC method](#); e.g. `x::+sc01A` and `y::+sc01B`.
- A percent sign (%) as a destination key. Instead use the [VK/SC method](#).
- "Return" as a destination key. Instead use "Enter".

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
```

Remapping via the Registry's "Scancode Map"

Advantages:

- Registry remapping is generally more pure and effective than [AutoHotkey's remapping](#). For example, it works in a broader variety of games, it has no known [alt-tab issues](#), and it is capable of firing AutoHotkey's hook hotkeys (whereas AutoHotkey's remapping requires a [workaround](#)).
- If you choose to make the registry entries manually (explained below), absolutely no external software is needed to remap your keyboard. Even if you use [KeyTweak](#) to make the registry entries for you, KeyTweak does not need to stay running all the time (unlike AutoHotkey).

Disadvantages:

- Registry remapping is relatively permanent: a reboot is required to undo the changes or put new ones into effect.
- Its effect is global: it cannot create remappings specific to a particular user, application, or locale.
- It cannot send keystrokes that are modified by    . For example, it cannot remap a lowercase character to an uppercase one.
- It supports only the keyboard (AutoHotkey has [mouse remapping](#) and some [limited joystick remapping](#)).

How to Apply Changes to the Registry: There are at least two methods to remap keys via the registry:

1. Use a program like [KeyTweak](#) (freeware) to visually remap your keys. It will change the registry for you.
2. Remap keys manually by creating a .reg file (plain text) and loading it into the registry. This is demonstrated at www.autohotkey.com/forum/post-56216.html#56216

Related Topics

[List of keys and mouse buttons](#) [GetKeyState](#)
[Remapping a joystick](#)

List of Keys, Mouse Buttons, and Joystick Controls

Mouse

General buttons

LButton	Left mouse button
RButton	Right mouse button
MButton	Middle or wheel mouse button

Advanced buttons

XButton1	4th mouse button. Typically performs the same function as Browser_Back.
XButton2	5th mouse button. Typically performs the same function as Browser_Forward.

Wheel

WheelDown	Turn the wheel downward (toward you).
WheelUp	Turn the wheel upward (away from you).
WheelLeft WheelRight	<p>[v1.0.48+]: Scroll to the left or right.</p> <p>Requires Windows Vista or later. These can be used as hotkeys with some (but not all) mice which have a second wheel or support tilting the wheel to either side. In some cases, software bundled with the mouse must instead be used to control this feature.</p>

Regardless of the particular mouse, [Send](#) and [Click](#) can be used to scroll horizontally in programs which support it.

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.

Although any single character can be used as a key name, its meaning (scan code or virtual keycode) depends on the current keyboard layout. Additionally, some special characters may need to be escaped or enclosed in braces, depending on the context. [v1.1.27+]: The letters a-z or A-Z can be used to refer to the corresponding virtual keycodes (usually vk41-vk5A) even if they are not included in the current keyboard layout.

General keys

CapsLock	CapsLock
Space	Space
Tab	Tab (separator key)
Enter (or Return)	Enter
Escape (or Esc)	Esc
Backspace (or BS)	Backspace

Cursor control keys

ScrollLock	ScrollLock
Delete (or Del)	Delete
Insert (or Ins)	Insert
Home	Home

End	End [redacted]
PgUp	PageUp [redacted] (key)
PgDn	PageDown [redacted] (key)
Up	↑ [redacted] (arrow key)
Down	↓ [redacted] (down arrow key)
Left	← [redacted] (left arrow key)
Right	→ [redacted] (right arrow key)

Numpad keys

Due to system behavior, the following keys separated by a slash are identified differently depending on whether NumLock [redacted] is ON. If NumLock [redacted] is OFF but Shift [redacted], the system temporarily releases Shift [redacted] acts as though NumLock [redacted]

Numpad0 / NumpadIns	0 [redacted] Insert [redacted]
Numpad1 / NumpadEnd	1 [redacted] End [redacted]
Numpad2 / NumpadDown	2 [redacted] ↓ [redacted]
Numpad3 / NumpadPgDn	3 [redacted] PageDown [redacted]
Numpad4 / NumpadLeft	4 [redacted] ← [redacted]
Numpad5 / NumpadClear	5 [redacted] typically does nothing
Numpad6 / NumpadRight	6 [redacted] → [redacted]
Numpad7 / NumpadHome	7 [redacted] Home [redacted]
Numpad8 / NumpadUp	8 [redacted] ↑ [redacted]

Numpad9 / NumpadPgUp	9 [] PageUp []
NumpadDot / NumpadDel	. [] Delete []
NumLock	NumLock [] (NumLock key)
NumpadDiv	/ [] (Division)
NumpadMult	* [] (Multiplication)
NumpadAdd	+ [] (Addition)
NumpadSub	- [] (Subtraction)
NumpadEnter	Enter []

Function keys

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

Modifier keys

LWin	Left [] Win [] responds to the <# hotkey prefix.
RWin	Right [] Win [] responds to the ># hotkey prefix. <div style="background-color: #e0ffe0; padding: 10px; border: 1px solid #c0ffc0;"> <p>Note: Unlike [] Control [] / [] Alt [] / [] Shift [], there is no generic/neutral "Win" key because the OS does not support it. However, hotkeys with the # modifier can be triggered by either [] Win [] key.</p> </div>

Control (or Ctrl)	<p>Control [redacted] (Control::) it fires upon release unless it has the tilde prefix. Corresponds to the ^ hotkey prefix.</p>
Alt	<p>Alt [redacted] hotkey (Alt::) it fires upon release unless it has the tilde prefix. Corresponds to the ! hotkey prefix.</p>
Shift	<p>Shift [redacted] key (Shift::) it fires upon release unless it has the tilde prefix. Corresponds to the + hotkey prefix.</p>
LControl (or LCtrl)	<p>Left Control [redacted] s to the <^ hotkey prefix.</p>
RControl (or RCtrl)	<p>Right Control [redacted] s to the >^ hotkey prefix.</p>
LShift	<p>Left Shift [redacted] nds to the <+ hotkey prefix.</p>
RShift	<p>Right Shift [redacted] nds to the >+ hotkey prefix.</p>
LAlt	<p>Left Alt [redacted] sponds to the <! hotkey prefix.</p>
RAlt	<p>Right Alt [redacted] sponds to the >! hotkey prefix.</p> <div style="background-color: #e0ffe0; padding: 10px; margin-top: 10px;"> <p>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.</p> </div>

Multimedia keys

The function assigned to each of the keys listed below can be overridden by modifying the Windows registry. This table shows the default function of each key on most versions of Windows.

Browser_Back	Back
Browser_Forward	Forward
Browser_Refresh	Refresh
Browser_Stop	Stop
Browser_Search	Search
Browser_Favorites	Favorites
Browser_Home	Homepage
Volume_Mute	Mute the volume
Volume_Down	Lower the volume
Volume_Up	Increase the volume
Media_Next	Next Track
Media_Prev	Previous Track
Media_Stop	Stop
Media_Play_Pause	Play/Pause
Launch_Mail	Launch default e-mail program
Launch_Media	Launch default media player
Launch_App1	Launch My Computer
Launch_App2	Launch Calculator

Special keys

AppsKey	<code>Menu</code> [redacted] the key that invokes the right-click context menu.
PrintScreen	<code>PrintScreen</code> [redacted]
CtrlBreak	<code>Ctrl</code> [redacted] <code>Break</code> [redacted]
Pause	<code>Pause</code> [redacted]
Break	<code>Break</code> [redacted] is synonymous with Pause, use <code>^CtrlBreak</code> in hotkeys instead of <code>^Pause</code> or <code>^Break</code> .
Help	<code>Help</code> [redacted] probably doesn't exist on most keyboards. It's usually not the same as <code>F1</code> [redacted]
Sleep	<code>Sleep</code> [redacted] the sleep key on some keyboards might not work with this.
SCnnn	Specify for nnn the scan code of a key. Recognizes unusual keys not mentioned above. See Special Keys for details.
	<p>Specify for nn the hexadecimal virtual key code of a key. This rarely-used method also prevents certain types of hotkeys from requiring the keyboard hook. For example, the following hotkey does not use the keyboard hook, but as a side-effect it is triggered by pressing <i>either</i> <code>Home</code> [redacted]</p> <p>NumpadHome:</p> <pre>^VK24::MsgBox You pressed Home or NumpadHome while holding down Control.</pre>

VKnn

Known limitation: VK hotkeys that are forced to use the [keyboard hook](#), such as `*VK24` or `~VK24`, will fire for only one of the keys, not both (e.g. NumpadHome but not `Home` ██████████). For more information about the VKnn method, see [Special Keys](#).

Warning: Only `Send`, `GetKeyName`, `GetKeyVK`, `GetKeySC` and `#MenuMaskKey` support combining VKnn and SCnnn. [v1.1.27+]: The presence of an invalid suffix prevents VKnn from being recognized. For example, `vk1BSC001::` raises an error in v1.1.27+, but `SC001` was ignored (had no effect) in previous versions.

Joystick

Joy1 through Joy32: The buttons of the joystick. To help determine the button numbers for your joystick, use this [test script](#). Note that [hotkey prefix symbols](#) such as ^ (control) and + (shift) are not supported (though [GetKeyState](#) can be used as a substitute). 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 joystick and you want to use one beyond the first, include the joystick number (max 16) in front of the control name. For example, 2joy1 is the second joystick's first button.

Note: If you have trouble getting a script to recognize your joystick, one person reported needing to specify a joystick number other than 1 even though only a single joystick was present. It is unclear how this situation arises or whether it is normal, but experimenting with the joystick number in the [joystick test script](#) can help determine if this applies to your system.

See Also:

Joystick remapping: methods of sending keystrokes and mouse clicks with a joystick.

Joystick-To-Mouse script: using a joystick as a mouse.

Hand-held Remote Controls

Respond to signals from hand-held remote controls via the [WinLIRC client script](#).

Special Keys

If your keyboard or mouse has a key not listed above, you might still be able to make it a hotkey by using the following steps:

1. Ensure that at least one script is running that is using the [keyboard hook](#). You can tell if a script has the keyboard hook by opening its main window and selecting "View->Key history" from the menu bar.
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 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 directly make that particular key a hotkey because your keyboard driver or hardware handles it at a level too low for AutoHotkey to access. For possible solutions, see further below.
6. If your key is detectable, make a note of the 3-digit hexadecimal 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_ThisHotkey% was pressed.
return
```

Reverse direction: To remap some other key to *become* a "mystery key", follow this example:

```
; 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 the Key History screen.
#c::Send {vkFFsc159}
```

Alternate solutions: If your key or mouse button is not detectable by the [Key](#)

[History](#) screen, one of the following might help:

1. Reconfigure the software that came with your mouse or keyboard (sometimes accessible in the Control Panel or Start Menu) to have the "mystery key" send some other keystroke. Such a keystroke can then be defined as a hotkey in a script. For example, if you configure a mystery key to send `Control` `F1` can then indirectly make that key as a hotkey by using `^F1::` in a script.
2. Try [AHKHID](#). You can also try searching the [forum](#) for a keywords like `RawInput*`, `USB HID` or `AHKHID`.
3. The following is a last resort and generally should be attempted only in desperation. This is because the chance of success is low and it may cause unwanted side-effects that are difficult to undo:
Disable or remove any extra software that came with your keyboard or mouse or change its driver to a more standard one such as the one built into the OS. This assumes there is such a driver for your particular keyboard or mouse and that you can live without the features provided by its custom driver and software.

Scripts

Related topics:

- [Using the Program](#): How to use AutoHotkey, in general.
- [Concepts and Conventions](#): General explanation of various concepts utilised by AutoHotkey.
- [Scripting Language](#): Specific details about syntax (how to write scripts).

Table of Contents

- [Introduction](#)
- [The Top of the Script \(the Auto-execute Section\)](#): This portion executes automatically when the script starts.
- [Splitting a Long Line into a Series of Shorter Ones](#): This can improve a script's readability and maintainability.
- [Convert a Script to an EXE \(ahk2exe\)](#): Convert a .ahk script into a .exe file that can run on any PC.
- [Passing Command Line Parameters to a Script](#): The variables %1%, %2%, etc. contain the incoming parameters.
- [Script File Codepage](#): Using non-ASCII characters safely in scripts.
- [Debugging a Script](#): How to find the flaws in a misbehaving script.

Introduction

Each script is a plain text file containing lines to be executed by the program (AutoHotkey.exe). A script may also contain [hotkeys](#) and [hotstrings](#), or even consist entirely of them. However, in the absence of hotkeys and hotstrings, a script will perform its commands sequentially from top to bottom the moment it is launched.

The program loads the script into memory line by line, and each line may be up to 16,383 characters long. During loading, the script is [optimized](#) and validated. Any syntax errors will be displayed, and they must be corrected before the script can run.

The Top of the Script (the Auto-execute Section)

After the script has been loaded, it begins executing at the top line, continuing until a [Return](#), [Exit](#), [hotkey/hotstring label](#), or the physical end of the script is encountered (whichever comes first). This top portion of the script is referred to as the *auto-execute* section.

Note: While the script's *first* hotkey/hotstring label has the same effect as [return](#), other hotkeys and labels do not.

A script that is not [persistent](#) and that lacks [hotkeys](#), [hotstrings](#), [OnMessage](#), and [GUI](#) 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 [thread](#) launched by a [hotkey](#), [hotstring](#), [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](#), [SendMode](#), [SetKeyDelay](#), [SetMouseDelay](#), [SetWinDelay](#), [SetControlDelay](#), [SetDefaultMouseSpeed](#), [CoordMode](#), [SetStoreCapsLockMode](#), [AutoTrim](#), [SetFormat](#), [StringCaseSense](#), [Thread](#), and [Critical](#).

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. When the auto-execute section finally completes (if ever), the defaults are updated again to be those that were in effect at the end of the auto-execute section. 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](#).

Splitting a Long Line into a Series of Shorter Ones

Long lines can be divided up into a collection of smaller ones to improve readability and maintainability. This does not reduce the script's execution speed because such lines are merged in memory the moment the script launches.

Method #1: A line that starts with "and", "or", ||, &&, a comma, or a [period](#) is automatically merged with the line directly above it (in v1.0.46+, the same is true for all other [expression operators](#) except ++ and --). In the following example, the second line is appended to the first because it begins with a comma:

```
FileAppend, This is the text to append.`n    ; A  
comment is allowed here.  
,  
%A_ProgramFiles%\SomeApplication\LogFile.txt    ;  
Comment.
```

Similarly, the following lines would get merged into a single line because the last two start with "and" or "or":

```
if (Color = "Red" or Color = "Green" or Color =  
"Blue"    ; Comment.  
    or Color = "Black" or Color = "Gray" or  
Color = "White")    ; Comment.  
    and ProductIsAvailableInColor(Product,  
Color)    ; Comment.
```

The [ternary operator](#) is also a good candidate:

```
ProductIsAvailable := (Color = "Red")  
    ? false    ; We don't have any red products,  
so don't bother calling the function.  
    : ProductIsAvailableInColor(Product, Color)
```

Although the indentation used in the examples above is optional, it might improve clarity by indicating which lines belong to ones above them. Also, it is not necessary to include extra spaces for lines starting with the words "AND" and "OR"; the program does this automatically. Finally, blank lines or [comments](#) may be added between or at the end of any of the lines in the above examples.

Method #2: This method should be used to merge a large number of lines or when the lines are not suitable for Method #1. Although this method is especially useful for [auto-replace hotstrings](#), it can also be used with any command or [expression](#). For example:

```
; EXAMPLE #1:
Var =
(
Line 1 of the text.
Line 2 of the text. By default, a linefeed (`n)
is present between lines.
)

; EXAMPLE #2:
FileAppend, ; The comma is required in this
case.
(
A line of text.
By default, the hard carriage return (Enter)
between the previous line and this one will be
written to the file as a linefeed (`n).
    By default, the tab to the left of this line
will also be written to the file (the same is
true for spaces).
By default, variable references such as %Var%
are resolved to the variable's contents.
), C:\My File.txt
```

In the examples above, a series of lines is bounded at the top and bottom by a pair of parentheses. This is known as a *continuation section*. Notice that the bottom line contains [FileAppend](#)'s last parameter after the closing parenthesis.

This practice is optional; it is done in cases like this so that the comma will be seen as a parameter-delimiter rather than a literal comma.

The default behavior of a continuation section can be overridden by including one or more of the following options to the right of the section's opening parenthesis. If more than one option is present, separate each one from the previous with a space. For example: `(LTrim Join| %`.

Join: Specifies how lines should be connected together. If this option is omitted, each line except the last will be followed by a linefeed character (`\n`). If the word *Join* is specified by itself, lines are connected directly to each other without any characters in between. Otherwise, the word *Join* should be followed immediately by as many as 15 characters. For example, `Join`s` would insert a space after each line except the last ("``s`" indicates a literal space -- it is a special escape sequence recognized only by *Join*). Another example is `Join`r`n`, which inserts CR+LF between lines. Similarly, `Join|` inserts a pipe between lines. To have the final line in the section also ended by a join-string, include a blank line immediately above the section's closing parenthesis.

Known limitation: If the Join string ends with a colon, it must not be the last option on the line. For example, `(Join:` is treated as the label "(Join" and `(LTrim Join:` is unsupported, but `(Join: C` is okay.

LTrim: Omits spaces and tabs at the beginning of each line. This is primarily used to allow the continuation section to be indented. Also, this option may be turned on for multiple continuation sections by specifying `#LTrim` on a line by itself. `#LTrim` is positional: it affects all continuation sections physically beneath it. The setting may be turned off via `#LTrim Off`.

RTrim0 (RTrim followed by a zero): Turns off the omission of spaces and tabs from the end of each line.

Comments (or **Comment** or **Com** or **C**) [v1.0.45.03+]: Allows [semicolon comments](#) inside the continuation section (but not `/* . . */`). Such comments (along with any spaces and tabs to their left) are entirely omitted from the joined result rather than being treated as literal text. Each comment can appear to the right of a line or on a new line by itself.

% (percent sign): Treats percent signs as literal rather than as variable

references. This avoids the need to [escape](#) each percent sign to make it literal. This option is not needed in places where percent signs are already literal, such as [auto-replace hotstrings](#).

, (comma): Treats commas as delimiters rather than as literal commas. This rarely-used option is necessary only for the commas between command parameters because in [function calls](#), the type of comma does not matter. Also, this option transforms only those commas that actually delimit parameters. In other words, once the command's final parameter is reached (or there are no parameters), subsequent commas are treated as literal commas regardless of this option.

` (accent): Treats each backtick character literally rather than as an [escape character](#). This also prevents commas and percent signs from being explicitly and individually escaped. In addition, it prevents the translation of any explicitly specified escape sequences such as ``r` and ``t`.

) [[v1.1.01+](#)]: If a closing parenthesis appears in the continuation section's options (except as a parameter of the [Join](#) option), the line is reinterpreted as an expression instead of the beginning of a continuation section. This allows expressions like `(x.y)[z]()` to work without the need to escape the opening parenthesis.

Remarks

[Escape sequences](#) such as ``n` (linefeed) and ``t` (tab) are supported inside the continuation section except when the [accent \(`\) option](#) has been specified.

When the [comment option](#) is absent, semicolon and `/*..*/` comments are not supported within the interior of a continuation section because they are seen as literal text. However, comments can be included on the bottom and top lines of the section. For example:

```
FileAppend,      ; Comment.
; Comment.
( LTrim Join     ; Comment.
    ; This is not a comment; it is literal.
Include the word Comments in the line above to
make it a comment.
```

```
), C:\File.txt ; Comment.
```

As a consequence of the above, semicolons never need to be [escaped](#) within a continuation section.

A continuation section cannot produce a line whose total length is greater than 16,383 characters (if it tries, the program will alert you the moment the script is launched). One way to work around this is to do a series of concatenations into a variable. For example:

```
Var =  
(  
...  
)  
Var = %Var%`n ; Add more text to the variable  
via another continuation section.  
(  
...  
)  
FileAppend, %Var%, C:\My File.txt
```

Since a closing parenthesis indicates the end of a continuation section, to have a line start with literal closing parenthesis, precede it with an accent/backtick: ``)`.

A continuation section can be immediately followed by a line containing the open-parenthesis of another continuation section. This allows the options mentioned above to be varied during the course of building a single line.

The piecemeal construction of a continuation section by means of [#Include](#) is not supported.

Convert a Script to an EXE (ahk2exe)

A script compiler (courtesy of fins) is included with the program.

Once a script is compiled, it becomes a standalone executable; that is, AutoHotkey.exe is not required in order to run the script. The compilation process creates an executable file which contains the following: the AutoHotkey interpreter, 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" (only available if the script compiler option was chosen when AutoHotkey was installed). This creates an EXE file of the same base filename as the script, which appears after a short time in the same directory. Note: The EXE file is produced using the same custom icon, .bin file and use [MPRESS](#) setting that were last used by Method #1 above.
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] [/bin  
AutoHotkeySC.bin] [/mpress 0or1]
```

For example:

```
Ahk2Exe.exe /in "MyScript.ahk" /icon  
"MyIcon.ico"
```

Usage:

- 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.

Notes:

- Compiling does not typically improve the performance of a script.
- As of v1.1.01, password protection and the /NoDecompile switch are not supported.
- The commands `#NoTrayIcon` and "`Menu, Tray, ShowMainWindow`" affect the behavior of compiled scripts.
- 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. [Compile_AHK II](#) can be used to facilitate this process. As of v1.1.01, the compiled script can be edited instead of AutoHotkeySC.bin.
- The method above can also be used to change existing icons or add new ones to all compiled scripts.
- The built-in variable `A_IsCompiled` contains 1 if the script is running in compiled form. Otherwise, it is blank.
- When parameters are passed to Ahk2Exe, a message indicating the success or failure of the compiling process is written to stdout. Although the message will not appear at the command prompt, it can be "caught" by means such as redirecting output to a file. [v1.0.43+]
- Additionally in the case of a failure, Ahk2Exe has exit codes indicating the kind of error that occurred. These error codes can be found at [GitHub \(ErrorCodes.md\)](#). [v1.1.22.03+]

The compiler's source code and newer versions can be found at [GitHub](#).

Compressing Compiled Scripts

Ahk2Exe optionally uses MPRESS (a freeware program by MATCODE Software) to compress compiled scripts. If `mpress.exe` is present in the "Compiler" subfolder where AutoHotkey was installed, it is used automatically unless it is disabled via `/mpress 0` or the GUI setting.

Official website (was offline in March 2016):

<http://www.matcode.com/mpress.htm>

Mirror (downloads and information): <https://autohotkey.com/mpress/>

Note: While compressing the script executable prevents casual inspection of the script's source code using a plain text editor like Notepad or a PE resource editor, it does not prevent the source code from being extracted by tools dedicated to that purpose.

Passing Command Line Parameters to a Script

Scripts support command line parameters. The format is:

```
AutoHotkey.exe [Switches] [Script Filename]
[Script Parameters]
```

And for compiled scripts, the format is:

```
CompiledScript.exe [Switches] [Script
Parameters]
```

Switches: Zero or more of the following:

Switch	Meaning	Works compiled?
/f or /force	Launch unconditionally, skipping any warning dialogs. This has the same effect as #SingleInstance Off .	Yes
/r or /restart	Indicate that the script is being restarted (this is also used by the Reload command, internally).	Yes
/ErrorStdOut	Send syntax errors that prevent a script from launching to stderr rather than displaying a dialog. See #ErrorStdOut for details. This can be combined with /iLib to validate the script without running it.	Yes
/Debug	[AHK_L 11+]: Connect to a debugging client. For more details, see Interactive Debugging .	No
/CPn	[AHK_L 51+]: Overrides the default codepage used to read script files. For more details, see Script File Codepage .	No

<p><code>/iLib</code> <code>"OutFile"</code></p>	<p>[v1.0.47+]: AutoHotkey loads the script but does not run it. For each script file which is auto-included via the library mechanism, two lines are written to the file specified by <i>OutFile</i>. These lines are written in the following format, where <i>LibDir</i> is the full path of the Lib folder and <i>LibFile</i> is the filename of the library:</p> <pre data-bbox="511 537 1146 726">#Include LibDir\ #IncludeAgain LibDir\LibFile.ahk</pre> <p>If the output file exists, it is overwritten. <i>OutFile</i> can be <code>*</code> to write the output to stdout.</p> <p>If the script contains syntax errors, the output file may be empty. The process exit code can be used to detect this condition; if there is a syntax error, the exit code is 2. The <code>/ErrorStdOut</code> switch can be used to suppress or capture the error message.</p>	<p>No</p>
--	---	-----------

Script Filename: This can be omitted if there are no *Script Parameters*. If omitted (such as if you run AutoHotkey directly from the Start menu), the program looks for a script file called `AutoHotkey.ahk` in the following locations, in this order:

- The directory which contains the [AutoHotkey executable](#).
- The current user's [Documents](#) folder.

The filename `AutoHotkey.ahk` depends on the name of the executable used to run the script. For example, if you rename AutoHotkey.exe to MyScript.exe, it will attempt to find `MyScript.ahk`. If you run AutoHotkeyU32.exe without parameters, it will look for AutoHotkeyU32.ahk.

Note: In old versions prior to [revision 51](#), the program looked for AutoHotkey.ini in the working directory or AutoHotkey.ahk in My Documents.

[v1.1.17+]: Specify an asterisk (*) for the filename to read the script text from standard input (stdin). For an example, see [ExecScript\(\)](#).

Script Parameters: The string(s) you want to pass into the script, with each separated from the next by a space. Any parameter that contains spaces should be enclosed in quotation marks. A literal quotation mark may be passed in by preceding it with a backslash (\"). Consequently, any trailing slash in a quoted parameter (such as "C:\My Documents\") is treated as a literal quotation mark (that is, the script would receive the string C:\My Documents"). To remove such quotes, use `StringReplace, 1, 1, "", All`.

[v1.1.27+]: Incoming parameters, if present, are stored as an array in the built-in variable `A_Args`, and can be accessed using [array syntax](#). `A_Args[1]` contains the first parameter. The following example exits the script when too few parameters are passed to it:

```
if A_Args.Length() < 3
{
    MsgBox % "This script requires at least 3
parameters but it only received "
A_Args.Length() "."
    ExitApp
}
```

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

```
for n, param in A_Args ; For each parameter:
{
    MsgBox Parameter number %n% is %param%.
}
```

If the parameters are file names, the following example can be used to convert them to their case-corrected long names (as stored in the file system), including complete/absolute path:

```

for n, GivenPath in A_Args ; For each parameter
(or file dropped onto a script):
{
    Loop Files, %GivenPath%, FD ; Include files
and directories.
        LongPath := A_LoopFileFullPath
        MsgBox The case-corrected long path name of
file`n%GivenPath%`nis:`n%LongPath%
}

```

Known limitation: dragging files onto a .ahk script may fail to work properly if 8-dot-3 (short) names have been turned off in an NTFS file system. One work-around is to [compile](#) the script then drag the files onto the resulting EXE.

Legacy: The command line parameters are also stored in the [variables](#) %1%, %2%, and so on, as in versions prior to [v1.1.27]. In addition, %0% contains the number of parameters passed (0 if none). However, these variables cannot be referenced directly in an expression because they would be seen as numbers rather than variables. The following example exits the script when too few parameters are passed to it:

```

if 0 < 3 ; The left side of a non-expression
if-statement is always the name of a variable.
{
    MsgBox This script requires at least 3
incoming parameters but it only received %0%.
    ExitApp
}

```

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

```

Loop, %0% ; For each parameter:
{

```

```

    param := %A_Index% ; Fetch the contents of
the variable whose name is contained in A_Index.
    MsgBox, 4,, Parameter number %A_Index% is
%param%. Continue?
    IfMsgBox, No
        break
}

```

If the parameters are file names, the following example can be used to convert them to their case-corrected long names (as stored in the file system), including complete/absolute path:

```

Loop %0% ; For each parameter (or file dropped
onto a script):
{
    GivenPath := %A_Index% ; Fetch the contents
of the variable whose name is contained in
A_Index.
    Loop %GivenPath%, 1
        LongPath = %A_LoopFileLongPath%
    MsgBox The case-corrected long path name of
file`n%GivenPath%`nis:`n%LongPath%
}

```

Script File Codepage [AHK_L 51+]

The characters a script file may contain are restricted by the codepage used to load the file.

- If the file begins with a UTF-8 or UTF-16 (LE) byte order mark, the appropriate codepage is used and the `/CPn` switch is ignored.
- If the `/CPn` switch is passed on the command-line, codepage `n` is used. For a list of valid numeric codepage identifiers, see [MSDN](#).
- In all other cases, the system default ANSI codepage is used.

Note that this applies only to script files loaded by AutoHotkey, not to file I/O within the script itself. [FileEncoding](#) controls the default encoding of files read or written by the script, while [IniRead](#) and [IniWrite](#) always deal in UTF-16 or ANSI.

As all text is converted (where necessary) to the [native string format](#), characters which are invalid or don't exist in the native codepage are replaced with a placeholder: ANSI '?' or Unicode '❖'. In Unicode builds, this should only occur if there are encoding errors in the script file or the codepages used to save and load the file don't match.

[RegWrite](#) may be used to set the default for scripts launched from Explorer (e.g. by double-clicking a file):

```
; Uncomment the appropriate line below or leave
them all commented to
; reset to the default of the current build.
Modify as necessary:
; codepage = 0           ; System default ANSI
codepage
; codepage = 65001      ; UTF-8
; codepage = 1200       ; UTF-16
; codepage = 1252       ; ANSI Latin 1; Western
European (Windows)
if (codepage != "")
    codepage := " /CP" . codepage
```

```
cmd="%A_AhkPath%"%codepage% "`%1" `%*
key=AutoHotkeyScript\Shell\Open\Command
if A_IsAdmin      ; Set for all users.
    RegWrite, REG_SZ, HKCR, %key%, , %cmd%
else              ; Set for current user only.
    RegWrite, REG_SZ, HKCU,
Software\Classes\%key%, , %cmd%
```

This assumes AutoHotkey has already been installed. Results may be less than ideal if it has not.

Debugging a Script

Commands such as [ListVars](#) and [Pause](#) can help you debug a script. For example, the following two lines, when temporarily inserted at carefully chosen positions, create "break points" in the script:

```
ListVars
Pause
```

When the script encounters these two lines, it will display the current contents of all variables for your inspection. When you're ready to resume, un-pause the script via the File or Tray menu. The script will then continue until reaching the next "break point" (if any).

It is generally best to insert these "break points" at positions where the active window does not matter to the script, such as immediately before a `WinActivate` command. This allows the script to properly resume operation when you un-pause it.

The following commands are also useful for debugging: [ListLines](#), [KeyHistory](#), and [OutputDebug](#).

Some common errors, such as typos and missing "global" declarations, can be detected by [enabling warnings](#).

Interactive Debugging [AHK_L 11+]

Interactive debugging is possible with a supported [DBGp client](#). Typically the following actions are possible:

- Set and remove breakpoints on lines - pause execution when a [breakpoint](#) is reached.
- Step through code line by line - step into, over or out of functions and subroutines.
- Inspect all variables or a specific variable.
- View the stack of running subroutines and functions.

Note that this functionality is disabled for compiled scripts.

To enable interactive debugging, first launch a supported debugger client then launch the script with the **/Debug** command-line switch.

```
AutoHotkey.exe /Debug[=SERVER:PORT] ...
```

SERVER and *PORT* may be omitted. For example, the following are equivalent:

```
AutoHotkey /Debug "myscript.ahk"  
AutoHotkey /Debug=localhost:9000 "myscript.ahk"
```

[AHK_L 59+]: To attach the debugger to a script which is already running, send it a message as shown below:

```
ScriptPath = ; SET THIS TO THE FULL PATH OF THE  
SCRIPT  
DetectHiddenWindows On  
IfWinExist %ScriptPath% ahk_class AutoHotkey  
    ; Optional parameters:  
    ; wParam = the IPv4 address of the  
debugger client, as a 32-bit integer.  
    ; lParam = the port which the debugger  
client is listening on.  
    PostMessage DllCall("RegisterWindowMessage",  
"str", "AHK_ATTACH_DEBUGGER")
```

Once the debugger client is connected, it may detach without terminating the script by sending the "detach" DBGp command.

Script Showcase

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

Variables and Expressions

Table of Contents

- Variables
- Expressions
- Operators in Expressions
- Built-in Variables
- Variable Capacity and Memory

Variables

See [Variables](#) for general explanation and details about how variables work.

Storing values in variables: To store a string or number in a variable, there are two methods: [legacy](#) and [expression](#). The legacy method uses the [equal sign operator \(=\)](#) to assign [unquoted literal strings](#) or variables enclosed in percent signs. For example:

```
MyNumber = 123
MyString = This is a literal string.
CopyOfVar = %Var% ; With the = operator,
percent signs are required to retrieve a
variable's contents.
```

By contrast, the expression method uses the [colon-equal operator \(:=\)](#) to store numbers, [quoted strings](#), and other types of [expressions](#). The following examples are functionally identical to the previous ones:

```
MyNumber := 123
MyString := "This is a literal string."
CopyOfVar := Var ; Unlike its counterpart in
the previous section, percent signs are not used
with the := operator.
```

The latter method is preferred by many due to its greater clarity, and because it supports an [expression syntax](#) nearly identical to that in many other languages.

You may have guessed from the above that there are two methods to erase the contents of a variable (that is, to make it blank):

```
MyVar =
MyVar := ""
```

The empty pair of quotes above should be used only with the := operator because if it were used with the = operator, it would store two literal quote-characters inside the variable.

Retrieving the contents of variables: Like the two methods of storing values, there are also two methods for retrieving them: [legacy](#) and [expression](#). The legacy method requires that each variable name be enclosed in percent signs to retrieve its contents. For example:

```
MsgBox The value in the variable named Var is
%Var%.
CopyOfVar = %Var%
```

By contrast, the expression method omits the percent signs around variable names, but encloses [literal strings](#) in quotes. Thus, the following are the expression equivalents of the previous examples:

```
MsgBox % "The value in the variable named Var is
" . Var . "." ; A period is used to concatenate
(join) two strings.
CopyOfVar := Var
```

In the MsgBox line above, [a percent sign and a space](#) is used to change the parameter from legacy to expression mode. This is necessary because the legacy method is used by default by all commands, except where otherwise documented.

Comparing variables: Please read the expressions section below for important notes about the different kinds of comparisons, especially about when to use parentheses.

Expressions

See [Expressions](#) for a structured overview and further explanation.

Expressions are used to perform one or more operations upon a series of variables, literal strings, and/or literal numbers.

Variable names in an expression are not enclosed in percent signs (except for [pseudo-arrays](#) and other [double references](#)). Consequently, literal strings must be enclosed in double quotes to distinguish them from variables. For example:

```
if (CurrentSetting > 100 or FoundColor <>
    "Blue")
    MsgBox The setting is too high or the wrong
    color is present.
```

In the example above, "Blue" appears in quotes because it is a literal string. To include an *actual* quote-character inside a literal string, specify two consecutive quotes as shown twice in this example: "She said, ""An apple a day. """.

Important: An if-statement that contains an expression is differentiated from a [traditional if-statement](#) such as `If FoundColor <> Blue` by making the character after the word "if" an open-parenthesis. Although this is usually accomplished by enclosing the entire expression in parentheses, it can also be done with something like `if (x > 0) and (y > 0)`. In addition, the open-parenthesis may be omitted entirely if the first item after the word "if" is a [function call](#) or an operator such as "not" or "!".

Empty strings: To specify an empty string in an expression, use an empty pair of quotes. For example, the statement `if (MyVar <> "")` would be true if *MyVar* is not blank. However, in a [traditional-if](#), a pair of empty quotes is treated literally. For example, the statement `if MyVar = ""` is true only if *MyVar* contains an actual pair of quotes. Thus, to check if a variable is blank with a traditional-if, use `=` or `<>` with nothing on the right side as in this example: `if`

Var = .

On a related note, any invalid expression such as `(x +* 3)` yields an empty string.

Storing the result of an expression: To assign a result to a variable, use the `:=` operator. For example:

```
NetPrice := Price * (1 - Discount/100)
```

Boolean values: When an expression is required to evaluate to true or false (such as an IF-statement), a blank or zero result is considered false and all other results are considered true. For example, the statement `if ItemCount` would be false only if ItemCount is blank or 0. Similarly, the expression `if not ItemCount` would yield the opposite result.

Operators such as NOT/AND/OR/>/=/< automatically produce a true or false value: they yield 1 for true and 0 for false. For example, in the following expression, the variable *Done* is assigned 1 if either of the conditions is true:

```
Done := A_Index > 5 or FoundIt
```

As hinted above, a variable can be used to hold a false value simply by making it blank or assigning 0 to it. To take advantage of this, the shorthand statement `if Done` can be used to check whether the variable Done is true or false.

The words *true* and *false* are built-in variables containing 1 and 0. They can be used to make a script more readable as in these examples:

```
CaseSensitive := false  
ContinueSearch := true
```

Integers and floating point: Within an expression, numbers are considered to be floating point if they contain a decimal point; otherwise, they are integers. For most operators -- such as addition and multiplication -- if either of the inputs is a floating point number, the result will also be a floating point number.

Within expressions and non-expressions alike, integers may be written in either hexadecimal or decimal format. Hexadecimal numbers all start with the prefix 0x. For example, `Sleep 0xFF` is equivalent to `Sleep 255`. [v1.0.46.11+]: Floating point numbers written in scientific notation are recognized; but only if they contain a decimal point (e.g. `1.0e4` and `-2.1E-4`).

Force an expression: An expression can be used in a parameter that does not directly support it (except OutputVar parameters) by preceding the expression with a percent sign and a space or tab. In [v1.1.21+], this prefix can be used in the InputVar parameters of all commands except the traditional IF commands (use `If (expression)` instead). This technique is often used to access [arrays](#). For example:

```
FileAppend, % MyArray[i], My File.txt
FileAppend, % MyPseudoArray%i%, My File.txt
MsgBox % "The variable MyVar contains " . MyVar
. ". "
Loop % Iterations + 1
WinSet, Transparent, % X + 100
Control, Choose, % CurrentSelection - 1
```

Operators in Expressions

See [Operators](#) for general information about operators.

Except where noted below, any blank value (empty string) or non-numeric value involved in a math operation is **not** assumed to be zero. Instead, it is treated as an error, which causes that part of the expression to evaluate to an empty string. For example, if the variable X is blank, the expression X+1 yields a blank value rather than 1.

For historical reasons, *quoted* numeric strings such as "123" are always considered non-numeric when used directly in an expression (but not when stored in a variable or returned by a function). This non-numeric attribute is propagated by [concatenation](#), so expressions like "0x" n also produce a non-numeric value (even when n contains valid hexadecimal digits). This problem can be avoided by assigning the value to a variable or passing it through a function like [Round\(\)](#). Scripts should avoid using quote marks around literal numbers, as the behavior may change in a future version.

Expression Operators (in descending precedence order)

If a variable is enclosed in percent signs within an expression (e.g. %Var%), whatever that variable contains is assumed to be the name or partial name of another variable (if there is no such variable, %Var% resolves to a blank string). This is most commonly used to reference [pseudo-array](#) elements such as the following example:

```
Var := MyArray%A_Index% + 100
```

%Var%

For backward compatibility, command parameters that are documented as "can be an expression" treat an isolated name in percent signs (e.g. %Var%, but not Array%i%) as though the percent signs are absent. This can be avoided by enclosing the

	<p>reference in parentheses; e.g. <code>Sleep (%Var%)</code> .</p> <p>[AHK_L 52+]: In addition to normal variables, %Var% may resolve to an environment variable, the clipboard, or any reserved/read-only variable. Prior to revision 52, %Var% yielded an empty string in these cases.</p>
x.y	<p>[AHK_L 31+]: Object access. Get or set a value or call a method of object <i>x</i>, where <i>y</i> is a literal value. See object syntax.</p>
new	<p>[v1.1.00+]: Creates a new object derived from another object. For example, <code>x := new y</code> is often equivalent to <code>x := {base: y}</code> . <code>new</code> should be followed by a variable or simple class name of the form <code>GlobalClass.NestedClass</code> , and optionally parameters as in <code>x := new y(z)</code> (where <i>y</i> is a variable, not a function name). For details, see Custom Objects.</p>
++ --	<p>Pre- and post-increment/decrement. Adds or subtracts 1 from a variable (but in versions prior to 1.0.46, these can be used only by themselves on a line; no other operators may be present). The operator may appear either before or after the variable's name. If it appears <i>before</i> the name, the operation is performed immediately and its result is used by the next operation. For example, <code>Var := ++X</code> increments X immediately and then assigns its value to <i>Var</i>. Conversely, if the operator appears <i>after</i> the variable's name, the operation is performed <i>after</i> the variable is used by the next operation. For example, <code>Var := X++</code> increments X only after assigning the current value of X to <i>Var</i>. Due to backward compatibility, the operators ++ and -- treat blank variables as zero, but only when they are alone on a line; for example, <code>y:=1, ++x</code> and <code>MsgBox % ++x</code> both produce a blank result when x is blank.</p>
	<p>Power. Both the base and the exponent may contain a decimal point. If the exponent is negative, the result will be formatted as a floating point number even if the base and exponent are both integers. Since ** is of higher precedence than unary minus, <code>-2**2</code> is evaluated like <code>-(2**2)</code> and so yields -4. Thus, to</p>

**

raise a literal negative number to a power, enclose it in parentheses such as `(-2)**2`.

Note: A negative base combined with a fractional exponent such as `(-2)**0.5` is not supported; it will yield an empty string. But both `(-2)**2` and `(-2)**2.0` are supported.

Unary minus (-): Although it uses the same symbol as the subtract operator, unary minus applies to only a single item or sub-expression as shown twice in this example: `-(3 / -x)`. On a related note, any unary plus signs (+) within an expression are ignored.

Logical-not (!): If the operand is blank or 0, the result of applying logical-not is 1, which means "true". Otherwise, the result is 0 (false). For example: `!x` or `!(y and z)`. Note: The word NOT is synonymous with ! except that ! has a higher precedence. [v1.0.46+]: Consecutive unary operators such as `!!Var` are allowed because they are evaluated in right-to-left order.

Bitwise-not (~): This inverts each bit of its operand. If the operand is a floating point value, it is truncated to an integer prior to the calculation. If the operand 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. For example, `~0xf0f` evaluates to 0xffff0f0 (4294963440).

-
!
~
& *

Address (&): `&MyVar` retrieves the address of `MyVar`'s contents in memory, which is typically used with [DllCall structures](#).

`&MyVar` also disables the caching of binary numbers in that variable, which can slow down its performance if it is ever used for math or numeric comparisons. Caching is re-enabled for a variable whenever its address changes (e.g. via [VarSetCapacity\(\)](#)).

Dereference (*): `*Expression` assumes that `Expression` resolves to a numeric memory address; it retrieves the byte at that

address as a number between 0 and 255 (0 is always retrieved if the address is 0; but any other invalid address must be avoided because it might crash the script). However, `NumGet()` generally performs much better when retrieving binary numbers.

Multiply (*): The result is an integer if both inputs are integers; otherwise, it is a floating point number.

True divide (/): Unlike `EnvDiv`, true division yields a floating point result even when both inputs are integers. For example, `3/2` yields 1.5 rather than 1, and `4/2` yields 2.0 rather than 2.

Floor divide (//): The double-slash operator uses high-performance integer division if the two inputs are integers. For example, `5//3` is 1 and `5//-3` is -1. If either of the inputs is in floating point format, floating point division is performed and the result is truncated to the nearest integer to the left. For example, `5//3.0` is 1.0 and `5.0//-3` is -2.0. Although the result of this floating point division is an integer, it is stored in floating point format so that anything else that uses it will see it as such. For modulo, see `mod()`.

The `*=` and `/=` operators are a shorthand way to multiply or divide the value in a variable by another value. For example, `Var*=2` produces the same result as `Var:=Var*2` (though the former performs better).

Division by zero yields a blank result (empty string).

Add (+) and subtract (-). On a related note, the `+=` and `-=` operators are a shorthand way to increment or decrement a variable. For example, `Var+=2` produces the same result as `Var:=Var+2` (though the former performs better). Similarly, a variable can be increased or decreased by 1 by using `Var++`, `Var--`, `++Var`, or `--Var`.

<p><< >></p>	<p>Bit shift left (<<) and right (>>). Example usage: <code>Value1 << Value2</code> . Any floating point input is truncated to an integer prior to the calculation. Shift left (<<) is equivalent to multiplying <i>Value1</i> by "2 to the <i>Value2</i>th power". Shift right (>>) is equivalent to dividing <i>Value1</i> by "2 to the <i>Value2</i>th power" and rounding the result to the nearest integer leftward on the number line; for example, <code>-3>>1</code> is -2.</p>
<p>& ^ </p>	<p>Bitwise-and (&), bitwise-exclusive-or (^), and bitwise-or (). Of the three, & has the highest precedence and has the lowest. Any floating point input is truncated to an integer prior to the calculation.</p>
<p>.</p>	<p>Concatenate. The period (dot) operator is used to combine two items into a single string (there must be at least one space on each side of the period). You may also omit the period to achieve the same result (except where ambiguous such as <code>x -y</code> , or when the item on the right side has a leading ++ or --). When the dot is omitted, there should be at least one space between the items to be merged.</p> <p>Example (expression method): <code>Var := "The color is " . FoundColor</code></p> <p>Example (traditional method): <code>Var = The color is %FoundColor%</code></p> <p>Sub-expressions can also be concatenated. For example: <code>Var := "The net price is " . Price * (1 - Discount/100)</code> .</p> <p>A line that begins with a period (or any other operator) is automatically appended to the line above it.</p>
<p>~=</p>	<p>[AHK_L 31+]: Shorthand for RegExMatch. For example, <code>"abc123" ~= "\d"</code> sets ErrorLevel to 0 and yields 4 (the position of the first numeric character). Prior to [v1.1.03], this operator had the same precedence as the <i>equal (=)</i> operator and was not fully documented.</p>

<p>> < >= <=</p>	<p>Greater (>), less (<), greater-or-equal (>=), and less-or-equal (<=). If either of the inputs is not a number, both are compared alphabetically (a quoted literal string such as "55" is always considered non-numeric in this context). The comparison is case sensitive only if StringCaseSense has been turned on. See also: Sort</p>
<p>= == <> !=</p>	<p>Equal (=), case-sensitive-equal (==), and not-equal (<> or !=). The operators != and <> are identical in function. The == operator behaves identically to = except when either of the inputs is not a number, in which case == is always case sensitive and = is always case insensitive (the method of insensitivity depends on StringCaseSense). By contrast, <> and != obey StringCaseSense.</p> <div style="background-color: #e0ffe0; padding: 10px; margin-top: 10px;"> <p>Note: A quoted literal string such as "55" is always considered non-numeric in this context.</p> </div>
<p>NOT</p>	<p>Logical-NOT. Except for its lower precedence, this is the same as the ! operator. For example, <code>not (x = 3 or y = 3)</code> is the same as <code>!(x = 3 or y = 3)</code>.</p>
<p>AND &&</p>	<p>Both of these are logical-AND. For example: <code>x > 3 and x < 10</code>. To enhance performance, short-circuit evaluation is applied. Also, a line that begins with AND/OR/&&/ (or any other operator) is automatically appended to the line above it.</p>
<p>OR </p>	<p>Both of these are logical-OR. For example: <code>x <= 3 or x >= 10</code>. To enhance performance, short-circuit evaluation is applied.</p>
<p>?:</p>	<p>Ternary operator [v1.0.46+]. This operator is a shorthand replacement for the if-else statement. It evaluates the condition on its left side to determine which of its two branches should become its final result. For example, <code>var := x>y ? 2 : 3</code> stores 2 in <i>Var</i> if x is greater than y; otherwise it stores 3. To enhance performance, only the winning branch is evaluated (see short-circuit evaluation).</p>
	<p>Assign. Performs an operation on the contents of a variable and</p>

stores the result back in the same variable (but in versions prior to 1.0.46, these could only be used as the leftmost operator on a line, and only the first five operators were supported). The simplest assignment operator is **colon-equals** (`:=`), which stores the result of an expression in a variable. For a description of what the other operators do, see their related entries in this table. For example, `Var //= 2` performs **floor division** to divide `Var` by 2, then stores the result back in `Var`. Similarly, `Var .= "abc"` is a shorthand way of writing `Var := Var . "abc"`.

Unlike most other operators, assignments are evaluated from right to left. Consequently, a line such as `Var1 := Var2 := 0` first assigns 0 to `Var2` then assigns `Var2` to `Var1`.

`:=`

`+=`

`-=`

`*=`

`/=`

`//=`

`.=`

`|=`

`&=`

`^=`

`>>=`

`<<=`

If an assignment is used as the input for some other operator, its value is the variable itself. For example, the expression `(Var+=2) > 50` is true if the newly-increased value in `Var` is greater than 50. This also allows an assignment to be passed **ByRef**, or its **address** taken; for example: `&(x:="abc")`.

The precedence of the assignment operators is automatically raised when it would avoid a syntax error or provide more intuitive behavior. For example: `not x:=y` is evaluated as `not (x:=y)`. Similarly, `++Var := X` is evaluated as `++(Var := X)`; and `Z>0 ? X:=2 : Y:=2` is evaluated as `Z>0 ? (X:=2) : (Y:=2)`.

Known limitations caused by backward compatibility (these may be resolved in a future release): 1) When `/=` is the leftmost operator in an expression and it is not part of a **multi-statement expression**, it performs **floor division** unless one of the inputs is floating point (in all other cases, `/=` performs **true division**); 2) **Date/time math** is supported by `+=` and `-=` only when that operator is the leftmost one on a line; 3) The operators `+=`, `-=`, and `*=` treat blank variables as zero, but only when they are alone on a line; for example, `y:=1`, `x+=1` and `MsgBox % x-=3` both produce a blank result when `x` is blank.

Comma (multi-statement) [v1.0.46+]. Commas may be used to write multiple sub-expressions on a single line. This is most commonly used to group together multiple assignments or function calls. For example: `x:=1, y+=2, ++index, MyFunc()`. Such statements are executed in order from left to right.

Note: A line that begins with a comma (or any other operator) is automatically [appended to the line above it](#). See also: [comma performance](#).

[v1.0.46.01+]: When a comma is followed immediately by a variable and an equal sign, that equal sign is automatically treated as an [assignment \(:=\)](#). For example, all of the following are assignments: `x:=1, y=2, a=b=c`. New scripts should not rely on this behavior as it may change. The rule applies only to plain variables and not [double-derefs](#), so the following contains only one assignment: `x:=1, %y%=2`

The following types of sub-expressions override precedence/order of evaluation:

<i>(expression)</i>	Any sub-expression enclosed in parentheses. For example, <code>(3 + 2) * 2</code> forces <code>3 + 2</code> to be evaluated first.
mod() round() abs()	Function call. The function name must be immediately followed by an open-parentheses, without any spaces or tabs in between. For details, see Functions .
<code>%func%()</code>	See Dynamically Calling a Function .
	Deprecated: This syntax is not recommended for use. Use <code>%func%()</code> (for function names and objects) or

<p><i>func</i>.() </p>	<p><code>func.Call()</code> (for function objects) instead.</p> <p>[AHK_L 48+]: Attempts to call an empty-named method of the object <i>func</i>. By convention, this is the object's "default" method. If <i>func</i> does not contain an object, the default base object is invoked instead.</p> <p>[v1.0.95+]: If <i>func</i> contains a function name, the named function is called.</p>
<p>Fn(<i>Params</i>*)</p>	<p>[AHK_L 60+]: Variadic function call. <i>Params</i> is an array (object) containing parameter values.</p>
<p><i>x</i>[<i>y</i>] [<i>a</i>, <i>b</i>, <i>c</i>]</p>	<p>[AHK_L 31+]: Member access. Get or set a value or call a method of object <i>x</i>, where <i>y</i> is a parameter list (typically an array index or key) or an expression which returns a method name.</p> <p>[v1.0.97+]: Array literal. If the open-bracket is not preceded by a value (or a sub-expression which yields a value), it is interpreted as the beginning of an array literal. For example, <code>[a, b, c]</code> is equivalent to <code>Array(a, b, c)</code> (<i>a</i>, <i>b</i> and <i>c</i> are variables).</p> <p>See array syntax and object syntax for more details.</p>
<p>{<i>a</i>: <i>b</i>, <i>c</i>: <i>d</i>}</p>	<p>[v1.0.97+]: Object literal. Create an object or associative array. For example, <code>x := {a: b}</code> is equivalent to <code>x := Object("a", b)</code> or <code>x := Object(), x.a := b</code>. See Associative Arrays for details.</p>

Performance: [v1.0.48+]: The comma operator is usually faster than writing

separate expressions, especially when assigning one variable to another (e.g. `x:=y, a:=b`). Performance continues to improve as more and more expressions are combined into a single expression; for example, it may be 35% faster to combine five or ten simple expressions into a single expression.

Built-in Variables

The variables below are built into the program and can be referenced by any script.

See [Built-in Variables](#) for general information.

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_IsSuspended](#), [A_BatchLines](#), [A_ListLines](#), [A_TitleMatchMode](#), (...more...)
- User Idle Time: [A_TimeIdle](#), [A_TimeIdlePhysical](#), [A_TimeIdleKeyboard](#), [A_TimeIdleMouse](#)
- GUI Windows and Menu Bars: [A_Gui](#), [A_GuiControl](#), [A_GuiEvent](#), [A_EventInfo](#)
- 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](#), [ClipboardAll](#), [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%), but cannot be referenced directly in an expression . The variable %0% contains the number of parameters passed (0 if none). For details, see the command line parameters .
A_Args [v1.1.27+]	Read/write: Contains an array of command line parameters. For details, see Passing Command Line Parameters to a Script .
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: C:\ and C:\My Documents. 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 omitted (even for root directories).
A_ScriptName	The file name of the current script, without its path, e.g. MyScript.ahk.
A_ScriptFullPath	The combination of the above two variables to give the complete file specification of the script, e.g. C:\My Documents\My Script.ahk
A_ScriptHwnd [v1.1.01+]	The unique ID (HWND/handle) of the script's hidden main window .
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: <code>MsgBox Could not write to log file (line number %A_LineNumber%)</code> .

	<p>Since a compiled script has merged all its #Include files into one big script, its line numbering may be different than when it is run in non-compiled mode.</p>
A_LineFile	<p>The full path and name of the file to which A_LineNumber belongs, which will be the same as A_ScriptFullPath unless the line belongs to one of a non-compiled script's #Include files.</p>
A_ThisFunc [v1.0.46.16+]	<p>The name of the user-defined function that is currently executing (blank if none); for example: MyFunction. See also: IsFunc()</p>
A_ThisLabel [v1.0.46.16+]	<p>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 (label), and OnExit. 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()</p>
A_AhkVersion	<p>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. The formatting of the version number allows a script to check whether A_AhkVersion is greater than some minimum version number with > or >= as in this example: <code>if A_AhkVersion >= 1.0.25.07</code>.</p>
A_AhkPath	<p>For non-compiled scripts: The full path and name of the EXE file that is actually running the current script. For example: C:\Program Files\AutoHotkey\AutoHotkey.exe</p> <p>For compiled scripts: The same as the above except the</p>

	AutoHotkey directory is discovered via the registry entry <i>HKLM\SOFTWARE\AutoHotkey\InstallDir</i> . If there is no such entry, A_AhkPath is blank.
A_IsUnicode	Contains 1 if strings are Unicode (16-bit) and an empty string (which is considered false) if strings are ANSI (8-bit). The format of strings depends on the version of AutoHotkey.exe which is used to run the script, or if it is compiled, which bin file was used to compile it.
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_MMMM	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 current user's language, e.g. Sunday
A_DDD	Current day of the week's abbreviation in 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: <code>FormatTime, OutputVar, , YDay0</code> .
A_YWeek	Current year and week number (e.g. 200453) according to ISO 8601. To separate the year from the week, use <code>StringLeft, Year, A_YWeek, 4</code> and <code>StringRight, Week, A_YWeek, 2</code> . Precise definition of A_YWeek: 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.
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: <code>FormatTime, OutputVar, , h:mm:ss tt</code>
A_Min	Current 2-digit minute (00-59).
A_Sec	Current 2-digit second (00-59).
A_MSec	Current 3-digit millisecond (000-999). To remove the leading zeros, follow this example: <code>Milliseconds := A_MSec + 0</code> .
	The current local time in <code>YYYYMMDDHH24MISS</code> format.

A_Now	<p>Note: Date and time math can be performed with EnvAdd and EnvSub. Also, FormatTime can format the date and/or time according to your locale or preferences.</p>
A_NowUTC	<p>The current Coordinated Universal Time (UTC) in YYYYMMDDHH24MISS format. UTC is essentially the same as Greenwich Mean Time (GMT).</p>
A_TickCount	<p>The number of milliseconds since the computer was rebooted. By storing <code>A_TickCount</code> in a variable, elapsed time can later be measured by subtracting that variable from the latest <code>A_TickCount</code> value. For example:</p> <pre data-bbox="521 877 1352 1205"> StartTime := A_TickCount Sleep, 1000 ElapsedTime := A_TickCount - StartTime MsgBox, %ElapsedTime% milliseconds have elapsed. </pre> <p>If you need more precision than <code>A_TickCount</code>'s 10ms, use QueryPerformanceCounter().</p>

Script Settings

A_IsSuspended	<p>Contains 1 if the script is suspended and 0 otherwise.</p>
A_IsPaused [v1.0.48+]	<p>Contains 1 if the thread immediately underneath the current thread is paused. Otherwise it contains 0.</p>
	<p>Contains 0 if Critical is off for the current thread. Otherwise it contains an integer greater</p>

<p>A_IsCritical [v1.0.48+]</p>	<p>than zero, namely the message-check frequency being used by Critical. Since <code>Critical 0</code> turns off critical, the current state of Critical can be saved and restored via <code>Old_IsCritical := A_IsCritical</code> followed later by <code>Critical %Old_IsCritical%</code>.</p>
<p>A_BatchLines</p>	<p>(synonymous with A_NumBatchLines) The current value as set by SetBatchLines. Examples: 200 or 10ms (depending on format).</p>
<p>A_ListLines [v1.1.28+]</p>	<p>Contains 1 if ListLines is enabled. Otherwise it contains 0.</p>
<p>A_TitleMatchMode</p>	<p>The current mode set by SetTitleMatchMode: 1, 2, 3, or RegEx.</p>
<p>A_TitleMatchModeSpeed</p>	<p>The current match speed (fast or slow) set by SetTitleMatchMode.</p>
<p>A_DetectHiddenWindows</p>	<p>The current mode (On or Off) set by DetectHiddenWindows.</p>
<p>A_DetectHiddenText</p>	<p>The current mode (On or Off) set by DetectHiddenText.</p>
<p>A_AutoTrim</p>	<p>The current mode (On or Off) set by AutoTrim.</p>
<p>A_StringCaseSense</p>	<p>The current mode (On, Off, or Locale) set by StringCaseSense.</p>
<p>A_FileEncoding</p>	<p>[AHK_L 46+]: Contains the default encoding for various commands; see FileEncoding.</p>
<p>A_FormatInteger</p>	<p>The current integer format (H or D) set by SetFormat. [AHK_L 42+]: This may also contain lower-case h.</p>
<p>A_FormatFloat</p>	<p>The current floating point number format set by SetFormat.</p>
<p>A_SendMode</p>	<p>[v1.1.23+]: The current mode (Event, Input, Play or InputThenPlay) set by SendMode.</p>

A_SendLevel	[v1.1.23+]: The current SendLevel setting (an integer between 0 and 100, inclusive).
A_StoreCapsLockMode	[v1.1.23+]: The current mode (On or Off) set by SetStoreCapsLockMode .
A_KeyDelay A_KeyDuration	The current delay or duration set by SetKeyDelay (always decimal, not hex). A_KeyDuration requires [v1.1.23+].
A_KeyDelayPlay A_KeyDurationPlay	The current delay or duration set by SetKeyDelay for the SendPlay mode (always decimal, not hex). Requires [v1.1.23+].
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 A_MouseDelayPlay	The current delay set by SetMouseDelay (always decimal, not hex). A_MouseDelay is for the traditional SendEvent mode, whereas A_MouseDelayPlay is for SendPlay . A_MouseDelayPlay requires [v1.1.23+].
A_DefaultMouseSpeed	The current speed set by SetDefaultMouseSpeed (always decimal, not hex).
A_CoordModeToolTip A_CoordModePixel A_CoordModeMouse A_CoordModeCaret A_CoordModeMenu	[v1.1.23+]: The current mode (Window, Client or Screen) set by CoordMode .
A_RegView	[v1.1.08+]: The current registry view as set by SetRegView .
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.
	Blank unless a custom tooltip for the tray icon

A_IconTip	has been specified via <code>Menu, Tray, Tip --</code> in which case it's the text of the tip.
A_IconFile	Blank unless a custom <code>tray icon</code> has been specified via <code>Menu, tray, icon --</code> 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	<p>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. Physical input from the user as well as artificial input generated by any program or script (such as the <code>Send</code> or <code>MouseMove</code> 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:</p> <pre>IfGreater, A_TimeIdle, 600000, MsgBox, The last keyboard or mouse activity was at least 10 minutes ago.</pre>
A_TimeIdlePhysical	<p>Similar to above but ignores artificial keystrokes and/or mouse clicks whenever the corresponding hook (<code>keyboard</code> or <code>mouse</code>) is installed; that is, it responds only to physical events. (This prevents simulated keystrokes and mouse clicks from falsely indicating that a user is present.) If neither hook is installed, this variable is equivalent to A_TimeIdle. If only one hook is installed, only its type of physical input affects A_TimeIdlePhysical (the other/non-installed hook's input, both physical and artificial, has no effect).</p>

A_TimeIdleKeyboard [v1.1.28+]	If the keyboard hook is installed, this is the number of milliseconds that have elapsed since the system last received physical keyboard input. Otherwise, this variable is equivalent to A_TimeIdle.
A_TimeIdleMouse [v1.1.28+]	If the mouse hook is installed, this is the number of milliseconds that have elapsed since the system last received physical mouse input. Otherwise, this variable is equivalent to A_TimeIdle.

GUI Windows and Menu Bars

A_DefaultGui [v1.1.23+]	The name or number of the current thread's default GUI .
A_DefaultListView [v1.1.23+]	The variable name or HWND of the ListView control upon which the ListView functions operate. If the default GUI lacks a ListView, this variable is blank.
A_DefaultTreeView [v1.1.23+]	The variable name or HWND of the TreeView control upon which the TreeView functions operate. If the default GUI lacks a TreeView, this variable is blank.
A_Gui	The name or number of the GUI 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).

<p>A_GuiWidth A_GuiHeight</p>	<p>These contain the GUI window's width and height when referenced in a GuiSize subroutine. They apply to the window's client area, which is the area excluding title bar, menu bar, and borders. [v1.1.11+]: These values are affected by DPI scaling.</p>
<p>A_GuiX A_GuiY</p>	<p>These contain the X and Y coordinates for GuiContextMenu and GuiDropFiles events. Coordinates are relative to the upper-left corner of the window. [v1.1.11+]: These values are affected by DPI scaling.</p>
<p>A_GuiEvent or A_GuiControlEvent</p>	<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 (      ). 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: Occurs only for GuiContextMenu, ListViews, and TreeViews.</p> <p>Context-sensitive values: For details see GuiContextMenu, GuiDropFiles, Slider, MonthCal, ListView, and TreeView.</p>
	<p>Contains additional information about the following</p>

A_EventInfo	<p>events:</p> <ul style="list-style-type: none"> • The OnClipboardChange label • Mouse wheel hotkeys (WheelDown/Up/Left/Right) • OnMessage() • RegisterCallback() • Regular Expression Callouts • GUI events, namely GuiContextMenu, GuiDropFiles, ListBox, ListView, TreeView, and StatusBar. If there is no additional information for an event, A_EventInfo contains 0.
-------------	---

Note: Unlike variables such as A_ThisHotkey, each [thread](#) retains its own value for A_Gui, A_GuiControl, A_GuiX/Y, A_GuiEvent, and A_EventInfo. Therefore, if a thread is interrupted by another, upon being resumed it will still see its original/correct values in these variables.

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 <u>current</u> 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.
	The most recently executed hotkey or non-auto-replace hotstring (blank if none), e.g. #z. This value will change if the current thread is

A_ThisHotkey	<p>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.</p> <p>When a hotkey is first created -- either by the Hotkey command or a double-colon label in the script -- its key name and the ordering of its modifier symbols becomes the permanent name of that hotkey, shared by all variants of the hotkey.</p> <p>See also: A_ThisLabel</p>
A_PriorHotkey	Same as above except for the previous hotkey. It will be blank if none.
A_PriorKey	<p>[v1.1.01+]: The name of the last key which was pressed prior to the most recent key-press or key-release, or blank if no applicable key-press can be found in the key history. All input generated by AutoHotkey scripts is excluded. For this variable to be of use, the keyboard or mouse hook must be installed and key history must be enabled.</p>
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.

Operating System and User Info

<p>ComSpec [v1.0.43.08+] A_ComSpec [v1.1.28+]</p>	<p>Contains the same string as the environment's ComSpec variable (e.g. C:\Windows\system32\cmd.exe). Often used with Run/RunWait.</p>
<p>A_Temp [v1.0.43.09+]</p>	<p>The full path and name of the folder designated to hold temporary files (e.g. C:\DOCUME~1\ <UserName>\LOCALS~1\Temp). It is retrieved from one of the following locations (in order): 1) the environment variables TMP, TEMP, or USERPROFILE; 2) the Windows directory.</p>
<p>A_OSType</p>	<p>The type of operating system being run. Since AutoHotkey 1.1 only supports NT-based operating systems, this is always WIN32_NT. Older versions of AutoHotkey return WIN32_WINDOWS when run on Windows 95/98/ME.</p>
<p>A_OSVersion</p>	<p>One of the following strings, if appropriate: WIN_7 [requires AHK_L 42+], WIN_8 [requires v1.1.08+], WIN_8.1 [requires v1.1.15+], WIN_VISTA, WIN_2003, WIN_XP, WIN_2000.</p> <p>Applying compatibility settings in the AutoHotkey executable or compiled script's properties causes the OS to report a different version number, which is reflected by A_OSVersion.</p> <p>[v1.1.20+]: If the OS version is not recognized as one of those listed above, a string in the format "major.minor.build" is returned. For example, <code>10.0.14393</code> is Windows 10 build 14393, also known as version 1607.</p> <div style="background-color: #e0e0e0; padding: 10px; margin-top: 10px;"> <p><code>; This example is obsolete as these operating systems are no longer supported.</code></p> </div>

	<pre> if A_OsVersion in WIN_NT4,WIN_95,WIN_98,WIN_ME ; Note: No spaces around commas. { MsgBox This script requires Windows 2000/XP or later. ExitApp } </pre>
A_Is64bitOS	[v1.1.08+]: Contains 1 (true) if the OS is 64-bit or 0 (false) if it is 32-bit.
A_PtrSize	[AHK_L 42+]: Contains the size of a pointer, in bytes. This is either 4 (32-bit) or 8 (64-bit), depending on what type of executable (EXE) is running the script.
A_Language	The system's default language, which is one of these 4-digit codes .
A_ComputerName	The name of the computer as seen on the network.
A_UserName	The logon name of the user who launched this script.
A_WinDir	The Windows directory. For example: C:\Windows
A_ProgramFiles	<p>The Program Files directory (e.g. C:\Program Files or C:\Program Files (x86)). This is usually the same as the ProgramFiles environment variable.</p> <p>On 64-bit systems (and not 32-bit systems), the following applies:</p> <ul style="list-style-type: none"> • If the executable (EXE) that is running the script is 32-bit, A_ProgramFiles returns the

or ProgramFiles	<p>path of the "Program Files (x86)" directory.</p> <ul style="list-style-type: none"> • For 32-bit processes, the <code>ProgramW6432</code> environment variable contains the path of the 64-bit Program Files directory. On Windows 7 and later, it is also set for 64-bit processes. • The <code>ProgramFiles(x86)</code> environment variable contains the path of the 32-bit Program Files directory. <p>[v1.0.43.08+]: The <code>A_</code> prefix may be omitted, which helps ease the transition to <code>#NoEnv</code>.</p>
<code>A_AppData</code> [v1.0.43.09+]	The full path and name of the folder containing the current user's application-specific data. For example: <code>C:\Documents and Settings\Username\Application Data</code>
<code>A_AppDataCommon</code> [v1.0.43.09+]	The full path and name of the folder containing the all-users application-specific data.
<code>A_Desktop</code>	The full path and name of the folder containing the current user's desktop files.
<code>A_DesktopCommon</code>	The full path and name of the folder containing the all-users desktop files.
<code>A_StartMenu</code>	The full path and name of the current user's Start Menu folder.
<code>A_StartMenuCommon</code>	The full path and name of the all-users Start Menu folder.
<code>A_Programs</code>	The full path and name of the Programs folder in the current user's Start Menu.
<code>A_ProgramsCommon</code>	The full path and name of the Programs folder in the all-users Start Menu.
<code>A_Startup</code>	The full path and name of the Startup folder in the current user's Start Menu.
<code>A_StartupCommon</code>	The full path and name of the Startup folder in the

	all-users Start Menu.
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_IsAdmin	<p>If the current user has admin rights, this variable contains 1. Otherwise, it contains 0.</p> <p>To have the script restart itself as admin (or show a prompt to the user requesting admin), use Run *RunAs. However, note that running the script as admin causes all programs launched by the script to also run as admin. For a possible alternative, see the FAQ.</p>
A_ScreenWidth A_ScreenHeight	<p>The width and height of the primary monitor, in pixels (e.g. 1024 and 768).</p> <p>To discover the dimensions of other monitors in a multi-monitor system, use SysGet.</p> <p>To instead discover the width and height of the entire desktop (even if it spans multiple monitors), use the following example:</p> <pre>SysGet, VirtualWidth, 78 SysGet, VirtualHeight, 79</pre> <p>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.</p>

A_ScreenDPI [v1.1.11+]	Number of pixels per logical inch along the screen width. In a system with multiple display monitors, this value is the same for all monitors. On most systems this is 96; it depends on the system's text size (DPI) setting. See also Gui -DPIScale .
A_IPAddress1 through 4	The IP addresses of the first 4 network adapters in the computer.

Misc.

A_Cursor	<p>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.</p>
A_CaretX A_CaretY	<p>The current X and Y coordinates of the caret (text insertion point). The coordinates are relative to the active window unless CoordMode is used to make them relative to the entire screen. If there is no active window or the caret position cannot be determined, these variables are blank.</p> <p>The following script allows you to move the caret around to see its current position displayed in an auto-update tooltip. Note that some windows (e.g. certain versions of MS Word) report the same caret position regardless of its actual position.</p> <pre data-bbox="511 1654 1351 1900"> #Persistent SetTimer, WatchCaret, 100 return WatchCaret: Tooltip, X%A_CaretX% </pre>

	<pre>Y%A_CaretY%, A_CaretX, A_CaretY - 20 return</pre>
Clipboard	Read/write: The contents of the OS's clipboard, which can be read or written to. See the Clipboard section.
ClipboardAll	The entire contents of the clipboard (such as formatting and text). See ClipboardAll .
ErrorLevel	Read/write: See ErrorLevel .
A_LastError	The result from the OS's GetLastError() function or the last COM object invocation. For details, see DllCall() and Run/RunWait .
True False	Contain 1 and 0. They can be used to make a script more readable. See Boolean Values for details.

Loop

A_Index	This is the number of the current loop iteration (a 64-bit integer). For example, the first time the script executes the body of a loop, this variable will contain the number 1. For details see Loop or While-loop .
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-reading loop .
A_LoopField	See parsing loop .

Variable Capacity and Memory

- 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 := ""`.
- There is no limit to how many variables a script may create. The program is designed to support at least several million variables without a significant drop in performance.
- Commands, functions, and expressions that accept numeric inputs generally support 15 digits of precision for floating point values. For integers, 64-bit signed values are supported, which range from -9223372036854775808 (-0x8000000000000000) to 9223372036854775807 (0x7FFFFFFFFFFFFFFF). Any integer constants outside this range are not supported and might yield inconsistent results. By contrast, arithmetic operations on integers wrap around upon overflow (e.g. `0x7FFFFFFFFFFFFFFF + 1 = -0x8000000000000000`).

Functions

Table of Contents

- Introduction and Simple Examples
- Parameters
- Optional Parameters
- Returning Values to Caller
- Variadic Functions
- Local Variables
- Dynamically Calling a Function
- Short-circuit Boolean Evaluation
- Using Subroutines Within a Function
- Return, Exit, and General Remarks
- Using #Include to Share Functions Among Multiple Scripts
- Libraries of Functions: Standard Library and User Library
- Built-in Functions

Introduction and Simple Examples

A function is similar to a subroutine (`Gosub`) except that it can accept parameters (inputs) from its caller. In addition, a function may optionally return a value to its caller. Consider the following simple function that accepts two numbers and returns their sum:

```
Add(x, y)
{
    return x + y    ; "Return" expects an
expression.
}
```

The above is known as a function *definition* because it creates a function named "Add" (not case sensitive) and establishes that anyone who calls it must provide exactly two parameters (x and y). To call the function, assign its result to a variable with the `:= operator`. For example:

```
Var := Add(2, 3) ; The number 5 will be stored
in Var.
```

Also, a function may be called without storing its return value:

```
Add(2, 3)
```

But in this case, any value returned by the function is discarded; so unless the function produces some effect other than its return value, the call would serve no purpose.

Since a function call is an `expression`, any variable names in its parameter list should not be enclosed in percent signs. By contrast, literal strings should be enclosed in double quotes. For example:

```
if InStr(MyVar, "fox")
    MsgBox The variable MyVar contains the word
fox.
```

Finally, functions may be called in the parameters of any command (except OutputVar and InputVar parameters such as those of [StringLen](#)). However, parameters that do not support [expressions](#) must use the "%" prefix as in this example:

```
MsgBox % "The answer is: " . Add(3, 2)
```

The "%" prefix is also permitted in parameters that natively support expressions, but it is simply ignored.

Parameters

When a function is defined, its parameters are listed in parentheses next to its name (there must be no spaces between its name and the open-parenthesis). If a function does not accept any parameters, leave the parentheses empty; for example: `GetCurrentTimestamp()`.

ByRef Parameters: From the function's point of view, parameters are essentially the same as [local variables](#) unless they are defined as *ByRef* as in this example:

```
Swap(ByRef Left, ByRef Right)
{
    temp := Left
    Left := Right
    Right := temp
}
```

In the example above, the use of *ByRef* causes each parameter to become an alias for the variable passed in from the caller. In other words, the parameter and the caller's variable both refer to the same contents in memory. This allows the Swap function to alter the caller's variables by moving *Left's* contents into *Right* and vice versa.

By contrast, if *ByRef* were not used in the example above, *Left* and *Right* would be copies of the caller's variables and thus the Swap function would have no external effect.

Since [return](#) can send back only one value to a function's caller, *ByRef* can be used to send back extra results. This is achieved by having the caller pass in a variable (usually empty) in which the function stores a value.

When passing large strings to a function, *ByRef* enhances performance and conserves memory by avoiding the need to make a copy of the string. Similarly, using *ByRef* to send a long string back to the caller usually performs better than something like `Return HugeString`.

[AHK_L 60+]: If something other than a modifiable variable is passed to a `ByRef` parameter, the function behaves as though the keyword "ByRef" is absent. For example, `Swap(A_Index, i)` stores the value of `A_Index` in `i`, but the value assigned to `Left` is discarded once the `Swap` function returns.

[v1.1.01+]: The `IsByRef()` function can be used to determine whether the caller supplied a variable for a given `ByRef` parameter.

Known limitations:

- Fields of objects are not considered variables for the purposes of `ByRef`. For example, if `foo.bar` is passed to a `ByRef` parameter, it will behave as though `ByRef` was omitted.
- It is not possible to pass `Clipboard`, `built-in variables`, or `environment variables` to a function's `ByRef` parameter, even when `#NoEnv` is absent from the script.
- Although a function may call itself recursively, if it passes one of its own `local variables` or non-`ByRef` parameters to itself `ByRef`, the new layer's `ByRef` parameter will refer to its own local variable of that name rather than the previous layer's. However, this issue does not occur when a function passes to itself a `global variable`, `static variable`, or `ByRef` parameter.
- If a parameter in a function-call resolves to a variable (e.g. `Var` or `++Var` or `Var*=2`), other parameters to its left or right can alter that variable before it is passed to the function. For example, `MyFunc(Var, Var++)` would unexpectedly pass 1 and 0 when `Var` is initially 0, even when the function's first parameter is not `ByRef`. Since this behavior is counterintuitive, it might change in a future release.
- `ByRef` is not directly supported in functions called by COM clients, or when calling COM methods. Instead, the script receives or must pass a `wrapper object` containing the `VarType` and address of the value.

Optional Parameters

When defining a function, one or more of its parameters can be marked as optional. This is done by appending `:=` (in [v1.1.09] or later) or `=`, followed by the parameter's default value, which must be one of the following: `true`, `false`, a literal integer, a literal floating point number, or a quoted/literal string such as "fox" or "" (but strings in versions prior to [v1.0.46.13] support only "").

The use of `=` (without a colon) is permitted for backward-compatibility, but not recommended, and will not be permitted by AutoHotkey v2. Regardless of which operator is used, default values which are strings must always be enclosed in quote marks.

The following function has its Z parameter marked optional:

```
Add(X, Y, Z:=0) {  
    return X + Y + Z  
}
```

When the caller passes **three** parameters to the function above, Z's default value is ignored. But when the caller passes only **two** parameters, Z automatically receives the value 0.

It is not possible to have optional parameters isolated in the middle of the parameter list. In other words, all parameters that lie to the right of the first optional parameter must also be marked optional. [AHK_L 31+]: Optional parameters may be omitted from the middle of the parameter list when calling the function, as shown below. For dynamic function calls and method calls, this requires [v1.1.12+].

```
MyFunc(1, , 3)  
MyFunc(X, Y:=2, Z:=0) { ; Note that Z must  
    still be optional in this case.  
    MsgBox %X%, %Y%, %Z%  
}
```

[v1.0.46.13+]: [ByRef parameters](#) also support default values; for example: `MyFunc(ByRef p1 = "")`. Whenever the caller omits such a parameter, the function creates a local variable to contain the default value; in other words, the function behaves as though the keyword "ByRef" is absent.

Returning Values to Caller

As described in [introduction](#), a function may optionally [return](#) a value to its caller.

```
Test := returnTest()  
MsgBox % Test  
  
returnTest() {  
    return 123  
}
```

If you want to return extra results from a function, you may also use [ByRef](#):

```
returnByRef(A, B, C)  
MsgBox % A ", " B ", " C  
  
returnByRef(ByRef val1, ByRef val2, ByRef val3)  
{  
    val1 := "A"  
    val2 := 100  
    val3 := 1.1  
    return  
}
```

[v1.0.97+]: [Objects](#) and [Arrays](#) can be used to return multiple values or even named values:

```
Test1 := returnArray1()  
MsgBox % Test1[1] ", " Test1[2]  
  
Test2 := returnArray2()  
MsgBox % Test2[1] ", " Test2[2]
```

```
Test3 := returnObject()
MsgBox % Test3.id ", " Test3.val

returnArray1() {
    Test := [123, "ABC"]
    return Test
}

returnArray2() {
    x := 456
    y := "EFG"
    return [x, y]
}

returnObject() {
    Test := {id: 789, val: "HIJ"}
    return Test
}
```

Variadic Functions [AHK_L 60+]

When defining a function, write an asterisk after the final parameter to mark the function as variadic, allowing it to receive a variable number of parameters:

```
Join(sep, params*) {
    for index,param in params
        str .= param . sep
    return SubStr(str, 1, -StrLen(sep))
}
MsgBox % Join("`n", "one", "two", "three")
```

When a variadic function is called, surplus parameters can be accessed via an object which is stored in the function's final parameter. The first surplus parameter is at `params[1]`, the second at `params[2]` and so on. As with any standard object, `params.MaxIndex()` can be used to determine the highest numeric index (in this case the number of parameters). However, if there are no parameters, `MaxIndex` returns an empty string.

Notes:

- The "variadic" parameter can only appear at the end of the formal parameter list.
- [RegEx callouts](#) cannot be variadic; the "variadic" parameter is tolerated but left blank.
- [Callbacks](#) pass surplus parameters [by address](#) rather than via an array.

Variadic Function Calls

While variadic functions can *accept* a variable number of parameters, an array of parameters can be passed to *any* function by applying the same syntax to a function-call:

```
substrings := ["one", "two", "three"]
MsgBox % Join("`n", substrings*)
```

Notes:

- Numbering of parameters within the source array begins at 1.
- Optional parameters may be entirely omitted from the array.
- The array of parameters may contain named items when calling a user-defined function; in any other case, named items are not supported.
- The target function may also be variadic, in which case named items are copied even if they have no corresponding formal parameter.
- This syntax can also be used when calling methods or retrieving properties of objects; for example, `Object.Property[Params*]` . [v1.1.12+]:
It can also be used for setting properties.

Known limitations:

- Only the right-most parameter can be expanded this way. For example, `MyFunc(x, y*)` is supported but `MyFunc(x*, y)` is not.
- There must not be any non-whitespace characters between the asterisk (`*`) and the symbol which ends the parameter list.

Local and Global Variables

Local Variables

Local variables are specific to a single function and are visible only inside that function. Consequently, a local variable may have the same name as a global variable and both will have separate contents. Separate functions may also safely use the same variable names.

All local variables which are not `static` are automatically freed (made empty) when the function returns.

Built-in variables such as `Clipboard`, `ErrorLevel`, and `A_TimeIdle` are never local (they can be accessed from anywhere), and cannot be redeclared.

Functions are **assume-local** by default. Variables accessed or created inside an assume-local function are local by default, with the following exceptions:

- `Super-global` variables, including `classes`.
- A `dynamic variable reference` may resolve to an existing global variable if no local variable exists by that name.
- `Commands that create pseudo-arrays` may create all elements as global even if only the first element is declared.

The default may also be overridden as shown below (by declaring the variable or by changing the mode of the function).

Force-local mode [v1.1.27+]: If the function's first line is the word "local", all variable references (even dynamic ones) are assumed to be local unless they are declared as `global` *inside* the function. Unlike the default mode, force-local mode has the following behavior:

- Super-global variables (including `classes`) cannot be accessed without declaring them inside the function.
- Dynamic variable references follow the same rules as non-dynamic ones. Only global variables which are declared inside the function can be accessed.
- `StringSplit` and other commands which create pseudo-arrays follow the

same rules as non-dynamic variable references (avoiding a common source of confusion).

- The *LocalSameAsGlobal* warning is never raised for variables within a force-local function.

Global variables

To refer to an existing global variable inside a function (or create a new one), declare the variable as global prior to using it. For example:

```
LogToFile(TextToLog)
{
    global LogFileName ; This global variable
    was previously given a value somewhere outside
    this function.
    FileAppend, %TextToLog%\n, %LogFileName%
}
```

Assume-global mode: If a function needs to access or create a large number of global variables, it can be defined to assume that all its variables are global (except its parameters) by making its first line either the word "global" or the declaration of a local variable. For example:

```
SetDefaults()
{
    global ; This word may be omitted if the
    first line of this function will be something
    like "local MyVar".
    MyGlobal := 33 ; Assigns 33 to a global
    variable, first creating the variable if
    necessary.
    local x, y:=0, z ; Local variables must be
    declared in this mode, otherwise they would be
    assumed global.
}
```

This assume-global mode can also be used by a function to create a global `array`, such as a loop that assigns values to `Array%A_Index%`.

Super-global variables [v1.1.05+]: If a global declaration appears outside of any function, it takes effect for all functions by default (excluding `force-local` functions). This avoids the need to redeclare the variable in each function. However, if a function parameter or local variable with the same name is declared, it takes precedence over the global variable. Variables created by the `class` keyword are also super-global.

Static variables

Static variables are always implicitly local, but differ from locals because their values are remembered between calls. For example:

```
LogToFile(TextToLog)
{
    static LoggedLines = 0
    LoggedLines += 1 ; Maintain a tally locally
    (its value is remembered between calls).
    global LogFileName
    FileAppend, %LoggedLines%: %TextToLog%\n,
    %LogFileName%
}
```

Static Initializers: In versions prior to 1.0.46, all static variables started off blank; so the only way to detect that one was being used for the first time was to check whether it was blank. [v1.0.46+]: A static variable may be initialized to something other than `""` by following it with `:=` or `=` followed by one of the following: `true`, `false`, a literal integer, a literal floating point number, or a literal/quoted string such as `"fox"`. For example: `static X:=0, Y:="fox"`. Each static variable is initialized only once (before the script begins executing).

[AHK_L 58+]: `Static var := expression` is supported. All such expressions are evaluated immediately before the script's auto-execute section in the order they are encountered in the script.

Assume-static mode [v1.0.48+]: A function may be defined to assume that all its variables are static (except its parameters) by making its first line the word "static". For example:

```
GetFromStaticArray(WhichItemNumber)
{
    static
    static FirstCallToUs := true ; A static
    declaration's initializer still runs only once
    (upon startup).
    if FirstCallToUs ; Create a static array
    during the first call, but not on subsequent
    calls.
    {
        FirstCallToUs := false
        Loop 10
            StaticArray%A_Index% := "Value #" .
    A_Index
    }
    return StaticArray%WhichItemNumber%
}
```

In assume-static mode, any variable that should not be static must be declared as local or global (with the same exceptions as for [assume-local mode](#), unless [force-local mode](#) is also in effect).

[v1.1.27+]: [Force-local mode](#) can be combined with assume-static mode by specifying `local` and then `static`, as shown below. This allows the function to use force-local rules but create variables as static by default.

```
global MyVar := "This is global"
DemonstrateForceStatic()

DemonstrateForceStatic()
{
    local
```

```
static
MyVar := "This is static"
ListVars
MsgBox
}
```

More about locals and globals

Multiple variables may be declared on the same line by separating them with commas as in these examples:

```
global LogFileName, MaxRetries := 5
static TotalAttempts = 0, PrevResult
```

[v1.0.46+]: A local or global variable may be initialized on the same line as its declaration by following it with `:=` or `=` followed by any [expression](#) (the `=` operator behaves the same as `:=` in declarations). Unlike [static initializers](#), the initializers of locals and globals execute every time the function is called, but only if/when the flow of control actually reaches them. In other words, a line like `local x = 0` has the same effect as writing two separate lines: `local x` followed by `x = 0`.

Because the words *local*, *global*, and *static* are processed immediately when the script launches, a variable cannot be conditionally declared by means of an [IF statement](#). In other words, a declaration inside an IF's or ELSE's [block](#) takes effect unconditionally for all lines between the declaration and the function's closing brace. Also note that it is not currently possible to declare a dynamic variable such as `global Array%i%`.

For commands that create [pseudo-arrays](#) (such as [StringSplit](#)), each variable in the resulting pseudo-array is local if the [assume-global mode](#) is not in effect or if the pseudo-array's first element has been declared as a local variable (this is also true if one of the function's parameters is passed `--` even if that parameter is [ByRef](#) `--` because parameters are similar to local variables). Conversely, if the first element has been [declared global](#), a global array is created. However, the *common source of confusion* below applies even in these cases. The first element for [StringSplit](#) is `ArrayName0`. For other array-creating commands such as

[WinGet List](#), the first element is ArrayName (i.e. without the number).

[v1.1.27+]: When [force-local mode](#) is in effect, these commands follow rules consistent with normal variable references; that is, any pseudo-array element not declared as global will be local even if other elements are declared global.

Within a function (unless [force-local mode](#) is in effect), any dynamic variable reference such as `Array%i%` always resolves to a local variable unless no variable of that name exists, in which case a global is used if it exists. If neither exists and the usage requires the variable to be created, it is created as a local variable unless the [assume-global mode](#) is in effect. Consequently, a function can create a global `array` manually (by means such as `Array%i% := A_Index`) only if it has been defined as an [assume-global](#) function.

Common source of confusion: Any *non*-dynamic reference to a variable creates that variable the moment the script launches. For example, when used outside a function, `MsgBox %Array1%` creates Array1 as a global the moment the script launches. Conversely, when used inside a function `MsgBox %Array1%` creates Array1 as one of the function's locals the moment the script launches (unless [assume-global](#) is in effect), even if Array and Array0 are declared global.

Dynamically Calling a Function

[v1.0.47.06+]: A function (even a [built-in function](#)) may be called dynamically via percent signs. For example, `%Var%(x, "fox")` would call the function whose name is contained in *Var*. Similarly, `Func%A_Index%()` would call `Func1()` or `Func2()`, etc., depending on the current value of `A_Index`.

[v1.1.07.00+]: *Var* in `%Var%()` can contain a function name or a [function object](#). If the function does not exist, the [default base object's](#) `__Call` meta-function is invoked instead.

If the function cannot be called due to one of the reasons below, the evaluation of the expression containing the call stops silently and prematurely, which may lead to inconsistent results:

- Calling a nonexistent function, which can be avoided by using `If IsFunc(VarContainingFuncName)`. Except for [built-in functions](#), the called function's [definition](#) must exist explicitly in the script by means such as [#Include](#) or a non-dynamic call to a [library function](#).
- Passing too few parameters, which can be avoided by checking `IsFunc()`'s return value (which is the number of mandatory parameters plus one).
[v1.0.48+]: Note that passing too many parameters is tolerated; each extra parameter is fully evaluated (including any calls to functions) and then discarded.

Finally, a dynamic call to a function is slightly slower than a normal call because normal calls are resolved (looked up) before the script begins running.

Short-circuit Boolean Evaluation

When *AND*, *OR*, and the [ternary operator](#) are used within an [expression](#), they short-circuit to enhance performance (regardless of whether any function calls are present). Short-circuiting operates by refusing to evaluate parts of an expression that cannot possibly affect its final result. To illustrate the concept, consider this example:

```
if (ColorName <> "" AND not
    FindColor(ColorName))
    MsgBox %ColorName% could not be found.
```

In the example above, the `FindColor()` function never gets called if the `ColorName` variable is empty. This is because the left side of the *AND* would be *false*, and thus its right side would be incapable of making the final outcome *true*.

Because of this behavior, it's important to realize that any side-effects produced by a function (such as altering a global variable's contents) might never occur if that function is called on the right side of an *AND* or *OR*.

It should also be noted that short-circuit evaluation cascades into nested *ANDs* and *ORs*. For example, in the following expression, only the leftmost comparison occurs whenever `ColorName` is blank. This is because the left side would then be enough to determine the final answer with certainty:

```
if (ColorName = "" OR FindColor(ColorName,
    Region1) OR FindColor(ColorName, Region2))
    break ; Nothing to search for, or a match
was found.
```

As shown by the examples above, any expensive (time-consuming) functions should generally be called on the right side of an *AND* or *OR* to enhance performance. This technique can also be used to prevent a function from being called when one of its parameters would be passed a value it considers

inappropriate, such as an empty string.

[v1.0.46+]: The [ternary conditional operator \(?:\)](#) also short-circuits by not evaluating the losing branch.

Using Subroutines Within a Function

Although a function cannot contain [definitions](#) of other functions, it can contain subroutines. As with other subroutines, use [Gosub](#) to launch them and [Return](#) to return (in which case the Return would belong to the Gosub and not the function).

Known limitation: Currently, the name of each subroutine (label) must be unique among those of the entire script. The program will notify you upon launch if there are duplicate labels.

If a function uses [Gosub](#) to jump to a public subroutine (one that lies outside of the function's braces), all variables outside are global and the function's own [local variables](#) are not accessible until the subroutine returns. However, `A_ThisFunc` will still contain the name of the function.

Although [Goto](#) cannot be used to jump from inside a function to outside, it is possible for a function to [Gosub](#) an external/public subroutine and then do a [Goto](#) from there.

Although the use of [Goto](#) is generally discouraged, it can be used inside a function to jump to another position within the same function. This can help simplify complex functions that have many points of return, all of which need to do some clean-up prior to returning.

A function may contain externally-called subroutines such as [timers](#), [GUI g-labels](#), and [menu items](#). This is generally done to encapsulate them in a separate file for use with [#Include](#), which prevents them from interfering with the script's [auto-execute section](#). However, the following limitations apply:

- Such subroutines should use only [static](#) and [global](#) variables (not [locals](#)) if their function is ever called normally. This is because a subroutine [thread](#) that interrupts a function-call thread (or vice versa) would be able to change the values of local variables seen by the interrupted thread. Furthermore, any time a function returns to its caller, all of its local variables are made blank to free their memory.
- Such subroutines should use only [global variables](#) (not [static variables](#)) as [GUI control variables](#).

- When a function is entered by a subroutine **thread**, any references to **dynamic variables** made by that thread are treated as **globals** (including commands that create arrays).

Return, Exit, and General Remarks

If the flow of execution within a function reaches the function's closing brace prior to encountering a [Return](#), the function ends and returns a blank value (empty string) to its caller. A blank value is also returned whenever the function explicitly omits [Return](#)'s parameter.

When a function uses the [Exit](#) command to terminate the [current thread](#), its caller does not receive a return value at all. For example, the statement `Var := Add(2, 3)` would leave `Var` unchanged if `Add()` exits. The same thing happens if a function causes a runtime error such as [running](#) a nonexistent file (when [UseErrorLevel](#) is not in effect).

A function may alter the value of [ErrorLevel](#) for the purpose of returning an extra value that is easy to remember.

To call a function with one or more blank values (empty strings), use an empty pair of quotes as in this example: `FindColor(ColorName, "")`.

Since calling a function does not start a new [thread](#), any changes made by a function to settings such as [SendMode](#) and [SetTitleMatchMode](#) will go into effect for its caller too.

The caller of a function may pass a nonexistent variable or [array](#) element to it, which is useful when the function expects the corresponding parameter to be [ByRef](#). For example, calling `GetNextLine(BlankArray%i%)` would create the variable `BlankArray%i%` automatically as a [local](#) or global (depending on whether the caller is inside a function and whether it has the [assume-global mode](#) in effect).

When used inside a function, [ListVars](#) displays a function's [local variables](#) along with their contents. This can help debug a script.

Style and Naming Conventions

You might find that complex functions are more readable and maintainable if their special variables are given a distinct prefix. For example, naming each parameter in a function's parameter list with a leading "p" or "p_" makes their special nature easy to discern at a glance, especially when a function has several dozen [local variables](#) competing for your attention. Similarly, the prefix "r" or "r_" could be used for [ByRef parameters](#), and "s" or "s_" could be used for [static variables](#).

The [One True Brace \(OTB\) style](#) may optionally be used to define functions. For example:

```
Add(x, y) {  
    return x + y  
}
```

Using #Include to Share Functions Among Multiple Scripts

The `#Include` directive may be used (*even at the top of a script*) to load functions from an external file.

Explanation: When the script's flow of execution encounters a function definition, it jumps over it (using an instantaneous method) and resumes execution at the line after its closing brace. Consequently, execution can never fall into a function from above, nor does the presence of one or more functions at the very top of a script affect the [auto-execute section](#).

Libraries of Functions: Standard Library and User Library [v1.0.47+]

A script may call a function in an external file without having to use `#Include`. For this to work, a file of the same name as the function must exist in one of the following library directories:

```
%A_ScriptDir%\Lib\ ; Local library - requires  
[AHK_L 42+].  
%A_MyDocuments%\AutoHotkey\Lib\ ; User library.  
path-to-the-currently-running-  
AutoHotkey.exe\Lib\ ; Standard library.
```

For example, if a script calls a nonexistent function `MyFunc()`, the program searches for a file named "MyFunc.ahk" in the user library. If not found there, it searches for it in the standard library. If a match is still not found and the function's name contains an underscore (e.g. `MyPrefix_MyFunc`), the program searches both libraries for a file named `MyPrefix.ahk` and loads it if it exists. This allows `MyPrefix.ahk` to contain both the function `MyPrefix_MyFunc` and other related functions whose names start with `MyPrefix_`.

[AHK_L 42+]: The local library is supported and is searched before the user library and standard library.

Only a direct function call such as `MyFunc()` can cause a library to be auto-included. If the function is only called dynamically or indirectly, such as by a timer or GUI event, the library must be explicitly included in the script. For example: `#Include <MyFunc>`

Although a library file generally contains only a single function of the same name as its filename, it may also contain private functions and subroutines that are called only by it. However, such functions should have fairly distinct names because they will still be in the global namespace; that is, they will be callable from anywhere in the script.

If a library file uses `#Include`, the working directory for `#Include` is the library file's own directory. This can be used to create a redirect to a larger library file that contains that function and others related to it.

The [script compiler \(ahk2exe\)](#) also supports library functions. However, it requires that a copy of `AutoHotkey.exe` exist in the directory above the compiler directory (which is normally the case). If `AutoHotkey.exe` is absent, the compiler still works but library functions are not automatically included.

Functions included from a library perform just as well as other functions because they are pre-loaded before the script begins executing.

Built-in Functions

Any optional parameters at the end of a built-in function's parameter list may be completely omitted. For example, `WinExist("Untitled - Notepad")` is valid because its other three parameters would be considered blank.

A built-in function is overridden if the script defines its own function of the same name. For example, a script could have its own custom `WinExist()` function that is called instead of the standard one. However, the script would then have no way to call the original function.

External functions that reside in DLL files may be called with [DllCall\(\)](#).

To get more details about a particular built-in function below, simply click on its name.

Frequently-used Functions

Function	Description
FileExist	Checks for the existence of a file or folder and returns its attributes.
GetKeyState	Returns true (1) if the specified key is down and false (0) if it is up.
InStr	Searches for a given occurrence of a string, from the left or the right.
RegExMatch	Determines whether a string contains a pattern (regular expression).
RegExReplace	Replaces occurrences of a pattern (regular expression) inside a string.
StrLen	Retrieves the count of how many characters are in a string.
StrReplace	Replaces occurrences of the specified substring with a new string.

StrSplit	Separates a string into an array of substrings using the specified delimiters.
SubStr	Retrieves one or more characters from the specified position in a string.
WinActive	Checks if the specified window exists and is currently active (foremost), and returns its Unique ID (HWND).
WinExist	Checks if a matching window exists and returns the Unique ID (HWND) of the first matching window.

Miscellaneous Functions

Function	Description
Asc	Returns the numeric value of the first byte or UTF-16 code unit in the specified string.
Chr	Returns the string (usually a single character) corresponding to the character code indicated by the specified number.
DllCall	Calls a function inside a DLL, such as a standard Windows API function.
Exception	Creates an object which can be used to throw a custom exception.
FileOpen	Provides object-oriented file I/O.
Format	Formats a variable number of input values according to a format string.
Func	Retrieves a reference to the specified function.
GetKeyName/VK/SC	Retrieves the name/text, virtual key code or scan code of a key.
IsByRef	Returns a non-zero number if a ByRef parameter of a function was supplied with the specified variable.
	Returns a non-zero number if the specified function

IsFunc	exists in the script.
IsLabel	Returns a non-zero number if the specified label exists in the script.
IsObject	Returns a non-zero number if the specified value is an object.
ListView	Functions to add, insert, modify or delete ListView rows/columns, or to get data from them.
NumGet	Returns the binary number stored at the specified address+offset.
NumPut	Stores a number in binary format at the specified address+offset.
OnMessage	Monitors a message/event.
Ord	Returns the ordinal value (numeric character code) of the first character in the specified string.
StrGet	Copies a string from a memory address, optionally converting it between code pages.
StrPut	Copies a string to a memory address, optionally converting it between code pages.
RegisterCallback	Creates a machine-code address that when called, redirects the call to a function in the script.
TreeView	Functions to add, modify or delete TreeView items, or to get data from them.
Trim	Trims characters from the beginning and/or end of a string.
VarSetCapacity	Enlarges a variable's holding capacity or frees its memory.

Math

Function	Description
----------	-------------

Abs	Returns the absolute value of <i>Number</i> .
Ceil	Returns <i>Number</i> rounded up to the nearest integer (without any .00 suffix).
Exp	Returns <i>e</i> (which is approximately 2.71828182845905) raised to the <i>N</i> th power.
Floor	Returns <i>Number</i> rounded down to the nearest integer (without any .00 suffix).
Log	Returns the logarithm (base 10) of <i>Number</i> .
Ln	Returns the natural logarithm (base <i>e</i>) of <i>Number</i> .
Max / Min	Returns the highest/lowest value of one or more numbers.
Mod	Returns the remainder when <i>Dividend</i> is divided by <i>Divisor</i> .
Round	Returns <i>Number</i> rounded to <i>N</i> decimal places.
Sqrt	Returns the square root of <i>Number</i> .
Sin / Cos / Tan	Returns the trigonometric sine/cosine/tangent of <i>Number</i> .
ASin / ACos / ATan	Returns the arcsine/arccosine/arctangent in radians.

Other Functions

Polyethene's Command Functions: Provides a callable function for each AutoHotkey command that has an OutputVar. This library can be included in any script via [#Include](#).

Labels

A label identifies a line of code, and can be used as a [Goto](#) target or to form a [subroutine](#).

```
LabelName:
```

To create a label, write the label name followed by a colon as shown above. Aside from whitespace and comments, no other code can be written on the same line.

Names: Label names are not case sensitive, and may consist of any characters other than space, tab, comma and the [escape character](#) (```). However, due to style conventions, it is generally better to use only letters, numbers, and the underscore character (for example: *MyListView*, *Menu_File_Open*, and *outer_loop*). Label names must be unique throughout the whole script.

Although there are no reserved names, it is strongly recommended that the following names not be used: On, Off, Toggle, AltTab, ShiftAltTab, AltTabAndMenu and AltTabMenuDismiss. These values have special meaning to the [Hotkey command](#).

Target: The target of a label is the next line of executable code. Executable code includes commands, assignments, [expressions](#) and [blocks](#), but not directives, labels, hotkeys or hotstrings. In the following example, `run_notepad` and `#n` both point at the `Run` line:

```
run_notepad:
#n::
    Run Notepad
    return
```

Execution: Like directives, labels have no effect when reached during normal execution. In the following example, a message box is shown twice - once during execution of the subroutine by [Gosub](#), and again after the subroutine

returns:

```
gosub Label1  
  
Label1:  
MsgBox %A_ThisLabel%  
return
```

Subroutines

A subroutine is a portion of code which can be *called* to perform a specific task. Execution of a subroutine begins at the target of a label and continues until a [Return](#) or [Exit](#) is encountered. Since the end of a subroutine depends on flow of control, any label can act as both a Goto target and the beginning of a subroutine.

Dynamic Labels

Many commands which accept a label name also accept a [variable](#) reference such as %MyLabel%, in which case the name stored in the variable is used as the target. However, performance is slightly reduced because the target label must be "looked up" each time rather than only once when the script is first loaded.

Hotkeys and Hotstrings

Hotkey and **hotstring labels** are also valid targets for **Goto**, **Gosub** and other commands. However, if a hotkey or hotstring has multiple **variants**, the variant closest to the top of the script is used. All of the hotkey's modifiers or hotstring's options are also part of its label name, but the final double-colon (::) is omitted.

Named Loops

A label can also be used to identify a loop for the [Continue](#) and [Break](#) commands. This allows the script to easily continue or break out of any number of nested loops.

Functions

[v1.1.20+]: [Functions](#) can be used in place of labels in a number of cases, including:

- [Gui events](#) such as GuiClose
- [Gui control events](#) (g-labels)
- [Hotkey](#)
- [Menu](#)
- [SetTimer](#)

The benefits of functions are that they can use local variables, and in some cases (such as Gui control events) they also accept parameters containing useful information.

Related

[IsLabel\(\)](#), [A_ThisLabel](#), [Gosub](#), [Goto](#), [OnExit](#), [SetTimer](#), [Hotkey](#), [Gui Events](#), [g-label](#), [OnClipboardChange](#)

Threads

The *current thread* is defined as the flow of execution invoked by the most recent event; examples include [hotkeys](#), [SetTimer subroutines](#), [custom menu items](#), and [GUI events](#). The *current 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 some of 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 a 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 [SendMode](#) 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.

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](#), [custom menu item](#), etc.) 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

Any thread ([hotkey](#), [timed subroutine](#), [custom menu item](#), etc.) with a priority lower than that of the *current thread* cannot interrupt it. During that time, such timers will not run, and any attempt by the user to create a thread (such as by pressing a [hotkey](#) or [GUI button](#)) 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 use [Critical](#) instead of 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.

Debugger Clients

Additional debugging features are supported via [DBGp](#), a common debugger protocol for languages and debugger UI communication. See [Interactive Debugging](#) for more details. Some UIs or "clients" known to be compatible with AutoHotkey are listed on this page.

SciTE4AutoHotkey

SciTE4AutoHotkey is a free, SciTE-based AutoHotkey script editor. In addition to DBGp support, it provides syntax highlighting, calltips/parameter info and auto-complete for AutoHotkey, and other useful editing features and scripting tools.

Debugging features include:

- Breakpoints.
- Run, Step Over/Into/Out.
- View the call stack.
- List name and contents of variables in local/global scope.
- Hover over variable to show contents.
- Inspect or edit variable contents.
- View structure of objects.

<http://fincs.ahk4.net/scite4ahk/>

XDebugClient

[XDebugClient](#) is a simple open-source front-end DBGp client based on the **.NET Framework 2.0**. XDebugClient was originally designed for PHP with Xdebug, but a custom build compatible with AutoHotkey is available below.

Changes:

- Allow the debugger engine to report a language other than "php".
- Added AutoHotkey syntax highlighting.
- Automatically listen for a connection from the debugger engine, rather than waiting for the user to click *Start Listening*.
- Truncate property values at the first null-character, since AutoHotkey currently returns the entire variable contents and XDebugClient has no suitable interface for displaying binary content.

Download: [Binary](#); [Source Code](#) (also see [SharpDevelop](#), [Dockpanel Suite](#) and [Advanced TreeView](#).)

Usage:

- Launch XDebugClient.
- Launch AutoHotkey /Debug. XDebugClient should automatically open the script file.
- Click the left margin to set at least one breakpoint.
- Choose Run from the Debug menu, or press 
- When execution hits a breakpoint, use the Debug menu or shortcut keys to step through or resume the script.

Features:

- Syntax highlighted, read-only view of the source code.
- Breakpoints.
- Run, Step Over/Into/Out.
- View the call stack.
- Inspect variables - select a variable name, right-click, Inspect.

Issues:

- The user interface does not respond to user input while the script is running.
- No mechanisms are provided to list variables or set their values.

Notepad++ DBGp Plugin

A DBGp client is available as a plugin for [Notepad++](#). It is designed for PHP, but also works well with AutoHotkey.

Download: See [DBGp plugin for Notepad++](#).

Usage:

- Launch Notepad++.
- Configure the DBGp plugin via *Plugins, DBGp, Config...*

Note: File Mapping must be configured. Most users will not be debugging remotely, and therefore may simply put a checkmark next to *Bypass all mapping (local windows setup)*.

- Show the debugger pane via the toolbar or **Plugins, DBGp, Debugger**.
- Open the script file to be debugged.
- Set at least one breakpoint.
- Launch AutoHotkey /Debug.
- Use the debugger toolbar or shortcut keys to control the debugger.

Features:

- Syntax highlighting, if configured by the user.
- Breakpoints.
- Run, Step Over/Into/Out, Run to cursor, Stop.
- View local/global variables.
- Watch user-specified variables.
- View the call stack.
- Hover over a variable to view its contents.
- User-configurable shortcut keys - Settings, Shortcut Mapper..., Plugin commands.

Issues:

- Hovering over a single-letter variable name does not work - for instance,

hovering over `a` will attempt to retrieve `a` or `a` .

- Hovering over text will attempt to retrieve a variable even if the text contains invalid characters.
- Notepad++ becomes unstable if `property_get` fails, which is particularly problematic in light of the above. As a workaround, AutoHotkey sends an empty property instead of an error code when a non-existent or invalid variable is requested.

Script-based Clients

A script-based DBGp library and example clients are available from GitHub.

- `dbg_console.ahk`: Simple command-line client.
- `dbg_test.ahk`: Demonstrates asynchronous debugging.
- `dbg_listvars.ahk`: Example client which just lists the variables of all running scripts.

GitHub: [Lexikos / dbg](#)

The `DebugVars` script provides a graphical user interface for inspecting and changing the contents of variables and objects in any running script (except compiled scripts). It also serves as a demonstration of the `dbg.ahk` library.

GitHub: [Lexikos / DebugVars](#)

Command-line Client

A command-line client is available from xdebug.org, however this is not suitable for most users as it requires a decent understanding of DBGp (the protocol).

Others

A number of other DBGp clients are available, but have not been tested with AutoHotkey. For a list, see [Xdebug: Documentation](#).

Objects

An *object* in AutoHotkey is an abstract datatype which provides three basic functions:

- GET a value.
- SET a value.
- CALL a method (that is, a function which does something with the target object).

Related topics:

- [Objects](#): General explanation of objects.
- [Object Protocol](#): Specifics about how a script interacts with an object.

IsObject can be used to determine if a value is an object:

```
Result := IsObject(expression)
```

Types of objects include:

- **Object** - scriptable associative array.
- **File** - provides an interface for file input/output.
- **Function objects** - [Func](#), [BoundFunc](#) or [user-defined](#).
- **ComObject** - wraps an IDispatch interface (a COM or "Automation" object).

Table of Contents

- [Basic Usage](#) - [Simple Arrays](#), [Associative Arrays](#), [Objects](#), [Freeing Objects](#), [Remarks](#)
- [Extended Usage](#) - [Function References](#), [Arrays of Arrays](#), [Arrays of Functions](#)
- [Custom Objects](#) - [Prototypes](#), [Classes](#), [Construction and Destruction](#), [Meta-Functions](#)
- [Default Base Object](#) - [Automatic Var Init](#), [Pseudo-Properties](#), [Debugging](#)
- [Implementation](#) - [Reference-Counting](#), [Pointers to Objects](#)

Basic Usage

Simple Arrays

Create an array:

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

Retrieve an item:

```
Value := Array[Index]
```

Assign an item:

```
Array[Index] := Value
```

Insert one or more items at a given index:

```
Array.InsertAt(Index, Value, Value2, ...)
```

Append one or more items:

```
Array.Push(Value, Value2, ...)
```

Remove an item:

```
RemovedValue := Array.RemoveAt(Index)
```

Remove the last item:

```
RemovedValue := Array.Pop()
```

If the array is not empty, `MinIndex` and `MaxIndex/Length` return the lowest and highest index currently in use in the array. Since the lowest index is nearly always 1, `MaxIndex` usually returns the number of items. However, if there are no integer keys, `MaxIndex` returns an empty string whereas `Length` returns 0. Looping through an array's contents can be done either by index or with a `For`-loop. For example:

```
array := ["one", "two", "three"]

; Iterate from 1 to the end of the array:
Loop % array.Length()
    MsgBox % array[A_Index]

; Enumerate the array's contents:
For index, value in array
    MsgBox % "Item " index " is " value ""
```

Associative Arrays

An associative array is an object which contains a collection of unique keys and a collection of values, where each key is associated with one value. Keys can be strings, integers or objects, while values can be of any type. An associative array can be created as follows:

```
Array := {KeyA: ValueA, KeyB: ValueB, ..., KeyZ: ValueZ}
Array := Object("KeyA", ValueA, "KeyB", ValueB, ..., "KeyZ", ValueZ)
```

Using the `{key:value}` notation, quote marks are optional for keys which consist only of word characters. Any expression can be used as a key, but to use a variable as a key, it must be enclosed in parentheses. For example, `{(KeyVar): Value}` and `{GetKey(): Value}` are both valid.

Retrieve an item:

```
Value := Array[Key]
```

Assign an item:

```
Array[Key] := Value
```

Remove an item:

```
RemovedValue := Array.Delete(Key)
```

Enumerating items:

```
array := {ten: 10, twenty: 20, thirty: 30}  
For key, value in array  
    MsgBox %key% = %value%
```

Associative arrays can be sparsely populated - that is, `{1:"a", 1000:"b"}` contains only two key-value pairs, not 1000.

In AutoHotkey v1.x, simple arrays and associative arrays are the same thing. However, treating `[]` as a simple linear array helps to keep its role clear, and improves the chance of your script working with a future version of AutoHotkey, which might differentiate between simple arrays and associative arrays.

Objects

For all types of objects, the notation `Object.LiteralKey` can be used to access a property, array element or method, where *LiteralKey* is an identifier or integer and *Object* is any expression. Identifiers are unquoted strings which may consist of alphanumeric characters, underscore and, in [v1.1.09+], non-ASCII characters. For example, `match.Pos` is equivalent to `match["Pos"]` while `arr.1` is equivalent to `arr[1]`. There must be no space after the dot.

Examples:

Retrieve a property:

```
Value := Object.Property
```

Set a property:

```
Object.Property := Value
```

Call a method:

```
ReturnValue := Object.Method(Parameters)
```

Call a method with a computed method name:

```
ReturnValue := Object[MethodName](Parameters)
```

Some properties of COM objects and user-defined objects can accept parameters:

```
Value := Object.Property[Parameters]  
Object.Property[Parameters] := Value
```

Related: [Object](#), [File Object](#), [Func Object](#), [COM object](#)

Known limitation:

- Currently `x.y[z]()` is treated as `x["y", z]()`, which is not supported. As a workaround, `(x.y)[z]()` evaluates `x.y` first, then uses the result as the target of the method call. Note that `x.y[z].Call()` does not have this limitation since it is evaluated the same as `(x.y[z]).Call()`.

Freeing Objects

Scripts do not free objects explicitly. When the last reference to an object is released, the object is freed automatically. A reference stored in a variable is released automatically when that variable is assigned some other value. For example:

```
obj := {} ; Creates an object.  
obj := "" ; Releases the last reference, and  
therefore frees the object.
```

Similarly, a reference stored in a field of another object is released when that field is assigned some other value or removed from the object. This also applies to arrays, which are actually objects.

```
arr := [{}] ; Creates an array containing an  
object.  
arr[1] := {} ; Creates a second object,  
implicitly freeing the first object.  
arr.RemoveAt(1) ; Removes and frees the second  
object.
```

Because all references to an object must be released before the object can be freed, objects containing circular references aren't freed automatically. For instance, if `x.child` refers to `y` and `y.parent` refers to `x`, clearing `x` and `y` is not sufficient since the parent object still contains a reference to the child and vice versa. To resolve this situation, remove the circular reference.

```
x := {}, y := {} ; Create two  
objects.  
x.child := y, y.parent := x ; Create a circular  
reference.  
  
y.parent := "" ; The circular  
reference must be removed before the objects can
```

```
be freed.  
x := "", y := "" ; Without the above  
line, this would not free the objects.
```

For more advanced usage and details, see [Reference Counting](#).

Remarks

Syntax

All types of objects support both array syntax (brackets) and object syntax (dots).

Additionally, object references can themselves be used in expressions:

- When an object reference is compared with some other value using one of `=` `==` `!=` `<>`, they are considered equal only if both values are references to the same object.
- Objects are always considered *true* when a boolean value is required, such as in `if obj`, `!obj` or `obj ? x : y`.
- An object's address can be retrieved using the `&` address-of operator. This uniquely identifies the object from the point of its creation to the moment its last reference is [released](#).

If an object is used in any context where an object is not expected, it is treated as an empty string. For example, `MsgBox %object%` shows an empty `MsgBox` and `object + 1` yields an empty string. Do not rely on this behaviour as it may change.

When a method-call is followed immediately by an assignment operator, it is equivalent to setting a property with parameters. For example, the following are equivalent:

```
obj.item(x) := y  
obj.item[x] := y
```

Compound assignments such as `x.y += 1` and `--arr[1]` are supported.

[v1.1.20+]: Parameters can be omitted when getting or setting properties. For example, `x[, 2]`. Scripts can utilize this by defining default values for parameters in [properties](#) and [meta-functions](#). The method name can also be completely omitted, as in `x[](a)`. Scripts can utilize this by defining a default value for the `__Call` [meta-function](#)'s first parameter, since it is not otherwise supplied with a value. Note that this differs from `x.(a)`, which is equivalent to `x[""](a)`. If the property or method name is omitted when invoking a COM object, its "default member" is invoked.

Keys

Some limitations apply to which values can be used as keys in objects created with `[]`, `{}` or the `new` operator:

- Integer keys are stored using the native signed integer type. AutoHotkey 32-bit supports integer keys in the range -2147483648 to 2147483647. AutoHotkey supports 64-bit integers, but only AutoHotkey 64-bit supports the full range as keys in an object.
- As a consequence of the point above, the string format of integer values is not retained. For example, `x[0x10]`, `x[16]` and `x[00016]` are equivalent. This also applies to numeric strings which don't have a decimal point.
- Quoted literal strings are considered purely non-numeric in v1.x, so `x[1]` and `x["1"]` are *not* equivalent. Additionally, if a quoted literal string is concatenated with another value (as in `"0x" x`), the result is treated as purely non-numeric. However, this does not apply to variables, so `x[1]` and `x[y:="1"]` are equivalent. This issue will be resolved in [AutoHotkey v2](#), so scripts should avoid using quoted numeric literals as keys.
- Floating-point numbers are not supported as keys - instead they are converted to strings. In v1.x, floating-point literals retain their original format whereas pure floating-point numbers (such as the result of `0+1.0` or `Sqrt(y)`) are forced into the current [float format](#). For consistency and clarity, scripts should avoid using floating-point literals as keys.
- By default, the string key "base" is used to retrieve or set the object's [base object](#), so cannot be used for storing ordinary values with a normal assignment. However, if a value is stored by some other means (such as `ObjRawSet(Object, "base", "")` or

`Object.SetCapacity("base", 0)`), the key "base" then acts like any other string.

- Although **built-in method** names such as "Length" can be used as keys, storing a value will prevent the corresponding method from being called (unless that value is a reference to the appropriate function, such as *ObjLength*).

Extended Usage

Function References [v1.1.00+]

If the variable *func* contains a function name, the function can be called one of two ways: `%func%()` or `func.()`. However, this requires the function name to be resolved each time, which is inefficient if the function is called more than once. To improve performance, the script can retrieve a reference to the function and store it for later use:

```
Func := Func("MyFunc")
```

A function can be called by reference using the following syntax:

```
RetVal := %Func%(Params)      ; Requires  
[v1.1.07+]  
RetVal := Func.Call(Params)   ; Requires  
[v1.1.19+]  
RetVal := Func.(Params)       ; Not recommended
```

For details about additional properties of function references, see [Func Object](#).

Arrays of Arrays

AutoHotkey supports "multi-dimensional" arrays by transparently storing arrays inside other arrays. For example, a table could be represented as an array of rows, where each row is itself an array of columns. In that case, the content of column *y* of row *x* can be set using either of the methods below:

```
table[x][y] := content ; A  
table[x, y] := content ; B
```

If `table[x]` does not exist, *A* and *B* differ in two ways:

- *A* fails whereas *B* automatically creates an object and stores it in `table[x]`.
- If `table`'s `base` defines `meta-functions`, they are invoked as follows:

```

table.base.__Get(table, x)[y] := content ;
A
table.base.__Set(table, x, y, content) ;
B

```

Consequently, *B* allows the object to define custom behaviour for the overall assignment.

Multi-dimensional assignments such as `table[a, b, c, d] := value` are handled as follows:

- If there is only one key remaining, perform the assignment and return. Otherwise:
- Search the object for the first key in the list.
- If a non-object is found, fail.
- If an object is not found, create one and store it.
- Recursively invoke the sub-object, passing the remaining keys and value - repeat from the top.

This behaviour only applies to script-created objects, not more specialized types of objects such as COM objects or COM arrays.

Arrays of Functions

An array of functions is simply an array containing function names or references. For example:

```

array := [Func("FirstFunc"), Func("SecondFunc")]

; Call each function, passing "foo" as a
parameter:
Loop 2
    array[A_Index].Call("foo")

```

```
; Call each function, implicitly passing the
array itself as a parameter:
Loop 2
    array[A_Index]()

FirstFunc(param) {
    MsgBox % A_ThisFunc ": " (isObject(param) ?
"object" : param)
}
SecondFunc(param) {
    MsgBox % A_ThisFunc ": " (isObject(param) ?
"object" : param)
}
```

For backward-compatibility, the second form will not pass *array* as a parameter if `array[A_Index]` contains a function name instead of a function reference. However, if `array[A_Index]` is [inherited](#) from `array.base[A_Index]`, *array* will be passed as a parameter.

Custom Objects

Objects created by the script do not need to have any predefined structure. Instead, each object can inherit properties and methods from its `base` object (otherwise known as a "prototype" or "class"). Properties and methods can also be added to (or removed from) an object at any time, and those changes will affect any and all derived objects. For more complex or specialized situations, a base object can override the standard behaviour of any objects derived from it by defining *meta-functions*.

Base objects are just ordinary objects, and are typically created one of two ways:

```
class baseObject {
    static foo := "bar"
}
; OR
baseObject := {foo: "bar"}
```

To create an object derived from another object, scripts can assign to the `base` property or use the `new` keyword:

```
obj1 := Object(), obj1.base := baseObject
obj2 := {base: baseObject}
obj3 := new baseObject
MsgBox % obj1.foo " " obj2.foo " " obj3.foo
```

It is possible to reassign an object's `base` at any time, effectively replacing all of the properties and methods that the object inherits.

Prototypes

Prototype or `base` objects are constructed and manipulated the same as any other object. For example, an ordinary object with one property and one method might be constructed like this:

```

; Create an object.
thing := {}
; Store a value.
thing.foo := "bar"
; Create a method by storing a function
reference.
thing.test := Func("thing_test")
; Call the method.
thing.test()

thing_test(this) {
    MsgBox % this.foo
}

```

When `thing.test()` is called, *thing* is automatically inserted at the beginning of the parameter list. However, for backward-compatibility, this does not occur when a function is stored by name (rather than by reference) directly in the object (rather than being inherited from a base object). By convention, the function is named by combining the "type" of object and the method name.

An object is a *prototype* or *base* if another object derives from it:

```

other := {}
other.base := thing
other.test()

```

In this case, *other* inherits *foo* and *test* from *thing*. This inheritance is dynamic, so if `thing.foo` is modified, the change will be reflected by `other.foo`. If the script assigns to `other.foo`, the value is stored in *other* and any further changes to `thing.foo` will have no effect on `other.foo`. When `other.test()` is called, its *this* parameter contains a reference to *other* instead of *thing*.

Classes [v1.1.00+]

At its root, a "class" is a set or category of things having some property or

attribute in common. Since a [base](#) or [prototype](#) object defines properties and behaviour for set of objects, it can also be called a *class* object. For convenience, base objects can be defined using the "class" keyword as shown below:

```
class ClassName extends BaseClassName
{
    InstanceVar := Expression
    static ClassVar := Expression

    class NestedClass
    {
        ...
    }

    Method()
    {
        ...
    }

    Property[] ; Brackets are optional
    {
        get {
            return ...
        }
        set {
            return ... := value
        }
    }
}
```

When the script is loaded, this constructs an object and stores it in the global (or [in v1.1.05+] [super-global](#)) variable *ClassName*. To reference this class inside a [force-local](#) function (or an [assume-local](#) or [assume-static](#) function prior to [v1.1.05]), a declaration such as `global ClassName` is required. If `extends BaseClassName` is present, *BaseClassName* must be the full name of another class (but as of [v1.1.11], the order that they are defined in does

not matter). The full name of each class is stored in `object.__Class`.

Because the class is referenced via a variable, the class name cannot be used to both reference the class and create a separate variable (such as to hold an instance of the class) in the same context. For example, `box := new Box` would replace the class object in `Box` with an instance of itself. [v1.1.27+]: [#Warn ClassOverwrite](#) enables a warning to be shown at load time for each attempt to overwrite a class.

Within this documentation, the word "class" on its own usually means a class object constructed with the `class` keyword.

Class definitions can contain variable declarations, method definitions and nested class definitions.

Instance Variables [v1.1.01+]

An *instance variable* is one that each instance of the class (that is, each object derived from the class) has its own copy of. They are declared like normal assignments, but the `this.` prefix is omitted (only directly within the class body):

```
InstanceVar := Expression
```

These declarations are evaluated each time a new instance of the class is created with the `new` keyword. The method name `__Init` is reserved for this purpose, and should not be used by the script. The `__New()` method is called after all such declarations have been evaluated, including those defined in base classes. *Expression* can access other instance variables and methods via `this`, but all other variable references are assumed to be global.

To access an instance variable (even within a method), always specify the target object; for example, `this.InstanceVar`.

[v1.1.08+]: Declarations like `x.y := z` are also supported, provided that `x` was previously declared in this class. For example, `x := {}, x.y := 42` declares `x` and also initializes `this.x.y`.

Static/Class Variables [v1.1.00.01+]

Static/class variables belong to the class itself, but can be inherited by derived objects (including sub-classes). They are declared like instance variables, but using the static keyword:

```
static ClassVar := Expression
```

Static declarations are evaluated only once, before the [auto-execute section](#), in the order they appear in the script. Each declaration stores a value in the class object. Any variable references in *Expression* are assumed to be global.

To assign to a class variable, always specify the class object; for example, **ClassName.ClassVar := Value**. If an object *x* is derived from *ClassName* and *x* itself does not contain the key "ClassVar", **x.ClassVar** may also be used to dynamically retrieve the value of **ClassName.ClassVar**. However, **x.ClassVar := y** would store the value in *x*, not in *ClassName*.

[v1.1.08+]: Declarations like **static x.y := z** are also supported, provided that **x** was previously declared in this class. For example, **static x := {}, x.y := 42** declares **x** and also initializes **ClassName.x.y**.

Nested Classes

Nested class definitions allow a class object to be stored inside another class object rather than a separate global variable. In the example above, **class NestedClass** constructs an object and stores it in **ClassName.NestedClass**. Sub-classes could inherit *NestedClass* or override it with their own nested class (in which case **new this.NestedClass** could be used to instantiate whichever class is appropriate).

```
class NestedClass
{
    ...
}
```

Methods

Method definitions look identical to function definitions. Each method has a hidden parameter named `this`, which typically contains a reference to an object derived from the class. However, it could contain a reference to the class itself or a derived class, depending on how the method was called. Methods are stored [by reference](#) in the class object.

```
Method()  
{  
    ...  
}
```

Inside a method, the pseudo-keyword `base` can be used to access the super-class versions of methods or properties which are overridden in a derived class. For example, `base.Method()` in the class defined above would call the version of *Method* which is defined by *BaseClassName*. [Meta-functions](#) are not called; otherwise, `base.Method()` behaves like `BaseClassName.Method.Call(this)`. That is,

- `base.Method()` always invokes the base of the class where the current method was defined, even if `this` is derived from a *sub-class* of that class or some other class entirely.
- `base.Method()` implicitly passes `this` as the first (hidden) parameter.

`base` only has special meaning if followed by a dot `.` or brackets `[]`, so code like `obj := base, obj.Method()` will not work. Scripts can disable the special behaviour of *base* by assigning it a non-empty value; however, this is not recommended. Since the variable *base* must be empty, performance may be reduced if the script omits [#NoEnv](#).

Properties [v1.1.16+]

Property definitions allow a method to be executed whenever the script gets or sets a specific key.

```

Property[]
{
    get {
        return ...
    }
    set {
        return ... := value
    }
}

```

Property is simply the name of the property, which will be used to invoke it. For example, `obj.Property` would call *get* while `obj.Property := value` would call *set*. Within *get* or *set*, `this` refers to the object being invoked. Within *set*, `value` contains the value being assigned.

Parameters can be passed by enclosing them in square brackets to the right of the property name, both when defining the property and when calling it. Aside from using square brackets, parameters of properties are defined the same way as parameters of methods - optional, `ByRef` and variadic parameters are supported.

The return value of *get* or *set* becomes the result of the sub-expression which invoked the property. For example, `val := obj.Property := 42` stores the return value of *set* in `val`.

Each class can define one or both halves of a property. If a class overrides a property, it can use `base.Property` to access the property defined by its base class. If *get* or *set* is not defined, it can be handled by a base class. If *set* is not defined and is not handled by a meta-function or base class, assigning a value stores it in the object, effectively disabling the property.

Internally, *get* and *set* are two separate methods, so cannot share variables (except by storing them in `this`).

[Meta-functions](#) provide a broader way of controlling access to properties and methods of an object, but are more complicated and error-prone.

Construction and Destruction

Whenever a derived object is created with the `new` keyword [requires v1.1.00+], the `__New` method defined by its base object is called. This method can accept parameters, initialize the object and override the result of the `new` operator by returning a value. When an object is destroyed, `__Delete` is called. For example:

```
m1 := new GMem(0, 20)
m2 := {base: GMem}.__New(0, 30)

class GMem
{
    __New(aFlags, aSize)
    {
        this.ptr := DllCall("GlobalAlloc",
"UInt", aFlags, "Ptr", aSize, "Ptr")
        if !this.ptr
            return ""
        MsgBox % "New GMem of " aSize " bytes at
address " this.ptr "."
        return this ; This line can be omitted
when using the 'new' operator.
    }

    __Delete()
    {
        MsgBox % "Delete GMem at address "
this.ptr "."
        DllCall("GlobalFree", "Ptr", this.ptr)
    }
}
```

`__Delete` is not called for any object which has the key `"__Class"`. [Class objects](#) have this key by default.

[v1.1.28+]: If an exception or runtime error is thrown while `__Delete` is executing and is not handled within `__Delete`, it acts as though `__Delete` was called from a new [thread](#). That is, an error dialog is displayed and `__Delete`

returns, but the thread does not exit (unless it was already exiting). Prior to v1.1.28, unhandled exceptions caused inconsistent behavior.

Meta-Functions

Method syntax:

```
class ClassName {  
    __Get([Key, Key2, ...])  
    __Set([Key, Key2, ...], Value)  
    __Call(Name [, Params...])  
}
```

Function syntax:

```
MyGet(this [, Key, Key2, ...])  
MySet(this [, Key, Key2, ...], Value)  
MyCall(this, Name [, Params...])
```

```
ClassName := { __Get: Func("MyGet"), __Set:  
Func("MySet"), __Call: Func("MyCall") }
```

Meta-functions define what happens when a key is requested but not found within the target object. For example, if `obj.key` has not been assigned a value, it invokes the `__Get` meta-function. Similarly, `obj.key := value` invokes `__Set` and `obj.key()` invokes `__Call`. These meta-functions (or methods) would need to be defined in `obj.base`, `obj.base.base` or such.

When the script gets, sets or calls a key which does not exist within the target object, the base object is invoked as follows:

- If this base object defines the appropriate meta-function, call it. If the meta-function explicitly `return s`, use the return value as the result of the operation (whatever caused the meta-function to be called) and return

control to the script. Otherwise, continue as described below.

Set: If the meta-function handled an assignment, it should return the value which was assigned. This allows assignments to be chained, as in `a.x := b.y := z`. The return value may differ from the original value of `z` (for instance, if restrictions are imposed on which values can be assigned).

- Search for a matching key in the base object's own fields.
- [v1.1.16+]: If a key corresponding to a property is found and it implements *get* or *set* (as appropriate), invoke the property and return. If this is a method call, invoke *get*.
- If no key was found, recursively invoke this base object's own base (apply each of these steps to it, starting at the top of this list). If we're not finished yet, search this base object for a matching key again in case one was added by a meta-function.

Due to backward-compatibility, this step is performed for *set* operations even if a key was found (unless it defines a property which implements *set*).

- If multiple parameters were given for *get* or *set* and a key was found, check its value. If that value is an object, handle the remaining parameters by invoking it, and do nothing further.
- If a key was found,
Get: Return the value.
Call: Attempt to call the value, passing the target object as the first parameter (`this`). The value should be a function name or a [function object](#).

If a meta-function stores a matching key in the object but does not `return`, the behaviour is the same as if the key initially existed in the object. For an example using `__Set`, see [Sub-classing Arrays of Arrays](#).

If the operation still hasn't been handled, check if this is a built-in method or property:

- *Get*: If the key is "base", return the object's base.
- *Set*: If the key is "base", set the object's base (or remove it if the value isn't an object).

- *Call*: Call a [built-in method](#) if applicable.

If the operation still hasn't been handled,

- *Get* and *Call*: Return an empty string.
- *Set*: If only one key parameter was given, store the key and value in the target object and return the assigned value. If multiple parameters were given, create a new object and store it using the first parameter as a key, then handle the remaining parameters by invoking the new object. (See [Arrays of Arrays](#).)

Known limitation:

- Using `return` without a value is equivalent to `return ""`. This may be changed in a future version so that `return` can be used to "escape" from a meta-function without overriding the default behaviour.

Dynamic Properties

[Property syntax](#) can be used to define properties which compute a value each time they are evaluated, but each property must be known in advance and defined individually in the script. By contrast, `__Get` and `__Set` can be used to implement properties which aren't known by the script.

For example, a "proxy" object could be created which sends requests for properties over the network (or through some other channel). A remote server would send back a response containing the value of the property, and the proxy would return the value to its caller. Even if the name of each property was known in advance, it would not be logical to define each property individually in the proxy class since every property does the same thing (send a network request). Meta-functions receive the property name as a parameter, so are a good solution for this problem.

Another use of `__Get` and `__Set` is to implement a set of related properties which share code. In the example below they are used to implement a "Color" object with R, G, B and RGB properties, where only the RGB value is actually stored:

```
red := new Color(0xff0000), red.R -= 5
cyan := new Color(0), cyan.G := 255, cyan.B :=
```

255

```
MsgBox % "red: " red.R ", " red.G ", " red.B " = "  
red.RGB
```

```
MsgBox % "cyan: " cyan.R ", " cyan.G ", " cyan.B "  
= " cyan.RGB
```

```
class Color
```

```
{
```

```
    __New(aRGB)
```

```
    {
```

```
        this.RGB := aRGB
```

```
    }
```

```
    static Shift := {R:16, G:8, B:0}
```

```
    __Get(aName)
```

```
    {
```

```
        ; NOTE: Using this.Shift here would  
        cause an infinite loop!
```

```
        shift := Color.Shift[aName] ; Get the  
        number of bits to shift.
```

```
        if (shift != "") ; Is it a known  
        property?
```

```
            return (this.RGB >> shift) & 0xff
```

```
        ; NOTE: Using 'return' here would break  
        this.RGB.
```

```
    }
```

```
    __Set(aName, aValue)
```

```
    {
```

```
        if ((shift := Color.Shift[aName]) != "")  
        {
```

```
            aValue &= 255 ; Truncate it to the  
            proper range.
```

```
            ; Calculate and store the new RGB
```

```

value.
        this.RGB := (aValue << shift) |
        (this.RGB & ~(0xff << shift))

        ; 'Return' must be used to indicate
a new key-value pair should not be created.
        ; This also defines what will be
stored in the 'x' in 'x := clr[name] := val':
        return aValue
    }
    ; NOTE: Using 'return' here would break
this._RGB and this.RGB.
}

; Meta-functions can be mixed with
properties:
RGB {
    get {
        ; Return it in hex format:
        return format("0x{:06x}", this._RGB)
    }
    set {
        return this._RGB := value
    }
}
}

```

However, in this case [Property syntax](#) could have been used instead, where code is shared by simply having each property call a central method. It is better to avoid using meta-functions where possible due to the high risk of misuse (see the notes in red above).

Objects as Functions

For an outline of how to create objects which can act as functions, see [Function Objects](#).

A function object can also act as a meta-function, such as to define dynamic properties similar to those in the previous section. Although it is recommended to use [property syntax](#) instead, the example below shows the potential of meta-functions for implementing new concepts or behaviour, or changing the structure of the script.

```
blue := new Color(0x0000ff)
MsgBox % blue.R ", " blue.G ", " blue.B

class Properties extends FunctionObject
{
    Call(aTarget, aName, aParams*)
    {
        ; If this Properties object contains a
        definition for this half-property, call it.
        if ObjHasKey(this, aName)
            return this[aName].Call(aTarget,
aParams*)
    }
}

class Color
{
    __New(aRGB)
    {
        this.RGB := aRGB
    }

    class __Get extends Properties
    {
        R() {
            return (this.RGB >> 16) & 255
        }
        G() {
            return (this.RGB >> 8) & 255
        }
        B() {
```

```

        return this.RGB & 255
    }
}
;...
}

```

Sub-classing Arrays of Arrays

When a [multi-parameter assignment](#) such as `table[x, y] := content` implicitly causes a new object to be created, the new object ordinarily has no base and therefore no custom methods or special behaviour. `__Set` may be used to initialize these objects, as demonstrated below.

```

x := {base: {addr: Func("x_Addr"), __Set:
Func("x_Setter")}}

; Assign value, implicitly calling x_Setter to
create sub-objects.
x[1,2,3] := "...

; Retrieve value and call example method.
MsgBox % x[1,2,3] "`n" x.addr() "`n" x[1].addr()
"`n" x[1,2].addr()

x_Setter(x, p1, p2, p3) {
    x[p1] := new x.base
}

x_Addr(x) {
    return &x
}

```

Since `x_Setter` has four mandatory parameters, it will only be called when there are two or more key parameters. When the assignment above occurs, the following takes place:

- `x[1]` does not exist, so `x_Setter(x, 1, 2, 3)` is called (`"..."` is not passed as there are too few parameters).
 - `x[1]` is assigned a new object with the same base as `x`.
 - No value is returned – the assignment continues.
- `x[1][2]` does not exist, so `x_Setter(x[1], 2, 3, "...")` is called.
 - `x[1][2]` is assigned a new object with the same base as `x[1]`.
 - No value is returned – the assignment continues.
- `x[1][2][3]` does not exist, but since `x_Setter` requires four parameters and there are only three (`x[1][2]`, `3`, `"..."`), it is not called and the assignment completes as normal.

Default Base Object

When a non-object value is used with object syntax, the *default base object* is invoked. This can be used for debugging or to globally define object-like behaviour for strings, numbers and/or variables. The default base may be accessed by using `.base` with any non-object value; for instance, `"".base`. Although the default base cannot be *set* as in `"".base := Object()`, the default base may itself have a base as in `"".base.base := Object()`.

Automatic Var Init

When an empty variable is used as the target of a *set* operation, it is passed directly to the `__Set` meta-function, giving it opportunity to insert a new object into the variable. For brevity, this example does not support multiple parameters; it could, by using a [variadic function](#).

```
"".base.__Set :=
Func("Default_Set_AutomaticVarInit")

empty_var.foo := "bar"
MsgBox % empty_var.foo

Default_Set_AutomaticVarInit(ByRef var, key,
value)
{
    if (var = "")
        var := Object(key, value)
}
```

Pseudo-Properties

Object "syntax sugar" can be applied to strings and numbers.

```
"".base.__Get :=
Func("Default_Get_PseudoProperty")
```

```

"" .base.is      := Func("Default_is")

MsgBox % A_AhkPath.length " == "
StrLen(A_AhkPath)
MsgBox % A_AhkPath.length.is("integer")

Default_Get_PseudoProperty(nonobj, key)
{
    if (key = "length")
        return StrLen(nonobj)
}

Default_is(nonobj, type)
{
    if nonobj is %type%
        return true
    return false
}

```

Note that built-in functions may also be used, but in this case the parentheses cannot be omitted:

```

"" .base.length := Func("StrLen")
MsgBox % A_AhkPath.length() " == "
StrLen(A_AhkPath)

```

Debugging

If allowing a value to be treated as an object is undesirable, a warning may be shown whenever a non-object value is invoked:

```

"" .base.__Get := "" .base.__Set := "" .base.__Call
:= Func("Default__Warn")

empty_var.foo := "bar"

```

```
x := (1 + 1).is("integer")

Default__Warn(nonobj, p1="", p2="", p3="",
p4="")
{
    ListLines
    MsgBox A non-object value was improperly
invoked.`n`nSpecifically: %nonobj%
}
```

Implementation

Reference-Counting

AutoHotkey uses a basic reference-counting mechanism to automatically free the resources used by an object when it is no longer referenced by the script. Script authors should not invoke this mechanism explicitly, except when dealing directly with unmanaged [pointers to objects](#).

Currently in AutoHotkey v1.1, temporary references created within an expression (but not stored anywhere) are released immediately after use. For example, `Fn(&{ })` passes an invalid address to the function, because the temporary reference returned by `{ }` is released immediately after the [address-of](#) operator is evaluated.

To run code when the last reference to an object is being released, implement the [__Delete](#) meta-function.

Known Limitations:

- Circular references must be broken before an object can be freed. For details and an example, see [Freeing Objects](#).
- Although references in static and global variables are released automatically when the program exits, references in non-static local variables or on the expression evaluation stack are not. These references are only released if the function or expression is allowed to complete normally.

Although memory used by the object is reclaimed by the operating system when the program exits, [__Delete](#) will not be called unless all references to the object are freed. This can be important if it frees other resources which are not automatically reclaimed by the operating system, such as temporary files.

Pointers to Objects

In some rare cases it may be necessary to pass an object to external code via `DllCall` or store it in a binary data structure for later retrieval. An object's address can be retrieved via `address := &object ;` however, this effectively

makes two references to the object, but the program only knows about the one in *object*. If the last *known* reference to the object was released, the object would be deleted. Therefore, the script must inform the object that it has gained a reference. There are two ways to do this:

```
; Method #1: Explicitly increment the reference
count.
address := &object
ObjAddRef(address)

; Method #2: Use Object(), which increments the
reference count and returns an address.
address := Object(object)
```

This function can also be used to convert an address back into a reference:

```
object := Object(address)
```

Either way, the script must also inform the object when it is finished with that reference:

```
; Decrement the object's reference count to
allow it to be freed:
ObjRelease(address)
```

Generally each new copy of an object's address should be treated as another reference to the object, so the script should call `ObjAddRef` when it gains a copy and `ObjRelease` immediately before losing one. For example, whenever an address is copied via something like `x := address`, `ObjAddRef` should be called. Similarly, when the script is finished with `x` (or is about to overwrite `x`'s value), it should call `ObjRelease`.

Note that the `Object()` function can be used even on objects which it did not create, such as [COM objects](#) and [File objects](#).

Frequently Asked Questions (FAQ)

Language Syntax

- When are quotation marks used with commands and their parameters?
- When exactly are variable names enclosed in percent signs?
- When should percent signs and commas be escaped?

General Troubleshooting

- What can I do if AutoHotkey won't install?
- Why do some lines in my script never execute?
- Why doesn't my script work on Windows xxx even though it worked on a previous version?
- How do I work around problems caused by User Account Control (UAC)?
- I can't edit my script via tray icon because it won't start due to an error. Can I find my script somewhere else?
- How can I find and fix errors in my code?
- Why is the Run command unable to launch my game or program?
- Why are the non-ASCII characters in my script displaying or sending incorrectly?
- Why don't Hotstrings, Send, and MouseClick work in certain games?
- How can performance be improved for games or at other times when the CPU is under heavy load?
- My antivirus program flagged AHK as malware. Does it really contain a virus?

Common Tasks

- Where can I find the official build, or older releases?
- Can I run AHK from a USB drive?
- How can the output of a command line operation be retrieved?
- How can a script close, pause, or suspend other script(s)?
- How can a repeating action be stopped without exiting the script?
- How can context sensitive help for AutoHotkey commands be used in any editor?
- How to detect when a web page is finished loading?

- How can dates and times be compared or manipulated?
- How can I send the current Date and/or Time?
- How can I send text to a window which isn't active or isn't visible?
- How can Winamp be controlled even when it isn't active?
- How can MsgBox's button names be changed?
- How can I change the default editor, which is accessible via context menu or tray icon?
- How can I save the contents of my GUI associated variables?
- Can I draw something with AHK?
- How can I start an action when a window appears, closes or becomes [in]active?

Hotkeys, Hotstrings, and Remapping

- How do I put my hotkeys and hotstrings into effect automatically every time I start my PC?
- I'm having trouble getting my mouse buttons working as hotkeys. Any advice?
- How can tab and space be defined as hotkeys?
- How can keys or mouse buttons be remapped so that they become different keys?
- How do I detect the double press of a key or button?
- How can a hotkey or hotstring 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.
- How can a prefix key be made to perform its native function rather than doing nothing?
- How can the built-in Windows shortcut keys, such as Win+U (Utility Manager) and Win+R (Run), be changed or disabled?
- Can I use wildcards or regular expressions in Hotstrings?
- How can I use a hotkey that is not in my keyboard layout?
- My keypad has a special 000 key. Is it possible to turn it into a hotkey?

Language Syntax

When are quotation marks used with commands and their parameters?

Double quotes (") have special meaning only within [expressions](#). In all other places, they are treated literally as if they were normal characters. However, when a script launches a program or document, the operating system usually requires quotes around any command-line parameter that contains spaces, such as in this example: `Run, Notepad.exe "C:\My Documents\Address List.txt"`.

When exactly are variable names enclosed in percent signs?

Variable names are always enclosed in percent signs except in cases illustrated in **bold** below:

- 1) In parameters that are input or output variables: `StringLen`, **OutputVar**, **InputVar**
- 2) On the left side of an assignment: `Var = 123abc`
- 3) On the left side of [traditional \(non-expression\) if-statements](#): `If Var1 < %Var2%`
- 4) Everywhere in [expressions](#). For example:

```
If (Var1 <> Var2)
    Var1 := Var2 + 100
```

When should percent signs and commas be escaped?

Literal percent signs must be [escaped](#) by preceding them with an accent/backtick. For example: `MsgBox The current percentage is 25`%.` Literal commas must also be escaped (` ,) except when used in [MsgBox](#) or the last parameter of any command (in which case the accent is permitted but not necessary).

When commas or percent signs are enclosed in quotes within an [expression](#), the accent is permitted but not necessary. For example: `Var := "15%"`.

General Troubleshooting

What can I do if AutoHotkey won't install?

7-zip Error: Use 7-zip or a compatible program to extract the setup files from the installer EXE, then run setup.exe or Installer.ahk (drag and drop Installer.ahk onto AutoHotkeyU32.exe).

AutoHotkey's installer comes packaged as a 7-zip self-extracting archive which attempts to extract to the user's Temp directory and execute a compiled script. Sometimes system policies or other factors prevent the files from being extracted or executed. Usually in such cases the message "7-zip Error" is displayed. Manually extracting the files to a different directory may help.

Setup hangs: If the setup window comes up blank or not at all, try one or both of the following:

- Hold **Ctrl** **Shift** installer starts. If you get a UAC prompt, hold **Ctrl** **Shift** click Yes/Continue. You should get a prompt asking whether you want to install with default options.
- Install using [command line options](#). If you have manually extracted the setup files from the installer EXE, use either `setup.exe /S` or `AutoHotkeyU32.exe Installer.ahk /S`.

Other: The suggestions above cover the most common problems. For further assistance, post on the forums.

Why do some lines in my script never execute?

Any lines you want to execute immediately when the script starts should appear at the top of the script, prior to the first [hotkey](#), [hotstring](#), or [Return](#). For details, see [auto-execute section](#).

Also, a [hotkey](#) that executes more than one line must list its first line *beneath* the hotkey, not on the same line. For example:

```
#space:: ; Win+Spacebar
Run Notepad
WinWaitActive Untitled - Notepad
WinMaximize
return
```

Why doesn't my script work on Windows xxx even though it worked on a previous version?

There are many variations of this problem, such as:

- I've upgraded my computer/Windows and now my script won't work.
- Hotkeys/hotstrings don't work when a program running as admin is active.
- Some windows refuse to be automated (e.g. Device Manager ignores Send).

If you've switched operating systems, it is likely that something else has also changed and may be affecting your script. For instance, if you've got a new computer, it might have different drivers or other software installed. If you've also updated to a newer version of AutoHotkey, find out which version you had before and then check the [changelog](#) and [compatibility notes](#).

[SoundGet](#), [SoundSet](#), [SoundGetWaveVolume](#) and [SoundSetWaveVolume](#) behave differently on Vista and later than on earlier versions of Windows. In particular, device numbers are different and some components may be unavailable. Behaviour depends on the audio drivers, which are necessarily different to the ones used on XP. The [soundcard analysis script](#) can be used to find the correct device numbers.

Also refer to the following question:

How do I work around problems caused by User Account Control (UAC)?

[User Account Control \(UAC\)](#) is a common cause of problems, especially when moving from Windows XP/Vista/7 to Vista/7/8/10. Although it is present in Windows Vista and later, it is enabled by default on new systems or new installs, and it is more difficult to disable on Windows 8 and later.

By default, UAC protects "elevated" programs (that is, programs which are running as admin) from being automated by non-elevated programs, since that would allow them to bypass security restrictions. Hotkeys are also blocked, so for instance, a non-elevated program cannot spy on input intended for an elevated program.

UAC may also prevent [SendPlay](#) and [BlockInput](#) from working.

Common workarounds are as follows:

- Enable the [Add 'Run with UI Access' to context menus](#) option in AutoHotkey Setup. This option can be enabled or disabled without reinstalling AutoHotkey by re-running AutoHotkey Setup from the Start menu. Once it is enabled, launch your script file by right-clicking it and selecting *Run with UI Access*, or use a [command line](#) like `"AutoHotkeyU32_UIA.exe" "Your script.ahk"` (but include full paths).
- Run the script [as administrator](#). Note that this also causes any programs launched by the script to run as administrator, and may require the user to accept an approval prompt when launching the script.
- Disable the local security policy "Run all administrators in Admin Approval Mode" (not recommended).
- Disable UAC completely. This is not recommended, and is not feasible on Windows 8 or later.

I can't edit my script via tray icon because it won't start due to an error. What do I do?

You need to fix the error in your script before you can get your tray icon back. But first, you need to find the script file.

Look for AutoHotkey.ahk in the following directories:

- Your *Documents* (or *My Documents*) folder.
- The directory where you installed AutoHotkey, usually C:\Program Files\AutoHotkey. If you are using AutoHotkey without having installed it, look in the directory which contains AutoHotkey.exe.

If you are running another AutoHotkey executable directly, the name of the

script depends on the executable. For example, if you are running AutoHotkeyU32.exe, look for AutoHotkeyU32.ahk. Note that depending on your system settings the ".ahk" part may be hidden, but the file should have an icon like 

You can usually edit a script file by right clicking it and selecting *Edit Script*. If that doesn't work, you can open the file in Notepad or another editor.

If you launch AutoHotkey from the Start menu or by running AutoHotkey.exe directly (without command line parameters), it will look for a script in one of the locations shown above. Alternatively, you can create a script file (something.ahk) anywhere you like, and run the script file instead of running AutoHotkey.

See also [Command Line Parameter "Script Filename"](#) and [Portability of AutoHotkey.exe](#).

How can I find and fix errors in my code?

For simple scripts, see [Debugging a Script](#). To show contents of a variable, use [MsgBox](#) or [ToolTip](#). For complex scripts, see [Interactive Debugging](#).

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). For example:

```
Run, %A_ProgramFiles%\Some Application\App.exe,  
%A_ProgramFiles%\Some Application
```

If the program you are trying to start is in `%A_WinDir%\System32` and you are using AutoHotkey 32-bit on a 64-bit system, the [File System Redirector](#) may be interfering. To work around this, use `%A_WinDir%\SysNative` instead; this is a virtual directory only visible to 32-bit programs running on 64-bit systems.

Why are the non-ASCII characters in my script displaying or

sending incorrectly?

Short answer: Save the script as UTF-8 with BOM.

Although AutoHotkey supports Unicode text, it is optimized for backward-compatibility, which means defaulting to the ANSI encoding rather than the more internationally recommended UTF-8. AutoHotkey will not automatically recognize a UTF-8 file unless it begins with a byte order mark.

In other words, UTF-8 files which lack a byte order mark are misinterpreted, causing non-ASCII characters to be decoded incorrectly. To resolve this, save the file as UTF-8 with BOM or [add the /CP65001 command line switch](#).

To save as UTF-8 with BOM in Notepad, select *UTF-8* from the *Encoding* drop-down in the Save As dialog.

To read other UTF-8 files which lack a byte order mark, use `FileEncoding UTF-8-RAW`, the `*P65001` option with `FileRead`, or `"UTF-8-RAW"` for the third parameter of `FileOpen`. The `-RAW` suffix can be omitted, but in that case any newly created files will have a byte order mark.

Note that INI files accessed with the standard INI commands do not support UTF-8; they must be saved as ANSI or UTF-16.

Why do Hotstrings, Send, and Click have no effect in certain games?

Not all games allow AHK to send keys and clicks or receive pixel colors.

But there are some alternatives, try all the solutions mentioned below. If all these fail, it may not be possible for AHK to work with your game. Sometimes games have a hack and cheat prevention measure, such as GameGuard and Hackshield. If they do, there is a high chance that AutoHotkey will not work with that game.

- Use SendPlay via the `SendPlay` command, `SendMode Play` and/or the `hotstring option SP`.

```
SendPlay, abc
```

```
SendMode, Play
Send, abc
```

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

```
; or
```

```
#Hotstring SP
::btw::by the way
```

Note: SendPlay may have no effect at all on Windows Vista or later if User Account Control is enabled, even if the script is running as an administrator.

- Increase [SetKeyDelay](#). For example:

```
SetKeyDelay, 0, 50
SetKeyDelay, 0, 50, Play
```

- Try [ControlSend](#), which might work in cases where the other Send modes fail:

```
ControlSend,, abc, game_title
```

- Try the down and up event of a key with the various send methods:

```
Send {KEY down}{KEY up}
```

- Try the down and up event of a key with a [Sleep](#) between them:

```
Send {KEY down}
Sleep 10 ; try various milliseconds
```

```
Send {KEY up}
```

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

If a script's [Hotkeys](#), [Clicks](#), or [Sends](#) are noticeably slower than normal while the CPU is under heavy load, raising the script's process-priority may help. To do this, include the following line near the top of the script:

```
Process, Priority, , High
```

My antivirus program flagged AutoHotkey or a compiled script as malware. Is it really a virus?

Although it is certainly possible that the file has been infected, most often these alerts are *false positives*, meaning that the antivirus program is mistaken. One common suggestion is to upload the file to an online service such as [virustotal](#) or [Jotti](#) and see what other antivirus programs have to say. If in doubt, you could send the file to the vendor of your antivirus software for confirmation. This might also help us and other AutoHotkey users, as the vendor may confirm it is a false positive and fix their product to play nice with AutoHotkey.

False positives might be more common for compiled scripts which have been compressed, such as with UPX (default for AutoHotkey 1.0 but not 1.1) or MPRESS (optional for AutoHotkey 1.1). As the default AutoHotkey installation does not include a compressor, compiled scripts are not compressed by default.

Common Tasks

Where can I find the official build, or older releases?

See [download page of AutoHotkey](#).

Can I run AHK from a USB drive?

See [Portability of AutoHotkey.exe](#). Note that if you use auto-included function libraries, AutoHotkey.exe and the Lib folder must be up one level from Ahk2Exe.exe (e.g. \AutoHotkey.exe vs \Compiler\Ahk2Exe.exe). Also note that Ahk2Exe saves settings to the following registry key:

```
HKCU\Software\AutoHotkey\Ahk2Exe .
```

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

Testing shows that due to file caching, a temporary file can be very fast for relatively small outputs. In fact, if the file is deleted immediately after use, it often does not actually get written to disk. For example:

```
RunWait %ComSpec% /c dir > C:\My Temp File.txt
FileRead, VarToContainContents, C:\My Temp
File.txt
FileDelete, C:\My Temp File.txt
```

To avoid using a temporary file (especially if the output is large), consider using the [Shell.Exec\(\)](#) method as shown in the examples for the [Run](#) command.

How can a script close, pause, or suspend other script(s)?

First, here is an example that closes another script:

```
DetectHiddenWindows On ; Allows a script's
hidden main window to be detected.
SetTitleMatchMode 2 ; Avoids the need to
```

```
specify the full path of the file below.  
WinClose Script's File Name.ahk - AutoHotkey ;  
Update this to reflect the script's name (case  
sensitive).
```

To [suspend](#) or [pause](#) another script, replace the last line above with one of these:

```
PostMessage, 0x111, 65305,,, Script's File  
Name.ahk - AutoHotkey ; Suspend.  
PostMessage, 0x111, 65306,,, Script's File  
Name.ahk - AutoHotkey ; Pause.
```

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

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.
```

To stop an action that is repeating inside a [Loop](#), consider the following working example, which is a hotkey that 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:: ; Win+Z hotkey (change this hotkey to suit  
your preferences).  
#MaxThreadsPerHotkey 1  
if KeepWinZRunning ; This means an underlying  
thread is already running the loop below.  
{  
    KeepWinZRunning := false ; Signal that  
thread's loop to stop.  
    return ; End this thread so that the one
```

```

underneath will resume and see the change made
by the line above.
}
; Otherwise:
KeepWinZRunning := true
Loop
{
    ; The next four lines are the action you
    want to repeat (update them to suit your
    preferences):
    ToolTip, Press Win-Z again to stop this from
    flashing.
    Sleep 1000
    ToolTip
    Sleep 1000
    ; But leave the rest below unchanged.
    if not KeepWinZRunning ; The user signaled
    the loop to stop by pressing Win-Z again.
        break ; Break out of this loop.
}
KeepWinZRunning := false ; Reset in preparation
for the next press of this hotkey.
return

```

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

Rajat created [this script](#).

How to detect when a web page is finished loading?

With Internet Explorer, perhaps the most reliable method is to use DllCall and COM as demonstrated at www.autohotkey.com/forum/topic19256.html. On a related note, the contents of the address bar and status bar can be retrieved as demonstrated at www.autohotkey.com/forum/topic19255.html.

Older, less reliable method: 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! -
WinActivate
StatusBarWait, Done, 30
if ErrorLevel
    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. Finally, there are several built-in [date/time variables](#), as well as the [FormatTime](#) command to create a custom date/time string.

How can I send the current Date and/or Time?

Use [FormatTime](#) or [built-in variables for date and time](#).

How can I send text to a window which isn't active or isn't visible?

Use [ControlSend](#).

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

See [Automating Winamp](#).

How can MsgBox's button names be changed?

Here is an [example](#).

How can I change the default editor, which is accessible via context menu or tray icon?

In the example section of [Edit](#) you will find a script that allows you to change the default editor.

How can I save the contents of my GUI associated variables?

Use [Gui Submit](#). For Example:

```
Gui, Add, Text,, Enter some Text and press  
Submit:  
Gui, Add, Edit, vAssociatedVar  
Gui, Add, Button,, Submit  
Gui, Show  
Return  
  
ButtonSubmit:  
Gui, Submit, NoHide  
MsgBox, Content of the edit control:  
%AssociatedVar%  
Return
```

Can I draw something with AHK?

See [GDI+ standard library](#) by tic. It's also possible with some rudimentary methods using Gui, but in a limited way.

How can I start an action when a window appears, closes or becomes [in]active?

Use [WinWait](#), [WinWaitClose](#) or [WinWait\[Not\]Active](#).

There are also user-created solutions such as [OnWin.ahk](#) and [\[How to\] Hook on to Shell to receive its messages](#).

Hotkeys, Hotstrings, and Remapping

How do I put my hotkeys and hotstrings into effect automatically every time I start my PC?

There are several ways to make a script (or any program) launch automatically every time you start your PC. The easiest is to place a shortcut to the script in the Startup folder:

1. Find the script file, select it, and press **Control** **C**
2. Press **Win** **R** to open the Run dialog, then enter `shell:startup` and click OK or **Enter** to open the Startup folder for the current user. To instead open the folder for all users, enter `shell:common startup` (however, in that case you must be an administrator to proceed).
3. Right click inside the window, and click "Paste Shortcut". The shortcut to the script should now be in the Startup folder.

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** button 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 that you want to use -- 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** **F1** can then indirectly configure that button as a hotkey by using `^F1::` in a 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 Tab and Space be defined as hotkeys?

Use the names of the keys (Tab and Space) rather than their characters. For

example, #Space is   ^!Tab is   

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

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

How do I detect the double press of a key or button?

Use [built-in variables for hotkeys](#) as follows:

```
~Ctrl::  
    if (A_ThisHotkey = A_PriorHotkey &&  
        A_TimeSincePriorHotkey < 200)  
        MsgBox double-press  
    return
```

How can a hotkey or hotstring 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.

The preferred method is [#IfWinActive](#). For example:

```
#IfWinActive, ahk_class Notepad  
^a::MsgBox You pressed Control-A while Notepad  
is active.
```

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:

```
$Numpad0::Send, {Numpad0}
```

The \$ prefix is needed to prevent a warning dialog about an infinite loop (since the hotkey "sends itself"). In addition, the above action occurs at the time the key is **released**.

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](#).

Can I use wildcards or regular expressions in Hotstrings?

Use the [script](#) by polyethene (examples are included).

How can I use a hotkey that is not in my keyboard layout?

See [Special Keys](#).

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

Yes. This [example script](#) makes the `000` `[REDACTED]` to an equals key. You can change the action by replacing the `Send, =` line with line(s) of your choice.

AutoHotkey Beginner Tutorial

Table of Contents

1. The Basics
 1. Downloading and installing AutoHotkey
 2. How to create a script
 3. You cannot merge commands
 4. How to find the help file on your computer
2. Hotkeys & Hotstrings
 1. Keys and their mysterious symbols
 2. Window specific hotkeys/hotstrings
 3. Multiple hotkeys/hotstrings per file
 4. Examples
3. Sending Keystrokes
 1. Games
4. Running Programs & Websites
5. Commands vs. Functions()
 1. Code blocks
6. Variables
 1. When to use percents
 2. Getting user input
 3. Other Examples?
7. Objects
 1. Creating Objects
 2. Using Objects
8. Other Helpful Goodies
 1. The mysterious square brackets
 2. Finding your AHK version
 3. Trial and Error
 4. Indentation
 5. Asking for Help
 6. Other links

1 - The Basics

Before we begin our journey, let me give some advice. Throughout this tutorial you will see a lot of text and a lot of code. For optimal learning power, it is advised that you read the text and **try** the code. Then, study the code. You can copy and paste most examples on this page. If you get confused, try reading the section again.

a. Downloading and installing AutoHotkey

Before learning to use AutoHotkey (AHK), you will need to download it. After downloading it, you may possibly need to install it. But that depends on the version you want. For this guide we will use the Installer since it is easiest to set up.

Text instructions:

1. Go to the AutoHotkey Homepage: <https://autohotkey.com/>
2. Click Download: <https://autohotkey.com/download/ahk-install.exe>
3. During installation of AutoHotkey, you will be asked to choose from UNICODE or ANSI. In short, you would probably want to choose UNICODE. It has support for non-English letters and numbers (characters). Keep going until you see an Install button.
4. Once done, great! Continue on to section b.

For a video instruction, watch [Install and Hello World](#) on YouTube.

b. How to create a script

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

Text instructions:

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

So now that you have created a script, we need to add stuff into the file. For a list of all built-in commands, function and variables, see [section 5](#).

Here is a very basic script containing a hotkey which types text using the [Send](#) command when the hotkey is pressed:

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

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

- The first line: `^j::` is the hotkey. `^` means **Ctrl** `j` the letter **J** anything to the **left** of `::` are the keys you need to press.
 - The second line: `Send, My First Script` is how you **send** keystrokes. `Send` is the command, anything after the comma (,) will be typed.
 - The third line: `return`. This will become your best friend. It literally **stops** code from going any further, to the lines below. This will prevent many issues when you start having a lot of stuff in your scripts.
8. Save the File.
 9. Double-click the file/icon in the desktop to run it. Open notepad or (anything you can type in) and press **Ctrl** `J`
 10. Hip Hip Hooray! Your first script is done. Go get some reward snacks then return to reading the rest of this tutorial.

For a video instruction, watch [Install and Hello World](#) on YouTube.

c. You cannot merge commands

When you are making your code, you might have the urge to put several commands on the same line or inside of each other, don't. [In section 5](#) we'll talk about why it doesn't work as you might expect and what you can do instead.

d. How to find the help file on your computer

There are a few ways to do this, I'll assume you have it installed to the default locations:

Method 1:

1. Find the Start menu or Start Orb on your screen, usually in the lower left.
2. Click **Programs** or **All Programs**.
3. Find **AutoHotkey** in the list.
4. You should then see **AutoHotkey Help File**. Click it.
5. Done!

Method 2:

1. Go to your desktop.
2. Find **My Computer** or **Computer**. Open it.
3. Go into your harddrive that contains **AutoHotkey**. Probably **C:** drive.
4. Search within all **Program Files** folders for **AutoHotkey**.
5. Look for **AutoHotkey.chm** or a file that says AutoHotkey and has a yellow question mark on it.
6. Done!

2 - Hotkeys & Hotstrings

What is a hotkey? A hotkey is a key that is hot to the touch. ... Just kidding. It is a key or key combination that the person at the keyboard presses to trigger some actions. For example:

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

What is a hotstring? Hotstrings are mainly used to expand abbreviations as you type them (auto-replace), they can also be used to launch any scripted action. For example:

```
::ftw::Free the whales
```

The difference between the two examples is that the hotkey will be triggered when you press `Ctrl` `J` while the hotstring will convert your typed "ftw" into "Free the whales".

"So, how exactly does a person such as myself create a hotkey?" Good question. A hotkey is created by using a single pair of colons. The key or key combo needs to go on the **left** of the `::`. And the content needs to go below, followed by a `return`.

Note: There are exceptions, but those tend to cause confusion a lot of the time. So it won't be covered in the tutorial, at least, not right now.

```
Esc::  
MsgBox, Escape!!!!  
return
```

A hotstring has a pair of colons on each side of the text you want to trigger the

text replacement. While the text to replace your typed text goes on the **right** of the second pair of colons.

Hotstrings, as mentioned above, can also launch scripted actions. That's fancy talk for "do pretty much anything". Same with hotkeys.

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

A nice thing to know is that you can have many lines of code for each hotkey, hotstring, label, and a lot of other things we haven't talked about yet.

```
^j::  
MsgBox, Wow!  
MsgBox, There are  
Run, notepad.exe  
WinActivate, Untitled - Notepad  
WinWaitActive, Untitled - Notepad  
Send, 7 lines{!}{Enter}  
SendInput, inside the CTRL{+}J hotkey.  
return
```

a. Keys and their mysterious symbols

You might be wondering "How the crud am I supposed to know that ^ means `Ctrl` [redacted]", good question. To help you learn what ^ and other symbols mean, gaze upon this chart:

Symbol	Description
#	Win [redacted] (ows logo key)
!	Alt [redacted]
^	Control [redacted]

+	Shift 
&	An ampersand may be used between any two keys or mouse buttons to combine them into a custom hotkey.

(For the full list of symbols, see the [Hotkey](#) page)

Additionally, for a list of all/most hotkey names that can be used on the **left** side of a hotkey's double-colon, see [List of Keys, Mouse Buttons, and Joystick Controls](#).

You can define a custom combination of two (and only two) keys (except joystick buttons) by using `&` between them. In the example below, you would hold down Numpad0 then press Numpad1 or Numpad2 to trigger one of the hotkeys:

```
Numpad0 & Numpad1::  
MsgBox, You pressed Numpad1 while holding down  
Numpad0.  
return  
  
Numpad0 & Numpad2::  
Run, notepad.exe  
return
```

But you are now wondering if hotstrings have any cool modifiers since hotkeys do. Yes, they do! Hotstring modifiers go between the first set of colons. For example:

```
:*:ftw::Free the whales
```

Visit [Hotkeys](#) and [Hotstrings](#) for additional hotkey and hotstring modifiers, information and examples.

b. Window specific hotkeys/hotstrings

Sometime you might want a hotkey or hotstring to only work (or be disabled) in a certain window. To do this, you will need to use either of these fancy commands with a # in-front of them:

```
#IfWinActive  
#IfWinExist
```

These special commands (technically called "directives") create context-sensitive hotkeys and hotstrings. Simply specify a window title. But in some cases you might want to specify criteria such as HWND, group or class. Those are a bit advanced and are covered more in-depth here: [The WinTitle Parameter & the Last Found Window](#).

```
#IfWinActive Untitled - Notepad  
#Space::  
MsgBox, You pressed WIN+SPACE in Notepad.  
return
```

To turn off context sensitivity for subsequent hotkeys or hotstrings, specify any #IfWin directive but leave all of its parameters blank. For example:

```
; Untitled - Notepad  
#IfWinActive Untitled - Notepad  
!q::  
MsgBox, You pressed ALT+Q in Notepad.  
return  
  
; Any window that isn't Untitled - Notepad  
#IfWinActive  
!q::  
MsgBox, You pressed ALT+Q in any window.  
return
```

When #IfWin directives are never used in a script, all hotkeys and hotstrings are enabled for all windows.

The #IfWin directives are positional: they affect all hotkeys and hotstrings physically beneath them in the script. They are also mutually exclusive; that is, only the most recent one will be in effect.

```
; Notepad
#IfWinActive ahk_class Notepad
#Space::
MsgBox, You pressed WIN+SPACE in Notepad.
return
::msg::You typed msg in Notepad

; MSPaint
#IfWinActive Untitled - Paint
#Space::
MsgBox, You pressed WIN+SPACE in MSPaint!
return
::msg::You typed msg in MSPaint!
```

For more in-depth information and similar commands, check out the [#IfWinActive](#) page.

c. Multiple hotkeys/hotstrings per file

This, for some reason crosses some people's minds. So I'll set it clear: AutoHotkey has the ability to have *as many* hotkeys and hotstrings in one file as you want. Whether it's 1, or 3253 (or more).

```
#i::
Run, http://www.google.com/
return

^p::
Run, notepad.exe
return
```

```
~j::  
Send, ack  
return  
  
*:acheiv::achiev  
:achievement::achievement  
:acquaintance::acquaintance  
*:adquir::acquir  
:aquisition::acquisition  
*:agrat::agrat  
*:align::align  
:ameria::America
```

The above code is perfectly acceptable. Multiple hotkeys, multiple hotstrings. All in one big happy script file.

d. Examples

```
::btw::by the way ; Replaces "btw" with "by the way" as soon as you press an default ending character.
```

```
*:btw::by the way ; Replaces "btw" with "by the way" without needing an ending character.
```

```
^n:: ; CTRL+N hotkey  
Run, notepad.exe ; Run Notepad when you press CTRL+N.  
return ; This ends the hotkey. The code below this will not get triggered.
```

```
^b:: ; CTRL+B hotkey  
Send, {Ctrl down}c{Ctrl up} ; Copies the selected text. ^c could be used as well, but this method is
```

more secure.

SendInput, [b]{Ctrl down}v{Ctrl up}[/b] ; Wraps the selected text in BBCode tags to make it bold in a forum.

return ; This ends the hotkey. The code below this point will not get triggered.

3 - Sending Keystrokes

So now you decided that you want to send (type) keys to a program. We can do that. Use the [Send](#) command. This command literally sends keystrokes, to simulate typing or pressing of keys.

But before we get into things, we should talk about some common issues that people have.

Just like hotkeys, the Send command has special keys too. [Lots and lots of them](#). Here are the four most common symbols:

Symbol	Description
!	Sends the Alt key. For example, <code>Send, This is text!a</code> would send the keys "This is text" and then press Alt A . Note: <code>!A</code> produces a different effect in some programs than <code>!a</code> . This is because <code>!A</code> presses Alt A while <code>!a</code> presses Alt A . In doubt, use lowercase.
+	Sends the Shift key. For example, <code>Send, +abC</code> would send the text "AbC", and <code>Send, !+a</code> would press Alt Shift A .
^	Sends the Control key. For example, <code>Send, ^!a</code> would press Ctrl Alt A and <code>Send, ^{Home}</code> would send Ctrl Home . <code>^A</code> produces a different effect in some programs than <code>^a</code> . This is because <code>^A</code> presses Ctrl Shift A while <code>^a</code> presses Ctrl A . In doubt, use lowercase.
#	Sends the Win key (the key with the Windows logo) therefore <code>Send #e</code> would hold down the Win key and then press the letter "e".

The [gigantic table on the Send page](#) shows pretty much every special key built-in to AHK. For example: `{Enter}` and `{Space}`.

Caution: This table **does not** apply to [hotkeys](#). Meaning, you do not wrap `Ctrl` or `Enter` (or any other key) inside curly brackets when making a hotkey.

An example showing what shouldn't be done to a hotkey:

```
; When making a hotkey...
; WRONG
{LCtrl}::
Send, AutoHotkey
return

; CORRECT
LCtrl::
Send, AutoHotkey
return
```

A common issue lots of people have is, they assume that the curly brackets are put in the documentation pages just for fun. But in fact **they are needed**. It's how AHK knows that `{!}` means "exclamation point" and not "press the `Alt` `[REDACTED]`". So please remember to check the table on the [Send](#) page and make sure you have your brackets in the right places. For example:

```
Send, This text has been typed{!} ; Notice the !
between the curly brackets? That's because if it
wasn't, AHK would press the ALT key.
```

```
; Same as above, but with the ENTER key. AHK
would type out "Enter" if
; it wasn't wrapped in curly brackets.
```

```
Send, Multiple Enter lines have Enter been sent.  
; WRONG  
Send, Multiple{Enter}lines have{Enter}been sent.  
; CORRECT
```

Another common issue is that people think that **everything** needs to be wrapped in brackets with the Send command. That is FALSE. If it's not in the chart, it does not need brackets. You do **not** need to wrap common letters, numbers or even some symbols such as `.` (period) in curly brackets. Also, with the Send commands you are able to send more than one letter, number or symbol at a time. So no need for a bunch of Send commands with one letter each. For example:

```
Send, {a}           ; WRONG  
Send, {b}           ; WRONG  
Send, {c}           ; WRONG  
Send, {a}{b}{c}    ; WRONG  
Send, {abc}         ; WRONG  
Send, abc           ; CORRECT
```

To hold down or release a key, enclose the key name in curly brackets and then use the word UP or DOWN. For example:

```
; This is how you hold one key down and press  
another key (or keys).  
; If one method doesn't work in your program,  
please try the other.  
Send, ^s           ; Both of these  
send CTRL+S  
Send, {Ctrl down}s{Ctrl up} ; Both of these  
send CTRL+S  
Send, {Ctrl down}c{Ctrl up}  
Send, {b down}{b up}  
Send, {Tab down}{Tab up}  
Send, {Up down}    ; Press down the up-arrow key.  
Sleep, 1000        ; Keep it down for one second.
```

```
Send, {Up up} ; Release the up-arrow key.
```

But now you are wondering "*How can I make my really long Send commands readable?*". Easy. Use what is known as a continuation section. Simply specify an opening parenthesis on a new line, then your content, finally a closing parenthesis on its own line. For more information, read about [Continuation Sections](#).

```
Send,  
(  
Line 1  
Line 2  
Apples are a fruit.  
)
```

Note: There are several different forms of Send. Each has their own special features. If one form of Send does not work for your needs, try another type of Send. Simply replace the command name "Send" with one of the following: SendRaw, SendInput, SendPlay, SendEvent. For more information on what each one does, [read this](#).

a. Games

This is important: A lot of games, especially modern ones, have cheat prevention software. Things like GameGuard, Hackshield, PunkBuster and several others. Not only is bypassing these systems in violation of the games policies and could get you banned, they are complex to work around.

If a game has a cheat prevention system and your hotkeys, hotstrings and Send commands do not work, you are out of luck. However there are methods that can increase the chance of working in some games, but there is no magical "*make it work in my game now!!!*" button. So try **ALL** of these before giving up.

There are also known issues with DirectX. If you are having issues and you

know the game uses DirectX, try the stuff described on the [FAQ](#) page. More DirectX issues may occur when using [PixelSearch](#), [PixelGetColor](#) or [ImageSearch](#). Colors might turn out black (0x000000) no matter the color you try to get. You should also try running the game in windowed mode, if possible. That fixes some DirectX issues.

There is no single solution to make AutoHotkey work in all programs. If everything you try fails, it may not be possible to use AutoHotkey for your needs.

4 - Running Programs & Websites

To run a program such as *mspaint.exe*, *calc.exe*, *script.ahk* or even a folder, you can use the [Run](#) command. It can even be used to open URLs such as <https://autohotkey.com/> . If your computer is setup to run the type of program you want to run, it's very simple:

```
; Run a program. Note that most programs will
require a FULL file path:
Run, %A_ProgramFiles%\Some_Program\Program.exe

; Run a website:
Run, https://autohotkey.com
```

There are some other advanced features as well, such as command line parameters and CLSID. If you want to learn more about that stuff, visit the [Run](#) page.

Here are a few more samples:

```
; Several programs do not need a full path, such
as Windows-standard programs:
Run, notepad.exe
Run, mspaint.exe

; Run the "My Documents" folder using a built-in
variable:
Run, %A_MyDocuments%

; Run some websites:
Run, https://autohotkey.com
Run, http://www.google.com
```

For more in-depth information and examples, check out the [Run](#) page.

5 - Commands vs. Functions()

AutoHotkey has two main types of things used by the scripter to create code: Commands and functions.

A list of all commands and built-in functions can be found [here](#).

Commands

You can tell what a command is by looking at its syntax (the way it looks). Commands do not use parentheses around the parameters like functions do. So a command would look like this:

```
Command, Parameter1, Parameter2, Parameter3
```

When using commands, you cannot squish other commands onto the same line as a previous command (exception: [IfEqual](#)). You cannot put commands inside the parameters of other commands. For example:

```
MsgBox, Hello Run, notepad.exe ; Wrong
MsgBox, Hello, Run, notepad.exe ; Wrong

MsgBox, Hello ; Correct
Run, notepad.exe
```

Commands also differ from function in that they use "legacy syntax". This means that you **need** percent signs around a variable, such as `%Var%`, and that any text and numbers do not need to be in quotation marks, such as `This is some text`. Additionally, you cannot do math in the parameters, unlike functions.

You can do math in parameters if you force an expression with a single `%`, but that will not be covered.

Functions

As stated above, functions are different because they use parentheses. A typical function looks like:

```
Function(Parameter1, Parameter2, Parameter3)
```

Functions have a few main differences:

1. You can do math in them:

```
SubStr(37 * 12, 1, 2)  
SubStr(A_Hour - 12, 2)
```

2. Variables do not need to be wrapped in percent signs:

```
SubStr(A_Now, 7, 2)
```

3. Functions can go inside of functions:

```
SubStr(A_AhkPath, InStr(A_AhkPath,  
"AutoHotkey"))
```

4. Text needs to be wrapped in quotes:

```
SubStr("I'm scripting, awesome!", 16)
```

A function usually return a value differently than a command does. Commands need an *OutputVar* parameter, functions do not. The most common way assigning the value of a function to a variable is like so:

```
MyVar := SubStr("I'm scripting, awesome!", 16)
```

This isn't the only way, but the most common. You are using **MyVar** to

store the return value of the function that is to the right of the `:=` operator. See [Functions](#) for more details.

In short:

```
; These are commands:
MsgBox, This is some text.
StringReplace, Output, Input, AutoHotkey,
AutoHotKey, All
SendInput, This is awesome{!}{!}{!}

; These are functions:
SubStr("I'm scripting, awesome!", 16)
FileExist(VariableContainingPath)
Output := SubStr("I'm scripting, awesome!", 16)
```

a. Code blocks

[Code blocks](#) are lines of code surrounded by little curly brackets (`{` and `}`). They group a section of code together so that AutoHotkey knows it's one big family and that it needs to stay together. They are most often used with functions and control flow statements such as [If](#) and [Loop](#). Without them, only the first line in the block is called.

In the following code, both lines are run only if *MyVar* equals 5:

```
if (MyVar = 5)
{
    MsgBox, MyVar equals %MyVar%!
    ExitApp
}
```

In the following code, the message box is only shown if *MyVar* equals 5. The script will always exit, even if *MyVar* is **not** 5:

```
if (MyVar = 5)
    MsgBox, MyVar equals %MyVar%!!
    ExitApp
```

This is perfectly fine since the if-statement only had one line of code associated with it. It's exactly the same as above, but I outdented the second line so we know it's separated from the if-statement:

```
if (MyVar = 5)
    MsgBox, MyVar equals %MyVar%!!
MsgBox, We are now 'outside' of the if-
statement. We did not need curly brackets since
there was only one line below it.
```

6 - Variables

Variables are like little post-it notes that hold some information. They can be used to store text, numbers, data from functions and commands or even mathematical equations. Without them, programming and scripting would be much more tedious.

Variables can be assigned a few ways. We'll cover the most common forms. Please pay attention to the equal sign (=).

Legacy text assignment

```
MyVar = Text
```

This is the simplest form for a variable, a legacy assignment. Simply type in your text and done.

Legacy variable assignment

```
MyVar = %MyVar2%
```

Same as above, but you are assigning a value of a variable to another variable.

Legacy mixed assignment

```
MyVar = %MyVar2% some text %MyVar3%.
```

A combination of the two legacy assignments above.

Expression text assignment

```
MyVar := "Text"
```

This is an expression assignment, due to the `:` before the `=`. Any text

needs to be in quotes.

Expression variable assignment

```
MyVar := MyVar2
```

In expression mode, variables do not need percent signs.

Expression number assignment

```
MyVar := 6 + 8 / 3 * 2 - Sqrt(9)
```

Thanks to expressions, you can do math!

Expression mixed assignment

```
MyVar := "The value of 5 + " MyVar2 " is: " 5  
+ MyVar2
```

A combination of the three expression assignments above.

Equal signs (=) with a symbol in front of it such as := += -= .= etc. are called **assignment operators** and always require an expression.

a. When to use percents

One of the most common issues with AutoHotkey involving variables is when to use the percent signs (%). Hopefully this will clear some confusion.

When to use percent signs:

- When you are using commands (see above), except when the parameter is *OutputVar* or *InputVar*.
- When you are assigning a value to a variable using the legacy mode (an equal sign with no symbol in front of it).

When **not** to use percent signs:

- In parameters that are input or output variables. For example: `StringLen, OutputVar, InputVar`
- On the left side of an assignment: `Var = 123abc`
- On the left side of legacy (non-expression) if-statements: `if Var1 < %Var2%`
- Everywhere in expressions. For example:

```
if (Var1 != Var2)
  Var1 := Var2 + 100
```

b. Getting user input

Sometimes you want to have the user to choose the value of stuff. There are several ways of doing this, but the simplest way is [InputBox](#). Here is a simple example on how to ask the user a couple of questions and doing some stuff with what was entered:

```
InputBox, OutputVar, Question 1, What is your
first name?
if (OutputVar = "Bill")
  MsgBox, That's an awesome name`,,
%OutputVar%.

InputBox, OutputVar2, Question 2, Do you like
AutoHotkey?
if (OutputVar2 = "yes")
  MsgBox, Thank you for answering
%OutputVar2%`, %OutputVar%! We will become great
friends.
else
  MsgBox, %OutputVar%`, That makes me sad.
```

c. Other Examples?

```
MsgBox, 4,, Would you like to continue?  
IfMsgBox, No  
    return ; If No, stop the code from going  
further.  
MsgBox, You pressed YES. ; Otherwise, the user  
picked yes.
```

```
; Some examples showing when to use percents and  
when not:  
Var = Text ; Assign some text to a variable  
(legacy).  
Number := 6 ; Assign a number to a variable  
(expression).  
Var2 = %Var% ; Assign a variable to another  
(legacy).  
Var3 := Var ; Assign a variable to another  
(expression).  
Var4 .= Var ; Append a variable to the end of  
another (expression).  
Var5 += Number ; Add the value of a variable to  
another (expression).  
Var5 -= Number ; Subtract the value of a variable  
from another (expression).  
Var6 := SubStr(Var, 2, 2) ; Variable inside a  
function. This is always an expression.  
Var7 = %Var% Text ; Assigns a variable to another  
with some extra text (legacy).  
Var8 := Var " Text" ; Assigns a variable to  
another with some extra text (expression).  
MsgBox, %Var% ; Variable inside a command.  
StringSplit, Var, Var, x ; Variable inside a  
command that uses InputVar and OutputVar.  
if (Number = 6) ; Whenever an IF has parentheses,  
it'll be an expression. So no percent signs.  
if (Var != Number) ; Whenever an IF has  
parentheses, it'll be an expression. So no percent
```

signs.

`if Number = 6 ;` Without parentheses, the IF is legacy. However, only variables on the 'right side' need percent signs.

`if Var1 < %Var2%` ; Without parentheses, the IF is legacy. However, only variables on the 'right side' need percent signs.

7 - Objects

Objects are a way of organizing your data for more efficient usage. Sometimes objects are referred to as arrays, but it's important to note that all arrays are just objects. We call objects different things depending on what we are using them for, but all objects are the same.

An object is basically a collection of variables. The variable names are known as "Keys", and the contents of the variables are "Values".

When you hear people calling an object an *array* or *indexed array*, it usually means that all the keys are sequential numbers 1 and up. When you hear people calling an object an *associative array*, it means that the keys are either strings (text) or non-sequential numbers. Sometimes it's a mix of both, and sequential numbers too!

There are no restrictions to what a key or value can be, and they can even be other arrays! When the values are arrays too, this is referred to as a *nested array*, and these will be explained later.

There are a number of reasons you might want to use an object for something. Some examples:

- You want to have a numbered list of things, such as a grocery list (this would be referred to as an indexed array)
- You want to represent a grid, perhaps for a board game (this would be done with nested objects)
- You have a list of things where each thing has a name, such as the characteristics of a fruit (this would be referred to as an associative array)

a. Creating Objects

There are a few ways to create an object, and the most common ones are listed below:

Bracket syntax

```
MyObject := ["one", "two", "three", 17]
```

This will start you off with what is sometimes called an "indexed array". An indexed array is an object representing a list of items, numbered 1 and up. In this example, the value "one" is stored in object key 1 (aka index 1), and the value 17 is stored in object key 4 (aka index 4).

Brace syntax

```
Banana := {"Color": "Yellow", "Taste":  
"Delicious", "Price": 3}
```

This will let you start off by defining what is sometimes called an "associative array". An associative array is a collection of data where each item has a name. In this example, the value "Yellow" is stored in the object key "Color". Also, the value 3 is stored in the object key "Price".

Array function

```
MyObject := Array("one", "two", "three", 17)
```

This is equivalent to the bracket syntax, but wrapped in a function.

Object function

```
Banana := Object("Color", "Yellow", "Taste",  
"Delicious", "Price", 3)
```

This is equivalent to the brace syntax, but wrapped in a function.

It's important to remember that every one of these definitions all create the same thing (objects), just with different keys.

b. Using Objects

There are many ways to use objects, including retrieving values, setting values, adding more values, and more.

To set values:

Bracket notation

```
Banana["Pickled"] := True ; This banana has  
been pickled. Eww.
```

Setting values in an object is as simple as setting the value of a variable. All you have to do is put your bracket notation on the left side of an expression assignment operator `:=`.

Dot notation

```
Banana.Consistency := "Mushy"
```

The same as above but with the dot notation.

To retrieve values:

Bracket notation

```
Value := Banana["Color"]
```

This allows you to use an expression as the key to get the value from your object. In this case, I used the simple expression `"Color"`, which is (unsurprisingly) the key `Color`. You will get a message box with the word "Yellow", because that is what we set the key `Color` to in the [previous section](#).

Dot notation

```
Value := Banana.Color
```

This only lets you use literal strings for the keys. You cannot use variables in your keys with dot notation.

To add new keys and values:

Bracket notation

```
MyObject["NewerKey"] := 3.1415
```

To directly add a key and value, just set a key that doesn't exist yet.

Dot notation

```
MyObject.NewKey := "Shiny"
```

The same as above but with the dot notation.

InsertAt method

```
MyObject.InsertAt(Index, Value1, Value2,  
Value3...)
```

Index is any integer key. This will shift ALL higher integer keys up by the number of values which were inserted, even if there are gaps (for example, only keys 1 and 100 exist, and you insert a value at key 50, it will shift 100 up to 101).

Push method

```
MyObject.Push(Value1, Value2, Value3...)
```

This "appends" the values to the end of the array *MyObject*. In other words, it inserts the values at the highest integer key plus one.

To remove keys and values:

Blanking the value out

```
Banana.Consistency := ""
```

The simplest way to remove a value is to just blank it out. You can do this by setting it to `""`, also known as an *empty string*. This doesn't remove the key, but it will make the value appear identical to an unset value. It is possible to tell that the key still exists by using the [HasKey](#) method, and it will still come up in a [For](#) loop.

Delete method

```
RemovedValue := MyObject.Delete(AnyKey)
```

This and the next methods below remove the key *and* the value. The previous value of `MyObject[AnyKey]` will be stored in *RemovedValue*.

```
NumberOfRemovedKeys :=  
MyObject.Delete(FirstKey, LastKey)
```

Allows you to remove a range of numbered/integer or string keys between *FirstKey* and *LastKey*. The value it gives will be the number of keys that were removed, which is useful if you have a gap between your keys (e.g. you specify keys 1 through four, but key number 2 doesn't exist, this will set *NumberOfRemovedKeys* to 3 as only three keys were there to be removed).

Pop method

```
MyObject.Pop()
```

This removes the highest integer key, and returns the value. There are no keys higher than it to be affected.

RemoveAt method

```
RemovedValue := MyObject.RemoveAt(Index)
```

```
NumberOfRemovedKeys :=  
MyObject.RemoveAt(Index, Length)
```

This removes all keys from *Index* to *Index + Length - 1* (inclusive). If *Length* is omitted it defaults to 1. After removing the keys it takes all higher numbered/integer keys and moves them down to fill the gap, so that if there was a value at *Index + Length* it will now be at *Index*. This is similar to how the `InsertAt` method with multiple specified values works.

8 - Other Helpful Goodies

We have reached the end of our journey, my good friend. I hope you have learned something. But before we go, here are some other things that I think you should know. Enjoy!

a. The mysterious square brackets

Throughout the documentation, you will see these two symbols ([and]) surrounding code in the yellow syntax box at the top of almost all pages. Anything inside of these brackets are **OPTIONAL**. Meaning the stuff inside can be left out if you don't need them. When writing your code, it is very important to **NOT** type the square brackets in your code.

On the [ControlGetText](#) page you will see this:

```
ControlGetText, OutputVar [, Control, WinTitle,  
WinText, ExcludeTitle, ExcludeText]
```

So you could simply do this if you wanted:

```
ControlGetText, OutputVar
```

Or add in some more details:

```
ControlGetText, OutputVar, Control, WinTitle
```

What if you wanted to use *ExcludeTitle* but not fill in *WinText* or *WinTitle*? Simple!

```
ControlGetText, OutputVar, Control,,,  
ExcludeTitle
```

Please note that you cannot IGNORE parameters, but you can leave them blank. If you were to ignore `WinTitle`, `WinText`, it would look like this and cause issues:

```
ControlGetText, OutputVar, Control, ExcludeTitle
```

b. Finding your AHK version

Run this code to see your AHK version:

```
MsgBox, %A_AhkVersion%
```

Or look for "AutoHotkey Help File" or "AutoHotkey.chm" in the start menu or your installation directory.

c. Trial and Error

Trial and Error is a very common and effective way of learning. Instead of asking for help on every little thing, sometimes spending some time alone (sometimes hours or days) and trying to get something to work will help you learn faster.

If you try something and it gives you an error, study that error. Then try to fix your code. Then try running it again. If you still get an error, modify your code some more. Keep trying and failing until your code fails no more. You will learn a lot this way by reading the documentation, reading errors and learning what works and what doesn't. Try, fail, try, fail, try, try, try, fail, fail, **succeed!**

This is how a lot of "pros" have learned. But don't be afraid to ask for help, we don't bite (hard). Learning takes time, the "pros" you encounter did not learn to be masters in just a few hours or days.

"If at first you don't succeed, try, try, try again." - Hickson, William E.

d. Indentation

This stuff (indentation) is very important! Your code will run perfectly fine without it, but it will be a major headache for you and other to read your code. Small code (25 lines or less) will probably be fine to read without indentation, but it'll soon get sloppy. It's best you learn to indent ASAP. Indentation has no set style, but it's best to keep everything consistent.

"What is indentation?" you ask? It's simply spacing to break up your code so you can see what belongs to what. People usually use 3 or 4 spaces or 1 tab per "level".

Not indented:

```
if (car = "old")
{
MsgBox, The car is really old.
if (wheels = "flat")
{
MsgBox, This car is not safe to drive.
return
}
else
{
MsgBox, Be careful! This old car will be
dangerous to drive.
}
}
else
{
MsgBox, My`, what a shiny new vehicle you have
there.
}
```

Indented:

```
if (car = "old")
{
    MsgBox, The car is really old.
```

```
if (wheels = "flat")
{
    MsgBox, This car is not safe to drive.
    return
}
else
{
    MsgBox, Be careful! This old car will be
dangerous to drive.
}
}
else
{
    MsgBox, My`, what a shiny new vehicle you
have there.
}
```

See Wikipedia's [Indentation style](#) page for various styles and examples. Choose what you like or learn to indent how you think it's easiest to read.

e. Asking for Help

Before you ask, try doing some research yourself or try to code it yourself. If that did not yield results that satisfy you, read below.

- Don't be afraid to ask for help, even the smartest people ask others for help.
- Don't be afraid to show what you tried, even if you think it's silly.
- Post anything you have tried.
- Pretend *everyone but you* is a doorknob and knows nothing. Give as much information as you can to educate us doorknobs at what you are trying to do. Help us help you.
- Be patient.
- Be polite.
- Be open.
- Be kind.
- Enjoy!

If you don't get an answer right away, wait at least 1 day (24 hours) before

asking for more help. We love to help, but we also do this for free on our own time. We might be at work, sleeping, gaming, with family or just too busy to help.

And while you wait for help, you can try learning and doing it yourself. It's a good feeling, making something yourself without help.

f. Other links

[Frequently Asked Questions \(FAQ\)](#)

Changes & New Features

Changes and new features introduced by the current branch of AutoHotkey development (a.k.a. AutoHotkey_L) are listed below.

For older changes, see [Archived Changes](#).

1.1.28.02 - April 7, 2018

Fixed `Control ChooseString` and `ControlGet FindString` ignoring the first two items.

Fixed `Control ChooseString` to send `WM_COMMAND` even if the control's ID is 0.

Fixed `WinActivate` to restore the active window, as originally intended for [v1.1.20.00].

1.1.28.01 - March 31, 2018

Fixed Thread treating omitted parameters as 0.

Fixed FileAppend to stderr (**).

Fixed `break label` being able to jump to an unrelated loop.

Reverted hotstring reset behaviour to pre-v1.1.28.00.

Added `Hotstring("Reset")` for manually resetting the hotstring recognizer.

1.1.28.00 - February 11, 2018

Changes:

- Changed `__Delete` to catch and report exceptions when called during object cleanup. It previously had the (erroneous) effect of "postponing" the exception until the next function call or the end of the try-block/thread.
- Changed hotstring recognizer to reset when focus changes instead of just when the active window changes.
- Changed `WinMenuSelectItem` to treat menu `0&` as the window's system menu.

New features:

- Added support for all built-in variables in the path passed to `#Include`.
- Added [A_TimeIdleKeyboard](#) and [A_TimeIdleMouse](#).
- Added [A_ListLines](#).
- Added [A_ComSpec](#) (alias of `ComSpec`).
- Added [A_LoopFilePath](#) (alias of `A_LoopFileFullPath`, which is a misnomer).
- Added hotstring [X option](#) to execute a same-line action instead of auto-replace.
- Added [Hotstring\(\)](#).
- Added [function hotstrings](#).
- Added `MaxParts` parameter to [StrSplit](#).
- Improved [#MenuMaskKey](#) to allow specifying VK and SC, or `vk00sc000`.

Bug fixes:

- Fixed `#MenuMaskKey` treating some valid keys (such as `Del`) as invalid.
- Optimised detection of `AltGr` on Unicode builds. This fixes a delay which occurred at startup (since v1.1.27.00) or the first time `Send` is called for each target keyboard layout (prior to v1.1.27.00).
- Fixed misleading vicinity lines for `x up::y` remap error.
- Fixed `Menu Tray, Icon, HICON:%hicon%`, which now uses *hicon* without resizing it.

1.1.27.07 - January 21, 2018

Fixed default size of Gui with +Parent to not be restricted by parent [broken by v1.1.27.05].

Fixed controls not redrawing if a separate Tab control is shown/hidden immediately after the control was invalidated (such as when showing/hiding multiple controls at once).

1.1.27.06 - January 16, 2018

Fixed hotstrings/Input causing stuck dead keys (broken by v1.1.27.05).

1.1.27.05 - January 16, 2018

Fixed visible Input end keys causing any pending dead key to double up.

Fixed hotstrings/Input affecting how Tab/Esc act in a dead key sequence.

Fixed A_IconFile returning an incorrect path if a DLL was used.

Fixed Gui with +Parent to center within the parent GUI vs. the screen.

Fixed Input/hotstring detection of dead key sequences in Universal Windows Platform (UWP) apps.

Fixed `< & v` and `> & v` being seen as duplicates, and similar cases. This affects custom combinations where the prefix key is also a modifier symbol. This also fixes hotkeys which consist only of modifier symbols and a single trailing space, such as `+ : :`, to be treated as errors rather than ignoring the space (hotkeys do not permit trailing space).

1.1.27.04 - January 10, 2018

Fixed #Warn ClassOverwrite giving erroneous warnings.

1.1.27.03 - January 6, 2018

Improved `Menu x, NoStandard` and `Menu x, DeleteAll` to work without attempting to destroy the underlying Win32 menu. This allows them to work when `x` is a menu bar or sub-menu of one.

Reworked the handling of `vkXXscYYY` :

- Fixed `GetKeyVK` and `GetKeyName` treating `vkXXscYYY` as `vk00scYYY`.
- `Send` is now more strict with `{vk...}` and invalid suffixes, consistent with similar changes made by [v1.1.27.00] (but `sc` is still supported).
- Reduced code size.

Fixed mishandling of numeric keys outside the 32-bit range (but inside the 64-bit range) in some corner cases on 64-bit builds. This only affects classes which use such numbers as names (not recommended for multiple reasons) and array access via `IDispatch` with such numbers as keys/member names.

1.1.27.02 - January 1, 2018

Fixed loading of bmp files as icons at original size.

Fixed compound assignments such as `Test.Prop[1] += 1` (broken by v1.1.27.01).

1.1.27.01 - December 31, 2017

Fixed program crashes caused by `++X` or `--X` in scripts which lack `#NoEnv` (broken by v1.1.27.00).

Fixed `#Warn ClassOverwrite` giving an erroneous warning for `++MyClass.X`.

Fixed remapping to allow custom combinations such as `a & b::c`.

Fixed `Send/hotstrings/Input` to adapt to the keyboard layout of the focused control instead of just the active window. In particular, this affects UWP apps such as Microsoft Edge.

Fixed hook hotkeys to suppress the Start menu activation which occurs when an isolated `Ctrl/Shift` up is received. `Ctrl` up may be generated by the system when switching from a keyboard layout with `AltGr` to one without (such as from UK to US), if `Ctrl` is not held down at the time. This fixes hotkeys such as `$#z::WinActivate x` when the change in focus causes those conditions.

Fixed `Input` not collecting characters when both `Shift` keys are down.

Fixed `Input` to use `"sc"` and `"vk"` for end keys in `ErrorLevel` rather than `"Sc"` and `"Vk"` (caused by v1.1.20).

Fixed `GetKeyName/VK/SC("vkXXscYYY")` where `YYY` begins with A-F (broken by v1.1.26).

1.1.27.00 - December 25, 2017

Changes:

Replaced AU3_Spy.exe with WindowSpy.ahk.

- AU3_Spy.exe is still launched if WindowSpy.ahk is not found.
- It now follows the focused control by default, and has a checkbox for both window and control to follow the mouse.
- It no longer takes over a global hotkey (Win+A). Instead, hold Ctrl or Shift to suspend updates (release them after focusing Window Spy).
- It is now possible to Alt-Tab to Window Spy on Windows 10 without the contents of the GUI changing.

Changed [a-z to mean vk41-vk5A](#) when absent from the keyboard layout, except with Raw mode or when sending single unmodified characters. This allows hotkeys and sent keyboard shortcuts to work more intuitively on certain non-English keyboard layouts.

Changed Send on ANSI versions to use [SendInput\(\)](#) in place of Alt+nnnnn for [special characters](#).

Changed the rules for [masking Alt/Win](#) after pressing a hook hotkey:

- Explicitly sent Alt/Win up may be masked. This fixes remappings such as `AppsKey::RWin`, but hotkeys which are intended to activate the Start Menu may require new workarounds.
- If Alt/Win is logically but not physically down, only hotkeys which require Alt/Win (such as `#a::`, not `*a::`) cause masking. This is to allow a remapping or wildcard hotkey to send the key-up without it being masked.
- Unsuppressed hotkeys such as `~#a::` no longer cause masking, because the unsuppressed keydown/keyup is sufficient to prevent a menu. However, mouse hotkeys like `~*MButton::` no longer suppress the Start Menu if combined with the Win key. It can be suppressed manually with `Send {Blind}{vk07}` or similar.
- The keyboard hook now tracks events in relation to Alt/Win, so that the mask key does not need to be sent if Alt/Win was already masked by some

other event (physical or sent).

The hotkeys `~LWin::` and `~RWin::` no longer suppress the Start Menu. See [#MenuMaskKey](#) for details and a workaround.

Added proper validation for vk or sc key names, so names such as "sc01notvalid" are no longer recognized as keys.

Scripts containing hotkeys of the form `VKnnSCnnn::` will need to be corrected by removing `SCnnn`, which was previously ignored.

Help file: Replaced the standard HTML Help Viewer sidebar with the new HTML5 sidebar developed by Ragnar-F.

New features:

- Added [Min/Max](#) built-in functions. [PR #84 from Ragnar-F]
- Added [A_Args](#) as an alternative to the numbered variables.
- Added [force-local mode](#) for functions.
- Added [#Warn ClassOverwrite](#).
- Added [{Text} mode](#) for Send and [T option](#) for hotstrings. These are like the Raw mode, but more reliable.
- Added [{key DownR}](#) and changed remapping to use it, to fix issues with `AppsKey::RWin` and similar.

Bug fixes:

- Fixed icon-loading to not default to ID 0 when the index is invalid.
- Fixed VK ↔ SC mapping of PrintScreen. SC → VK was already correct on Vista and later.
- Fixed Hotkey control returning scXXX instead of names in some cases.
- Fixed ListVars crashing if a ByRef parameter refers to a variable containing an object. [PR #86 from HotKeyIt]
- Fixed some (very unlikely) memory leaks.
- Fixed menu handles not being freed if only the standard items are used.
- Fixed bold font not being applied to default menu item if it has a submenu and a Win32 menu handle.

- Fixed Send to use the correct modifier state with Unicode chars.
- Fixed `ControlSend {u n}`, where u is Unicode, to send n times, not just 1.
- Fixed inconsistent behavior of AltGr between OS versions. Specifically, `RAlt::` once again causes the system to "release" LCtrl on Windows 10, as it did prior to v1.1.26.01 (but unlike those old versions, it prevents the RAlt-up from reaching the active window). This change should not affect layouts without AltGr.
- Fixed `Menu Tray, Icon, SB_SetIcon` and `LoadPicture` with non-zero Icon option to allow bitmaps (but convert if needed).
- Fixed menu items to not disappear when a cursor is set as a menu item's icon.
- Improved launching of Window Spy and the help file:
 - If AutoHotkey is installed but the current executable is in a different directory (i.e. portable), the installed file is no longer preferred as it may be the wrong version. It may still be used as a fallback.
 - On failure to launch the file, show the error message inside the dialog, not in its titlebar.
- Fixed one-line hotkeys with expressions beginning in `SC` or `VK`.
- Fixed `> / <` hotkey modifiers incorrectly allowing both keys to be pressed. For example, `<^A` could erroneously be triggered by LCtrl+RCtrl+A, thereby preventing the `<>^A` hotkey from working.
- Fixed auto-replace hotstrings inserting the literal text "{Raw}" in some cases (specifically, when the replacement contains `{Raw}` and the O, R and * options were not used).
- Fixed some hotkeys not using the hook when eclipsed by a wildcard hotkey, depending on the order of definition.
- Fixed key-down hotkeys to revert to "reg" if they were only "k-hook" because of a corresponding key-up hotkey which has since been disabled. (Prior to v1.1.07.03 this already happened, but the key-down hotkey was never set to "k-hook" in the first place if defined after the key-up.)
- Fixed hook hotkeys to ignore modifier changes made by `Send !^+# {key}` or when sending Unicode characters. This restores Send to how it was prior to v1.1.06 when at SendLevel 0, but keeps the v1.1.06+ behavior at SendLevel >= 1, allowing Send to trigger the right hotkeys.
- Improved the odds that Send from a "reg" hotkey such as `^m::Send x` will restore the modifier state if Send is being called for the very first time. This makes it less likely to produce a different (and unexpected) result the

first time it is held down, such as xmmm instead of xxxx.

1.1.26.01 - July 16, 2017

Fixed RegDelete (with no args) failing when A_LoopRegSubKey is blank, such as when enumerating keys directly under the root key.

Fixed `RAlt/LAlt::` sometimes failing to prevent menu activation after the user alt-tabs away from a window and reactivates it.

Fixed SC → VK translation for multimedia keys, PrintScreen, CtrlBreak, Pause, LWin and RWin, with the exception that multimedia keys, PrintScreen and CtrlBreak are still translated incorrectly on Windows 2000 and XP.

Fixed VK → SC translation for multimedia keys, CtrlBreak and Pause.

Fixed 'Wait commands such as RunWait to log extra lines for ListLines only when necessary to show that it is still waiting; that is, after resuming from an interruption which logged other lines.

1.1.26.00 - June 10, 2017

Changed the format ListVars uses to display variables containing objects. The object's class name is now shown.

Added "class" and "clsid" modes to ComObjType().

Revised class names shown by the debugger and significantly reduced code size in the process.

Revised FileSetAttrib, FileSetTime and FileDelete to reduce code size. There should not be any changes in behaviour.

Made other minor optimizations to code size.

Fixed GetKeySC() with the key names Esc, Escape, ScrollLock and PrintScreen.

Fixed hotstring word detection of words containing nonspacing marks, such as Arabic diacritics.

1.1.25.02 - May 13, 2017

Fixed GUI option strings being permanently truncated if an error is raised.

Fixed properties and class sub-variables to allow non-ASCII characters.

Fixed `new` operator to allow `__New` to throw or exit.

1.1.25.01 - March 5, 2017

Fixed Send to allow other scripts to act on modifier state changes which immediately precede a special character, such as the Shift release in `Send Mört`.

1.1.25.00 - March 4, 2017

Added [Hotkey, If, % FunctionObject](#).

Fixed website address in Help menu.

1.1.24.05 - February 3, 2017

Fixed WinSet AlwaysOnTop/Trans/TransColor to work on windows with 0 ExStyle.

1.1.24.04 - December 17, 2016

Fixed `File.ReadNum()` repeating old data if a prior `File.Read()` had stopped reading at the end of the buffer.

Improved buffer utilisation when `File.Read()` reaches the end of the buffered data.

Fixed `GuiControl +Password` to use the default bullet character on XP and later.

Fixed `GuiControl +/-Password` to redraw the control immediately.

1.1.24.03 - November 19, 2016

Fixed COM exception messages to not end in ``r`.

Fixed `{U+x}` to support supplementary characters ($x > 0xFFFF$).

Fixed class properties which lack `get{}` to return an empty value instead of the internal Property object when the class has a base which does not define a value for the property. Properties defined in baseless classes already behaved correctly.

Fixed the background color of controls on a Tab3 control when the system has visual styles disabled (i.e. Windows Classic).

Fixed handling of thread-exit in functions which are called directly by an event (such as `OnExit` functions). This bug had no known effect except in the v2-alpha branch.

Fixed the debugger to detect disconnection immediately (i.e. when a client terminates without calling `stop/detach`).

Fixed the debugger to treat `property_get -m 0` as "unlimited", as per the DBGp spec.

Fixed the debugger to expect base64 for `property_set -t integer/float`, as per the DBGp spec.

1.1.24.02 - October 13, 2016

Added a [Run with UI Access](#) option to the installer.

Added documentation for Tab3 [autosizing](#).

Fixed several Tab3 bugs:

- Autosizing was not being triggered by the first call to [Gui Show](#) if the `AutoSize` or `w` and `h` options were used.
- If a Tab3 control is empty when autosizing occurs, it now retains the default size instead of becoming unusable.
- Autosizing now occurs immediately before creating another tab control, as though [Gui Tab](#) was called.
- Autosizing failed if the tab control had negative screen coords.
- Hiding a Tab3 control with `GuiControl` now also hides its dialog/background.

Fixed `OnMessage` to pass `lParam` as unsigned on 32-bit.

Fixed `Gui Name :` to allow spaces before the colon.

Fixed identifying a `Gui` or `GuiControl` by `HWND` with a negative value.

Fixed `ComObject` to suppress any errors raised by [disconnection of events](#) after the script has released the object.

Fixed `Gui Show` causing the window to shrink if it had a multi-line menu bar.

Fixed `LV_InsertCol(n, width)` not causing `ListView` scrollbar to update on some OS versions.

Optimized code size of `FileRecycleEmpty`.

1.1.24.01 - August 2, 2016

Changed `Hotkey, If, Expression` to raise an error if the unused third parameter is not blank.

Fixed `&&`, `||` and ternary to release any object used as the condition, as in `if (a.subObject && b)`.

Fixed `Gui/GuiControl` failing to "check" radio buttons within a `Tab3` control.

Fixed `FileCreateShortcut` to allow relative paths for the `LinkFile` parameter on Windows 10.

Fixed themed `Tab3` control to not override the custom text color of its controls.

Fixed debugger mishandling continuation commands in some specific cases.

1.1.24.00 - May 22, 2016

Breaking changes:

Passing SetTimer's *Label* parameter an empty variable or an expression which results in an empty value is now considered an error. The parameter must be either given a non-empty value or **completely omitted**.

Run-once timers are automatically deleted after running if they are associated with an object created by the script. This allows the object to be freed if the script is no longer referencing it, but it also means the timer's *Period* and *Priority* are not retained.

SetTimer, Hotkey, GuiControl and Menu now check the minimum parameter count of functions given by reference (not just by name, as in previous versions). If the function requires too many parameters, an error is raised or ErrorLevel is set (as appropriate).

Backward-compatible changes:

If SetTimer's *Label* parameter is **omitted** and *A_ThisLabel* is empty, the current timer (if any) is used.

New features:

Added **Tab3 control type**, solving a number of issues with Tab controls.

Bug-fixes:

Fixed GuiControl to update controls when adding/deleting tabs. Specifically:

- Selecting a new tab with `||` now works correctly.
- Deleting all tabs now hides the controls of the former tab.

Fixed `+Disabled` / `+Hidden` losing effect on controls in a tab.

Fixed disabled tab controls to ignore Ctrl+Tab.

Fixed `Gui Tab` without parameters to start a new radio group if applicable.
`Gui Tab` with parameters already had this effect.

1.1.23.07 - May 20, 2016

Fixed Menu x, Insert, y, z, % object to use the object, not ignore it.

Fixed Menu x, Add, :item text, % object to use the object, not look for a submenu.

1.1.23.06 - May 15, 2016

Fixed `break label` crashing the program if nested directly inside its target loop.

1.1.23.05 - March 27, 2016

Fixed InputBox, MsgBox, FileSelectFile and FileSelectFolder sending an unwanted Alt-up if Alt is down. This was due to an error with the Ctrl/Shift workaround added in v1.1.22.01.

Improved the Ctrl/Shift workaround to avoid unnecessarily "masking" the Win key if Ctrl, Shift or Alt is also down.

1.1.23.04 - March 26, 2016

Fixed LV_Modify to support omitting Options, as in `LV_Modify(r,, col1)` .

1.1.23.03 - March 12, 2016

Fixed __Delete meta-functions erroneously suppressing or prematurely re-throwing exceptions, when they are called during exception propagation.

Fixed load-time detection of function calls where a required parameter is blank, as in `fn(x, , y)`.

1.1.23.02 - March 12, 2016

Fixed RegDelete deleting the entire key instead of the default value.

Code maintenance; minor code size improvement.

1.1.23.01 - January 24, 2016

Fixed a theoretical issue with loading scripts from weird/very long paths.

Ahk2Exe: Fixed Unicode in compiled scripts (broken by v1.1.23.00 release).

1.1.23.00 - January 16, 2016

Added [Menu Insert](#) sub-command.

Added [MenuGetHandle\(\)](#).

Added [MenuGetName\(\)](#).

Added menu item options: Radio, Right, Break and BarBreak.

Improved the Menu command to allow identifying items by position: **1&**

Added [LoadPicture\(\)](#).

Added [hicon/hbitmap: syntax](#) for passing handles to commands which normally load an image from file.

Added built-in variables: [A_CoordModeToolTip/Pixel/Mouse/Caret/Menu](#), [A_DefaultGui](#), [A_DefaultListView](#), [A_DefaultTreeView](#), [A_KeyDelayPlay](#), [A_KeyDuration/Play](#), [A_MouseDelayPlay](#), [A_SendLevel](#), [A_SendMode](#) and [A_StoreCapsLockMode](#).

Added **Ix** Hotkey option to set the hotkey variant's input level to x.

Improved Picture control to support BackgroundTrans with icons.

Improved Picture control to reduce flicker when loading large images.

Small optimizations to the menu code and built-in var lookups.

Fixed conversion of menu items to/from separators.

Fixed [A_ThisMenuItemPos](#) to support duplicate item names.

Fixed sub-menus sometimes not being recreated after a menu is deleted.

Fixed [AutoHotkeyXXX.exe](#) to launch [AutoHotkey.chm](#), not [AutoHotkeyXXX.chm](#).

1.1.22.09 - November 11, 2015

Fixed some issues with SetTimer Delete.

- KeyHistory's timer count erroneously included deleted timers.
- Calling KeyHistory within a thread started by a timer crashed the script if the timer had been deleted but not turned off.
- Deleting the most recently created timer prevented subsequently created timers from working unless ALL timers were deleted.

1.1.22.08 - November 11, 2015

Fixed For-loop to pass control correctly when `_NewEnum/Next` throws an exception.

Fixed Finally to suspend the pending exception until its body has been evaluated, not just until the first built-in function call.

Fixed load-time detection of invalid jumps from Finally blocks (broken by v1.1.20).

1.1.22.07 - September 27, 2015

Fixed Gui control `w-1` and `h-1` options failing when DPI is 150+% and the Gui has not applied `-DPIScale` .

Fixed `~key up` hotkeys to not perform an automatic `Send {key down}` . This was occurring only if the hotkey was turned on after the key was pressed down.

1.1.22.06 - September 13, 2015

Fixed Input and hotstrings to catch Unicode/non-key character events.

Fixed auto-sizing of Edit controls to include the last line if blank.

Fixed handling of out-of-memory in ComObj functions.

Fixed ComObjArray to ignore excess parameters rather than crashing (only applies to dynamic calls).

Fixed GuiControl to append--not prepend--tabs, as documented.

Fixed XP64 support.

1.1.22.05 - September 10, 2015

Fixed icons which have non-numeric IDs loading with sub-optimal quality.

Fixed Gui Destroy not releasing function objects which were set by the +g option.

Fixed Gui Show to avoid attempting to force-activate a child GUI.

Fixed failure to set Caps/Num/ScrollLock state while the key is down.

Fixed Gui Color causing incorrect text color (usually black).

Changed Gui Show to improve the odds of GuiSize executing immediately.

For developers:

- Modified project config to support more VC++ versions.
- Fixed various build warnings on VC++ 2015.
- 64-bit binaries are now built with VC++ 2015. 32-bit binaries still use VC++ 2010.

1.1.22.04 - August 19, 2015

Fixed ObjRawSet() to return nothing.

Added Windows 10 supportedOS tag to the exe manifest (avoids some issues with the Program Compatibility Assistant and similar).

Added detection of syntax errors after ")" in a function declaration.

1.1.22.03 - July 12, 2015

Fixed A_EndChar returning a truncated value for Unicode end chars.

Small implementation changes:

- Changed A_Language to use GetSystemDefaultUILanguage().
- Refactored Window Spy/help file launching from tray menu to improve code re-use.
- Optimized Gui/Menu/Hotkey/Hotstring/OnClipboard message handling (minor).

1.1.22.02 - May 27, 2015

Fixed TreeView to not raise `*` events for unknown notifications.

Fixed crashing/bad behaviour when a timer deletes itself.

Fixed RWin-up being `masked` in some rare cases where LWin-up wouldn't have been.

1.1.22.01 - May 24, 2015

Fixed Text/Edit/Button control sizing to compensate for character overhang.

Fixed registry commands to allow `:` in the SubKey when combined with RootKey.

Fixed hotkey prioritization to take modifiers into consideration.

Refactored else/try/catch/finally handling to support `hotkey::try cmd`.

Added a workaround for the script's dialogs acting as though Ctrl or Shift is pressed after they are blocked by the keyboard hook.

1.1.22.00 - May 1, 2015

Added SetErrorMode(SEM_FAILCRITICALERRORS) on program startup to suppress system-level error messages such as "There is no disk in the drive". Calling Drive or DriveGet no longer affects the process' error mode.

Changed MonthCal controls to have tab-stop by default on Vista or later.

Improved ComObjConnect to use IProvideClassInfo when available.

Fixed some issues with method/property definitions following an end brace on the same line.

Fixed Text/Link control auto-sizing to compensate for the +Border (WS_BORDER) style.

Fixed `Break N` when Loop is used directly below If/Else/Try/Catch.

1.1.21.03 - April 12, 2015

Fixed detection of naming conflicts between properties and methods.

1.1.21.02 - April 4, 2015

Fixed `OnMessage(msg, fobj, 0)` to do nothing if *fobj* wasn't previously registered.

1.1.21.01 - April 3, 2015

Fixed StrReplace() to allow ReplaceText to be omitted.

Fixed class variables to allow non-ASCII names.

1.1.21.00 - March 28, 2015

Added `Loop`, `Reg`, `RootKey[\Key, Mode]`.

Added `Loop`, `Files`, `FilePattern [, Mode]`.

Changed `InputVar` parameters to allow `% expression` (except with `If` commands).

Revised `Object` methods:

- Added `Object.InsertAt()`, `Object.Push()` and `ObjRawSet()`.
- Added `Object.Delete()`, `Object.RemoveAt()` and `Object.Pop()`.
- Added `Object.Length()`.

Added `Ord()` and updated `Chr()` to support supplementary chars (>0xFFFF).

Added `StrReplace()`.

Removed the obsolete and undocumented 5-parameter mode of `RegRead`, which was exclusively for `AutoIt v2` compatibility (the extra parameter was unused).

Changed `RegRead`, `RegWrite` and `RegDelete` so that the `RootKey` and `SubKey` parameters can optionally be combined.

1.1.20.03 - March 21, 2015

Fixed MouseGetPos (OutputVarControl), GuiContextMenu and GuiDropFiles (A_GuiControl) to not ignore disabled controls [broken by v1.1.20.00].

1.1.20.02 - March 11, 2015

Fixed add-first mode of OnMessage.

Fixed A_OSVersion for unrecognized OSes on x64 builds.

Fixed ExitApp to unpause the script before releasing objects in case a __delete meta-function is called.

Trivial optimizations and code maintenance.

1.1.20.01 - March 10, 2015

Reverted the changes made in v1.1.20.00 to saving and restoring of ErrorLevel on thread interrupt, due to unintended consequences.

- ErrorLevel is not reset or cleared when a new thread starts. Instead, it retains the value it had in the interrupted thread, as in v1.1.19 and older.
- If ErrorLevel contains an object and the thread is interrupted, the object is replaced with an empty string when the thread resumes, as in v1.1.19 and older.
- If ErrorLevel contains a string longer than 127 characters, it is truncated when the thread resumes, as in v1.1.19 and older.

Re-fixed timers sometimes causing ErrorLevel to be formatted as hex.

Fixed class methods to retain a counted reference to the class.

1.1.20.00 - March 8, 2015

New features:

Enhanced Hotkey, Menu, SetTimer, Gui events and Gui control events to accept [a function instead of a label](#), if no label is found.

Enhanced Hotkey, Menu, SetTimer and Gui control events to accept a [function object](#). Added a Delete sub-command for SetTimer to allow the object to be released.

Enhanced [OnMessage\(\)](#) to allow any number of functions or function objects to monitor a single message.

Added [OnExit\(\)](#) and [OnClipboardChange\(\)](#), which accept a function name or object.

Added capability to [point hotkey labels at a function definition](#).

Added [U/L/T modifiers](#) to perform case conversion with Format().

Added the [E option](#) to the Input command, for handling end keys by character instead of keycode.

Added `**` (stderr) support to FileAppend.

Added [ObjBindMethod\(obj, method, args*\)](#) and [Func.Bind\(args*\)](#).

Changes:

Changed the default behavior when the main script file is not found:

- If no script was specified and the default script files are not found, show the new [Welcome](#) page in the help file.
- If a script was specified or the help file is missing (or is named differently to the executable), just show an error message.

Changed WinActivate to restore the window if already active but minimized.

Changed WinActivate to look for a visible window to activate if DetectHiddenWindows is off and the active window is hidden, instead of doing nothing.

Changed the method used by A_CaretX/A_CaretY to retrieve the caret position.

- Returns blank in more cases where the real caret position can't be determined, instead of returning the top-left of the active window.
- Less likely to cause side-effects.
- Works in console windows.

Changed A_OSVersion to use [RtlGetVersion\(\)](#), so that it can detect Windows 10 and hopefully future versions.

Changed A_OSVersion to return a version number of the form "major.minor.build" if it doesn't have a name for the OS.

Changed objects to support `x[, y]`, `x.y[, z]` and `x[](y)`.

- User-defined objects can utilize this by specifying default values for parameters of properties and meta-functions. For `__Call`, the first parameter is omitted, as in `x.__Call(, y)`.
- COM objects invoke `DISPID_VALUE` if the member name is omitted. For example, `x[]` retrieves x's default property and `fn[]()` can be used to call JScript functions.

Several under-the-hood changes to fix bugs, reduce code size or improve performance, including:

- Changes to preparsing of `#if` and static var initializers.
- Changes to preparsing of `{ blocks }`.

Bug-fixes:

Fixed VK to keyname conversions for keys 'A' to 'Z' to respect layout.

- Applies to: `GetKeyName`, Hotkey control, `A_PriorKey`, `KeyHistory` and `Input EndKey ErrorLevel` (except where Shift key is required).
- All other keys, including those that produce non-ASCII letters, were already translated according to AutoHotkey's current keyboard layout.

Fixed FileAppend to * (stdout) to respect the encoding parameter instead of always outputting ANSI text.

Fixed auto-sizing of GUIs with only one scrollbar.

Fixed `Exception(m, n)` crashing when n is too far out of bounds.

Fixed GuiContextMenu to set `A_GuiEvent` correctly on x64.

Fixed FileGetSize sometimes giving an indeterminate result if the file doesn't exist or can't be opened.

Fixed thread interrupts to save and restore `ErrorLevel` more fully [reverted in v1.1.20.01 due to unintended consequences].

Fixed ControlClick Pos mode to ignore disabled controls.

Fixed odd behaviour when Gui `+MaxSize` is smaller than `+MinSize`.

Fixed GuiControl/Get requiring a Gui name when given a control `HWND`.

Fixed meta-functions interfering with the line number reported by `Exception()`.

1.1.19.03 - February 11, 2015

Improved remapping to allow `SCXXX::Y` when the current keyboard layout does not map `SCXXX` to a VK code. However, `Y` must still exist on the current keyboard layout.

Fixed `break n` to work correctly when `until` is present, instead of terminating the thread.

Fixed `ControlGetFocus` disrupting the user's ability to double-click (thanks HotKeyIt).

Fixed `ListView` to not call the control's `g-label` for unsupported (and previously unknown) notifications such as `LVN_GETEMPTYMARKUP`.

Fixed `#Include *i <X>` to exit the program as intended if `X` has a syntax error.

Fixed `for var in expression` sometimes crashing the program when `expression` calls a script function.

1.1.19.02 - January 25, 2015

Removed the 16-color icons which were used on Win9x/Win2k.

Removed the separate tray icon resources.

Improved selection of icon size when loading icons from DLL/EXE files.

- In short, prefer to downscale rather than upscale. This is especially helpful on systems with 125% or 150% DPI, where the system icon sizes are 20/40 or 24/48 instead of 16/32. If all of these sizes are present in the icon resource, this change makes no difference.

Fixed some issues with hotkey validation at load time.

- Hotkeys which are never valid, like foo::, are now always treated as errors instead of giving a misleading warning about keyboard layout.
- Hotkeys like ^!ä:: which are inactive because of the current keyboard layout are now allowed to have a same-line action, and the label is registered correctly (e.g. as "^!ä" and not as "^!ä:"). If the /iLib command line switch is used, the warning is suppressed.
- Remappings which are inactive because of the current keyboard layout now show two warnings instead of an unrecoverable error.
- If a Wheel hotkey is used as a prefix key, there is only one error message instead of two, and it respects /ErrorStdOut.

Fixed /iLib switch to write the file even if there's a syntax error.

Fixed return/break/continue/goto in try..finally.

1.1.19.01 - January 5, 2015

Fixed Hotkey command crashing the program when trying to create new variants of existing hotkeys (broken by v1.1.19.00).

Made some minor optimizations to the Hotkey command and A_TimeIdle, relating to removal of Win9x support.

1.1.19.00 - January 4, 2015

Added a name for `Func.Call()`. `Func.()` still works but is deprecated.

Fixed some issues with `X.Y` and `X.Y(Z)` in VBScript/JScript/C#:

- If `X.__Call` contained a function *name*, it was being returned instead of called.
- When `X.Y(Z)` returned a value, *Z* was ignored. Now it acts like `X.Y[Z]` when `X.Y` is not a function.

Fixed the Hotkey command ignoring the `~` (pass-through) prefix if *Label* was omitted.

Fixed the Hotkey command ignoring the `$` (use-hook) prefix if the hotkey variant was already created without `$`.

Fixed `%Fn%` syntax to work with JavaScript Function objects.

Fixed EXE manifest to disable UAC installer detection heuristics (broken by v1.1.17).

Improved the way threads are represented on the debugger's call stack.

- The type of thread is shown instead of the label name, which is still available in the next stack entry. For hotkeys, the hotkey is shown instead.
- The line number of the sub or function that the thread called is shown instead of the line the script was at before starting the thread.

1.1.18.00 - December 30, 2014

Improved IDispatch implementation for AutoHotkey objects:

- `X.Y` in VBScript and C# now returns `X.Y` if it is NOT a Func object; previously, it attempted to call `X.Y()` regardless of type.
- `X(Y)` in VBScript, JScript and C# now returns `X[Y]` unless `X` is a Func object, in which case it is called as before.
- `X[Y]` in C# now returns `X[Y]` if `X` is NOT a Func object; previously, it attempted to call `X[Y]()` regardless of type.
- `X.Y()` in C# now returns `X.Y` if it is NOT a Func object, due to ambiguity in the C# dispatch implementation.
- Unhandled exceptions are now converted to IDispatch exceptions.

Added support for creating new properties in JavaScript/IE DOM objects.

Fixed `FileAppend`, `%VarContainingClipboardAll%`, `File` causing crashes.

1.1.17.01 - December 28, 2014

Fixed COM event handlers not receiving the final object parameter.

1.1.17.00 - December 27, 2014

New features:

Added [Format\(\)](#).

[FileOpen\(\)](#) can now be used to read from stdin or write to stdout or stderr by passing "*" or "*" as the filename. `AutoHotkey.exe *` can be used to execute script text piped from another process instead of read from file. For an example, see [ExecScript\(\)](#).

Added support for passing AutoHotkey objects to COM APIs as IDispatch.

Added support for VT_BYREF in ComObject wrappers. Use `obj[]` to access the referenced value.

Bug-fixes:

Fixed blue-screening on XP SP2.

Fixed owned ComObjArrays to be returned by value, not by pointer. That is, a copy of the array is returned instead of a pointer to an array which is about to be deleted.

Changed the URL in the sample script which is created when you first run AutoHotkey.

Fixed `sc15D` to map to `vk5D` / AppsKey (Send, GetKeyName, etc.).

Fixed Edit controls to not treat AltGr+A as Ctrl+A.

Fixed static class vars to not act as a target for labels which immediately precede the class.

1.1.16.05 - September 28, 2014

Fixed `x.y/[]/{}` sometimes falsely being flagged as variadic.

1.1.16.04 - September 16, 2014

Fixed a crash which occurred when overwriting a non-writable property.

Fixed a crash which occurred when a RegExMatch object is created with uncaptured subpatterns during a regex callout.

1.1.16.03 - September 11, 2014

Fixed some bad behaviour in Abs() and Mod() caused by a broken bug-fix in v1.1.16.01.

1.1.16.02 - September 11, 2014

Fixed parser to allow `#include` inside a class (broken by a6ea27f).

Fixed Clipboard returning binary `CF_HDROP` data in some instances.

Improved accuracy of `FileGetSize` on files which are still open.

Improved for-loop compatibility with COM objects.

1.1.16.01 - September 10, 2014

Added syntax for defining dynamic [properties](#) in classes.

Added `x+m` and `y+m` options for Gui control positioning.

Added the `#InputLevel` of each hotkey to `ListHotkeys` (when non-zero).

Optimized `RegExMatch` with the `O` (object) option; capturing part of a long string is much faster than before in some cases.

Fixed objects to check for new methods after `__Call` completes (consistent with `__Set` and `__Get`).

Fixed some undefined behaviour occurring after `VarSetCapacity` fails.

Fixed `FileRead` to correctly report an error with files over 4GB, instead of truncating even when it wasn't requested.

Fixed `FileRead *c` to null-terminate correctly when byte length is odd.

Fixed some `ClipboardAll` issues and changed behaviour:

- [#ClipboardTimeout](#) is ignored for `GetClipboardData()`, since it never actually worked and it caused problems. This fixes timeouts and lost data when a file is on the clipboard (any file on Windows 8, or any file in a zip file or similar).
- Zero-length clipboard items are allocated 1 byte upon restoring to the clipboard, since setting a zero-length item seems to always fail.
- The presence of `MSDEVColumnSelect/MSDEVLineSelect` is now preserved when set by `Scintilla`, though any data set by `VS` is discarded.
- When writing `ClipboardAll` directly to file, it now prefers Unicode (other usages of `ClipboardAll` have had this behaviour since v1.1.08).
- `ClipboardAll` and `FileRead *c` on Unicode builds now round length up when odd. This increases the reported `StrLen` by 1, potentially avoiding truncation of the last byte.

Fixed `A_EventInfo` for `TreeView` items with negative screen coords.

Fixed a possible ListView/TreeView message filtering issue.

Fixed A_LineFile/A_LineNumber in #If expressions.

Fixed #If to add itself to ListLines (though as "If").

Fixed `Abs([" - "] *)` and similar crashing the program.

Fixed `} funcdef(){ globally and }` ending a method/property/class.

1.1.15.04 - August 12, 2014

Fixed a performance issue with `&var_containing_long_string_of_digits`.

Fixed built-in variables corrupting memory when an object is assigned.

Fixed parser to disallow `continue` where appropriate even if the current function is defined inside a Loop.

Fixed `File.Encoding := x "-RAW"` breaking encoding/decoding of non-ASCII characters. The `-RAW` suffix is now ignored.

1.1.15.03 - August 2, 2014

Fixed meta-functions to exit the thread correctly after using Exit or Throw.

Fixed FileInstall to use A_WorkingDir when Dest is relative in a non-compiled script.

1.1.15.02 - July 7, 2014

Fixed debugging negative integer keys in objects.

1.1.15.01 - June 30, 2014

Changed built-in error and warning dialogs to be always-on-top.

Fixed `Until` not breaking out of recursive file/registry loops.

1.1.15.00 - May 4, 2014

Added A_OSVersion value `WIN_8.1`.

Changed the Hotkey command to apply/remove passthrough behavior on existing hotkey variants depending on whether the [tilde \(~\) prefix](#) is present.

Changed exe manifest to allow [GetVersion/GetVersionEx](#) to detect Windows 8.1.

1.1.14.04 - April 25, 2014

Fixed DllCall() crashing the script when *Function* is pure integer 0.

Fixed `IsFunc("ComObj(")` to return false instead of throwing an exception.

Fixed ControlClick to send XButton1/XButton2 correctly.

1.1.14.03 - February 14, 2014

Fixed a memory leak in `new X` when `X.__Init` aborts the thread.

Fixed breakpoints shifting onto class var initializers when they are interspersed with method definitions.

1.1.14.02 - January 30, 2014

Fixed `x::` and `x up::` both firing on key-up if the state of the modifier keys prevented `x` from firing on key-down. The intended behaviour as of v1.1.14 is for both hotkeys to fire on key-up if `x` is also used in a custom combination like `x & y::`.

1.1.14.01 - January 15, 2014

Fixed Try without Catch/Finally crashing if no exception was thrown.

1.1.14.00 - January 15, 2014

Added **Finally** statement for performing cleanup after try/catch. [fincs]

Changed behaviour of `~ prefix` with custom combos to be more intuitive.

Fixed `x & y::` to fire both `x::` and `x up::` when x is released.

Fixed ImageSearch to set ErrorLevel=2, not 0, when a GDI call fails.

Fixed EnvGet crashing the program when the var exceeds 32767 chars.

Fixed a minor error-handling bug with StatusBarWait's ErrorLevel.

Fixed a cause of heap corruption with FileRead *c on ANSI builds.

Fixed passing of local vars in recursive calls to variadic functions.

Fixed script failing to load if the first line is an expression in parentheses.

Fixed PixelSearch Fast mode to set OutputVarX/Y only on success.

1.1.13.01 - October 11, 2013

Fixed variadic function calls with named values for required parameters.

1.1.13.00 - August 27, 2013

Ported [StrSplit\(\)](#) from v2 alpha.

1.1.12.00 - August 14, 2013

Optional parameters can be omitted by writing two consecutive commas, as in `InStr(a, b, , 2)`. Unlike previous versions, this now works for objects (including COM objects) and built-in functions. `[a, , b]` can be used to create a sparse array.

Object properties can now be set using variadic syntax, as in `x[y*]:=z`, where `y` contains an array of keys/indices or parameters.

1.1.11.02 - July 28, 2013

Fixed GuiControl/Get to accept the ClassNN of a ComboBox's child Edit.
Specifying the HWND of a ComboBox's child Edit was already supported as a means of identifying the ComboBox.

1.1.11.01 - June 25, 2013

Fixed InputBox default width/height (broken by v1.1.11.00 - commit [7373cc6443](#)).

Fixed DllCall arg type validation to handle SYM_OBJECT safely, and to respect #Warn UseUnset (when var name is not a valid type).

Changed VarSetCapacity(var) to never warn about uninitialized vars.

1.1.11.00 - June 21, 2013

Added support for `%A_LineFile%` in `#Include`.

Reduced the file size of AutoHotkeySC.bin by reducing the resolution of the (rarely used) filetype icon.

Fixed `class X extends Y` to allow Y to be defined after X.

Developed by fins: AutoHotkey is now DPI-aware. The Gui commands automatically compensate when DPI is not 96. See [Gui -DPI Scale](#).

1.1.10.01 - May 17, 2013

Improved RegRead to support REG_BINARY values larger than 64K.

Improved RegWrite to support REG_BINARY/REG_MULTI_SZ values larger than 64K.

Fixed Process Close (and possibly other things) on Windows XP.

1.1.10.00 - May 11, 2013

Added basic support for [custom Gui control types](#).

Revised [SoundSet](#), [SoundGet](#) and 'WaveVolume commands to better support Windows Vista and later.

Fixed `Run "<file>" <args>` to not pass the space as part of the args.

Fixed some issues with reading from console input (CONIN\$).

1.1.09.04 - March 14, 2013

Fixed Gui menu accelerator keys not working when the Gui itself (not a control) has focus.

1.1.09.03 - February 9, 2013

Fixed [two bugs](#) which affected certain assignments where the target variable contains an unflushed binary number but has zero capacity.

Fixed GuiControl and GuiControlGet acting on the wrong control when given a HWND which does not belong to the current Gui. Instead, the condition is treated as an error.

Fixed OnMessage functions to set A_Gui and A_GuiControl correctly when the target Gui is a child window.

1.1.09.02 - December 24, 2012

Fixed MsgBox to show an error message when given invalid Options, instead of silently exiting the thread.

Fixed syntax errors in class var initializers causing the program to crash without the proper error message.

1.1.09.01 - December 15, 2012

Fixed AND/OR following a multi-statement comma, like `((x, y) and z)`.

Fixed RegExReplace to support duplicate subpattern names correctly.

Fixed Object.Remove() not freeing string keys.

Fixed base.Invoke() to not depend on dynamic variable resolution.

1.1.09.00 - November 7, 2012

Breaking changes:

Removed `.aut file compatibility` and `#AllowSameLineComments`.

Removed undocumented AutoIt v2 commands.

Backward-compatible changes:

Allow optional parameters to be declared with `:=` instead of `=`, for consistency with variable declarations and expressions.

Allow non-ASCII characters in unquoted keys (`{key: x}` and `obj.key`).

Changed name in version info from "AutoHotkey_L" to "AutoHotkey".

Bug-fixes:

Fixed instability caused by warning dialogs appearing part-way through evaluation of a command's args.

Fixed PixelSearch Fast mode treating "not found" as an error.

Fixed `ahk_exe` to allow other `ahk_` criteria after it.

Fixed `else continue 2` and similar requiring braces around the loop.

Fixed RegRead indicating failure after successfully reading binary data.

Fixed `File.Length` to compensate for or flush any buffered data.

Fixed Gui sizing to account for scrollbars when present.

Debugger:

Added support for asynchronous commands.

Improved handling of breakpoints on lines like `else foo()` or `{ bar()` so that the debugger will actually break when appropriate.

Optimized code size, fixed several bugs and made the debugger generally more robust.

1.1.08.01 - August 3, 2012

Debugger: Fixed max_depth being either half the intended limit or unlimited.

1.1.08.00 - July 14, 2012

Breaking changes:

Changed the default script codepage to ANSI, as the previous behaviour was a common source of confusion. UTF-8 files must now have a byte order mark (BOM) to be recognized correctly. Notepad adds a BOM to any file saved as UTF-8.

Changed `return x` to preserve formatting of numeric strings; it is now equivalent to `return (x)`.

Changed `Gui, Name: New` to set the new GUI as the default.

Other changes:

Changed ClipboardAll to prefer CF_UNICODETEXT over other text formats.

Changed Gui Show to allow floating-point numbers.

Changed A_OSVersion to return WIN_8 on Windows 8.

Changed AutoHotkey.exe file description to include "ANSI/Unicode 32/64-bit".

Changed the parser to allow lines like `new MyObject()` with no assignment.

Upgraded PCRE to 8.30.

Improved wording of some [warnings](#) and added a pointer to the documentation at the bottom of the warning dialog.

Improvements to the debugger:

- Added basic support for inspecting Func, ComObject and RegExMatchObject objects.
- Fixed the "attach debugger" message being ignored in some situations. See commit [83f0a0e](#).

New features:

Added support for the PCRE construct `(*MARK:NAME)` via `RegexMatchObject.Mark`.

Added support for `classvar.x := y` assignments in class definitions (after declaring *classvar*).

Added [A_Is64bitOS](#).

Added [SetRegView](#) and [A_RegView](#).

Fixes:

Fixed `Func.()` to merely skip the function call if mandatory parameters were omitted instead of causing the thread to silently exit.

Fixed `Object.Remove()` not releasing object keys.

Fixed key-down/key-up hotkey pairs to suppress input correctly when only one hotkey in the pair is enabled.

Fixed `#Include <Lib>` to not affect the working directory used by subsequent `#include` directives.

Fixed `objaddrf()` acting like `ObjRelease()` when called with a lower-case 'a'.

Fixed `A_AhkPath` in 32-bit compiled scripts to detect 64-bit AutoHotkey installations and vice versa.

Fixed `TreeView` controls with `-Background` option rendering incorrectly on older OSes.

Fixed error messages shown during execution of `While/Until` to identify the appropriate line.

1.1.07.03 - March 25, 2012

Fixed Ctrl/Alt/Shift key-down hotkey breaking any corresponding key-up hotkey defined before it.

Fixed key-down hotkeys to always use the hook if defined after a key-up hotkey of the same key. If the key-down hotkey used the "reg" method, the hook would block the key-down event and trigger only the key-up hotkey.

Fixed load-time checks interpreting expressions with no derefs as raw text. For example, `gosub % "foo"` resolved to `"foo":` at load-time. Similarly, `% ""` was seen as invalid in various cases even if an empty string should be allowed.

1.1.07.02 - March 22, 2012

Fixed a rare crash which can occur when `GetClipboardData()` fails.

Fixed `ComObjArray()` to return an empty string on failure instead of an arbitrary integer.

Fixed `Object.Remove(i, "")` affecting integer keys when `!Object.HasKey(i)`.

1.1.07.01 - March 2, 2012

Fixed FileRead to ignore #MaxMem when reading UTF-16 on Unicode builds.

Fixed dynamic function calls with built-in vars such as `%A_ThisLabel%`.

1.1.07.00 - February 27, 2012

Enhanced `%var%()` to support [function objects](#) and the default `__Call` meta-function.

Fixed `ControlGet List` to work for ListViews where the script and the target process aren't both 32-bit or both 64-bit.

Fixed `SendEvent` with a key delay of 0; a change introduced by v1.1.05.04 caused it to be slower than intended.

Fixed `Object.Remove(i)` not adjusting keys if `Object[i]` doesn't exist.

Fixed an error in `ComObjType()` which may have caused unpredictable behaviour when it is called via an [alternative name](#).

Fixed `ExitApp` to exit the thread if an `OnExit` subroutine prevents the script from terminating, instead of resuming execution after the `block` which contained `ExitApp`.

Calling a function via an object no longer acts as a barrier to exiting the thread. Consequently, if `Exit` is used or a runtime error occurs within a [class method](#), the entire thread exits instead of just the function.

Calling a base-class method using `base.Method()` or similar inside a class definition no longer causes a `UseUnset` warning if the method doesn't exist. Consequently, instantiating an object which has [instance variables](#) no longer causes a warning about the internal `base.__Init` method if the base class has no instance variables.

1.1.06.02 - February 13, 2012

Fixed IniRead crashing when Section is omitted but Key isn't.

Fixed accuracy of FileGetSize with files which are 4GB or larger.

1.1.06.01 - February 12, 2012

Fixed MsgBox smart comma handling to require numeric Options, not Title.

1.1.06.00 - February 12, 2012

New features:

Integrated `#InputLevel` directive and `SendLevel` command by Russell Davis.

Integrated support for `Link` controls by ChrisS85.

Breaking changes:

Changed command parser to avoid trimming escaped characters such as ``t` or ``` at the beginning or end of an arg.

Changed `MsgBox`'s smart comma handling to improve flexibility and consistency.

- `%` can now be used to make `Options` or `Timeout` an expression.
- If the first arg is an expression, any unescaped comma which is not enclosed in quote marks or parentheses/brackets/braces will cause multi-arg mode to be used. These commas were formerly interpreted as multi-statement operators within the first-and-only arg (`Text`).
- When `Title` is an expression, unescaped commas contained within the expression no longer interfere with smart comma handling.
- If there are exactly two args and the first is empty or an integer, multi-arg mode is used. The former behaviour was to combine both into a single arg (`Text`).
- `Timeout` can be a literal number or a single deref (and optionally part of a number; for example, `%Timeout%.500`). Contrary to the documentation, the former behaviour interpreted most other cases beginning with `%` as expressions (containing a double-deref).
- `Title` can be an expression even if `Text` and `Options` are omitted.

Changed `A_IsUnicode/A_IsCompiled` to be defined as an empty string in ANSI versions/uncompiled scripts instead of being left undefined. This allows them to be checked without triggering `#Warn` warnings. Side effects include:

- Attempting to assign directly to `A_IsCompiled` or `A_IsUnicode` always

causes a load-time error. Dynamic assignments always fail, either silently or with an error dialog as appropriate. Previously assignments were allowed in uncompiled/non-Unicode scripts.

- Attempting to take the address of `A_IsCompiled` or `A_IsUnicode` always fails.
- `A_IsCompiled` and `A_IsUnicode` no longer appear in `ListVars` when referenced by a script which is not compiled/Unicode.

Changed `Send` and related commands to respect `#MenuMaskKey` when changing modifier keystates.

Other changes:

Changed `GuiControl Choose` to remove the `ListBox/ComboBox/DDL`'s current selection when `N=0`.

Changed `RegisterCallback` to allow a `Func object` in place of a name.

Changed `ListLines` to show filenames (except when compiled).

Improved `Run` to output a process ID if possible when `ShellExecuteEx` is used.

Fixes:

Fixed handle leaks in `RegRead`.

Fixed `x.y++` and similar to assign an empty string if `x.y` is non-numeric.

Fixed `SendInput Win+L` workaround causing `Win` key to "stick down".

Fixed `Ahk2Exe` auto-including the wrong file if an auto-include used `#Include <Lib>`.

1.1.05.06 - December 31, 2011

Fixed inc (++) and dec (--) having no effect when used on an object field containing a string.

Fixed inc (++) and dec (--) to cause a warning when used on an uninitialized variable.

1.1.05.05 - December 17, 2011

Fixed `continue n` misbehaving when an inner loop has no braces.

Fixed `RegexMatchObject[Name]` to work correctly with duplicate names.

1.1.05.04 - December 5, 2011

Fixed: Selected sub-command of ControlGet was unreliable on x64.

Fixed: CPU was maxed out while waiting if an underlying thread displayed a dialog (and possibly in other cases).

1.1.05.03 - November 30, 2011

Fixed Loop ... Until VarContainingObject.

1.1.05.02 - November 20, 2011

Fixed false detection of end-of-file when loading a compiled script in some cases.

Fixed SendInput to automatically release modifier keys when sending special characters or {U+xxxx} (which was broken by v1.1.00.01).

Fixed ComObjConnect to filter out non-dispatch interfaces instead of allowing the script to crash.

Fixed `new %VarContainingClassName%()`.

1.1.05.01 - October 16, 2011

Fixed class declarations to allow directives such as #Include.

1.1.05.00 - October 8, 2011

Added [Client coordinate mode](#).

Added [object output mode](#) for RegExMatch and RegEx callouts.

Added [super-global](#) declarations, which avoid the need to repeat global declarations.

Breaking change: Class declarations such as `CLASS C` now create a super-global variable instead of an ordinary global.

Added more detail to unhandled exception error dialogs.

Changed [Gui +Owner](#) to work even after the GUI is created.

Changed instance var declarations in class definitions to avoid leaving empty key-value pairs in the class object.

Changed `#Include` to use standard error message formatting when it fails (more detail; also respects `ErrorStdOut`).

Changed [Throw](#) to throw an Exception object by default when its parameter is omitted.

Changed format of runtime error messages slightly to improve consistency and code size.

Modified PCRE to use UTF-16 for input on Unicode builds, for performance.

Upgraded PCRE to 8.13.

Fixed thread not exiting as intended when an assignment in an expression fails.

Fixed `#MaxMem` preventing assignments to variables which don't require expansion.

Fixed inability of Try/Catch to catch COM errors.

Fixed GuiControlGet incorrectly treating parameter #2 as invalid in some cases.

Fixed input vars being resolved too soon, allowing them to be invalidated by an expression later on the same line.

Fixed RegEx callouts not causing matching to abort when an exception is thrown.

Fixed DllCall setting ErrorLevel to -4 when it should be -3.

Fixed While appearing twice in ListLines for its first iteration.

Fixed Try/Catch to allow If/Loop/For/While with no enclosing block.

Fixed enumerator object not being released when Until is used to break a For-loop.

1.1.04.01 - September 15, 2011

Fixed FileRemoveDir setting ErrorLevel incorrectly.

1.1.04.00 - September 11, 2011

Warning: This release contains a number of potentially script-breaking changes.

Added exception handling support: `try/catch/throw` and `Exception()`.

Added StdOut mode for `#Warn`.

Added `Gui +HwndVARIABLE` option.

Added `Gui, New [, Options, Title]`.

Added automatic support for keyboard accelerators such as Ctrl+O in `Gui menus`.

Changed handling of `#Include <Lib>` when the `/iLib` command-line switch is present to resolve a `problem` with Ahk2Exe.

Changed `GuiControl` to retain the Progress control's current value when its style is changed.

Changed `GuiControl` and `GuiControlGet` to allow a HWND to be passed in the `ControlID` parameter.

Removed the 1GB limit from `FileRead`.

Improved error detection:

- `Hotkey, If, Expression`, where *Expression* does not match an existing `#If` expression, is caught at load-time when possible.
- `Hotkey, IfSomething`, where *Something* is invalid, is caught at load-time.
- Class definitions with missing braces are detected as errors.
- If a function call is used on the first line of a class definition, it is now correctly treated as an error.
- `GroupAdd` now shows an error message when the group name is blank, instead of silently exiting the thread.

- Removed some redundant "unset var" warnings which appeared when using the OutputDebug or StdOut warning modes.
- If an unrecognized option is used with [Gui](#), [Gui Show](#), [Gui New](#) or [GuiControl](#), an error message is shown and the thread exits unless [try](#) is used. This validation is not done at load-time due to complexity (it is common for the option parameters to contain variable references).
- RegRead, RegWrite and RegDelete now set A_LastError to the result of the operating system's GetLastError() function.
- [+LastFoundExist](#) is now treated as an error if it is combined with another option ([+LastFound](#) should be used in that case).

Fixed a [bug](#) affecting recursive variadic functions.

1.1.03.00 - August 28, 2011

Added support for GUI names.

Added support for identifying a GUI by its HWND.

Added `+Parent%ParentGui%` Gui option.

Added support for external windows as Gui owners via `+Owner%HWND%`.

Added Name sub-command for GuiControlGet.

Added support for ActiveX controls via the Gui command.

Fixed: Empty hotkey control returned "vk00".

Fixed: Crashes and memory leaks related to COM events/ComObjConnect.

Fixed: `GuiControlGet OutputVar, Subcmd, %OutputVar%` always failed.

Changed "Missing (/[{" error messages to "Unexpected)/]" for greater clarity.

Changed ListLines to display While and Until lines which are executed each iteration.

Changed `~=` to have higher precedence than `!=</><=>=` but lower than concat, and added it to the documentation.

1.1.02.03 - August 21, 2011

Fixed (Debugger): numchildren attribute did not include Object.Base.

1.1.02.02 - August 20, 2011

Fixed: Variable capacity was capped at 2GB on x64 builds.

Fixed: Last Found Window not set by `#if WinExist(T)`.

1.1.02.01 - August 13, 2011

Changed A_PriorKey to exclude key-up events.

Fixed process name/path retrieval in certain cases, including:

- Retrieving name/path of a 64-bit process from a 32-bit script.
- Retrieving name/path of an elevated process from a non-elevated process (UAC).

1.1.02.00 - August 6, 2011

Added TV_SetImageList().

Characters which require non-standard shift states 16 and 32 now use a fallback method instead of producing the wrong keystrokes.

Revised handling of dead keys to solve problems which occur when multiple scripts with hotstrings are active.

1.1.01.00 - July 30, 2011

Added support for instance variables in class definitions, using simple assignment syntax.

Removed `var` keyword used in class definitions; use `static` instead.

Added new built-in variables: `A_ScriptHwnd` and `A_PriorKey`.

Added new built-in functions: `GetKeyName()`, `GetKeyVK()`, `GetKeySC()` and `IsByRef()`.

Added new sub-command: `WinGet`, `OutputVar`, `ProcessPath`.

Added the capability to specify a window by process name or path: `ahk_exe %Name%.exe` or `ahk_exe %FullPath%`.

Optimized `ProcessName` sub-command of `WinGet`.

Changed `SetTimer` to use `A_ThisLabel` if `Label` is omitted.

Updated `ComObjConnect()` to support using an object in place of a function name prefix.

Improved `ComObjConnect()` to allow the prefix/object to be updated without first disconnecting.

Improved parsing of continuation sections to allow expressions like `(x.y)[z]` `()` without escaping "(".

Replaced the method used to store script text in 32-bit compiled scripts; now consistent with 64-bit.

Fixed detection of `AltGr` in the active window's keyboard layout (on Unicode builds).

Fixed `SendInput` applying a redundant `LCtrl`-up some time after `AltGr`-up.

1.1.00.01 - July 17, 2011

Fixed: Modifier keys weren't auto-released when sending special chars.

Fixed: Scancode/modifier-key mapping conflicts such as sc1xx vs ^sc0xx.

Fixed: \$ and #UseHook had no effect if used only on the second or subsequent instance(s) of a hotkey.

Fixed: Potential crash when returning a value from a __Delete meta-function.

Fixed: "Uninitialized variable" warnings were triggered by the debugger.

Changed: `base.Method()` no longer triggers a default meta-function or a warning.

Changed: `Gui +(Ex)Style` no longer hides/shows the Gui.

Changed the debugger to report type="undefined" for uninitialized variables.

Added check to avoid incorrectly sending keystrokes for characters which actually require the "hankaku" key.

Added support for integers as class variable names.

Added "Static" keyword for declaring class variables.

1.1.00.00 - May 1, 2011

New features:

Implemented basic [class definition syntax](#).

Implemented the [new](#) keyword for creating a derived object.

Added [Func\(\)](#) for retrieving a reference to an existing function and improved [IsFunc](#) to recognize [function references](#).

Added support for [++](#) and [--](#) with object fields, such as [x.y\[z\]++](#).

Changes:

Changed `__Delete` to not trigger `__Call`.

Changed `OnClipboardChange` to use `AddClipboardFormatListener` when available (i.e. on Windows Vista and later) to improve reliability.

Auto-concat with `(` is more selective, so some invalid expressions like `12(34)` will no longer work.

Fixes:

Fixed `SetTimer Label, -0` to be treated as "run-once, very soon".

Fixed `A_MyDocuments` etc. to use `SHGetFolderPath` instead of undocumented registry keys.

Fixed non-empty `ExcludeText` causing windows with no text to be excluded.

1.0.97.02 - April 14, 2011

Fixed misinterpretation of comma as an arg delimiter when enclosed in `{ }` in an expression.

Fixed For-loop to set `A_Index` only after the *Expression* arg is evaluated, so that the outer loop's value can be used.

Fixed default script location to be based on the directory containing the EXE (as documented), not the working directory.

Improved load-time validation to detect invalid attempts to jump out of a function with `Goto`.

1.0.97.01 - April 2, 2011

Fixed the 64-bit build to not truncate HWNDs or SendMessage/PostMessage params to 32 bits.

Fixed `* / ::` being treated as an invalid hotkey (broken since L54).

Fixed the icons.

1.0.97.00 - March 28, 2011

Added `{key: value}` as syntax sugar for `Object("key", value)`.

Added `[x, y, z]` as syntax sugar for `Array(x, y, z)`, which is a new function equivalent to `Object(1, x, 2, y, 3, z)`.

Added slight optimization: resolve any reference to `True`, `False`, `A_PtrSize` or `A_IsUnicode` in expressions at load-time.

Fixed hotkey parser to treat `x & ^y` as an error instead of ignoring `^`.

1.0.96.00 - March 21, 2011

New features:

ComObjQuery: Queries a COM object for an interface or service.

ComObjFlags: Retrieves or changes flags which control a COM wrapper object's behaviour.

ComObjCreate allows non-dispatch objects to be created if an interface identifier (IID) is given.

COM arrays support for-loops and the Clone() method.

ListVars shows the inner variant type and value of each COM wrapper object, in addition to the wrapper's address.

Changes:

When a literal integer or variable containing both a numeric string and cached binary integer is assigned to a field of an object, an integer is stored instead of a string. This particularly benefits scripts calling COM methods which accept integers but not strings, or have different behaviour depending on the type of value.

NULL values are now allowed with ComObjParameter for the VT_UNKNOWN and VT_DISPATCH types.

Improved support for Common Language Runtime (.NET) objects via COM.

FileRecycle should now warn before deleting a file which is too large to recycle.

When a SafeArray created with ComObjArray is assigned to an element of another SafeArray, a separate copy is created. This prevents the script from crashing due to a second attempt at "destroying" the array. ComObjFlags allows this behaviour to be avoided.

Fixes:

Assigning to a COM array of VT_DISPATCH or VT_UNKNOWN crashed the script.

Break and Continue were tolerated outside of loops in some cases.

Standalone carriage-return (`\r`) was not correctly interpreted as end-of-line.

MouseMove miscalculated negative coordinates with the Input and Event send modes.

Selecting *Edit This Script* from the tray menu or using the Edit command crashed the script (broken by v1.0.95).

Error dialogs pointed at irrelevant lines of code in some cases.

1.0.95.00 - March 12, 2011

All file I/O has been heavily optimized.

Added `#Warn` to assist with debugging; initial design by ac.

By default, if `name_var` contains a function name, `name_var . ()` calls the function. This can be overridden via the [default base object](#), as before.

Run supports verbs with parameters, such as `Run *RunAs %A_ScriptFullPath% /Param .`

If an operator which can accept either one operand (`&x`) or two *numeric* operands (`x & y`) follows a quoted literal string, auto-concat occurs and the operator is applied only to the right-hand operand. This is because quoted literal strings are always considered non-numeric and are therefore not valid input for numeric operators. For example, expressions like `"x" &y` and `"x" ++y` now work.

Fixed:

- Wildcard hotkeys were not respecting modifiers such as `^!+` in specific cases.
- `File.Pos` returned garbage for non-seeking file types; now it returns -1.
- `File.AtEOF` was incorrectly true in some cases.
- COM wrapper objects left `A_LastError` unset in some cases.
- Gui submenu icons did not work on Windows 2000/XP/Server 2003.
- `SplashImage` clipped the image if `height > width`.
- `ComObjConnect` did not alert when the first parameter is invalid.
- `SplashImage` now uses GDI+ only when the other methods fail, for compatibility.
- Tilde in `~x::` now affects `x & y::` in the same way that `~x & z::` would, instead of having no effect.
- `A_PriorHotkey` and `A_TimeSincePriorHotkey` now have the expected values when used with `#If`.
- `RegExReplace` did not advance through the string correctly after a match failure if the string contained non-ASCII characters.

1.0.92.02 - January 19, 2011

Fixed a memory leak which occurred when the return value of an object invocation or built-in function was not used, such as `file.ReadLine()` or `SubStr(x, y, z)` alone on a line.

Replaced the fix from v1.0.92.01 with a better fix, which also fixes `k::MsgBox()`, `x:=y` and doesn't break `if()`.

1.0.92.01 - January 18, 2011

Changed: FileCreateDir now sets A_LastError.

Fixed: `GuiControl()` or similar was sometimes misinterpreted as a command.

1.0.92.00 - January 13, 2011

Added support for compound assignments with objects, such as `x.y += z`.

Improved IniWrite in Unicode builds to prevent an empty line from appearing at the top of each new file.

Improved the parser to be more permissive about what can follow `{ / } / Else`, especially when combined.

1.0.91.05 - January 1, 2011

Fixed: Cleanup code in COM method calls was freeing the wrong parameters.

Fixed (ANSI): DllCall set incorrect length for WStr parameters on output.

Fixed: Variadic function calls were unstable when param object was empty.

1.0.91.04 - December 29, 2010

Fixed (Unicode): RegExReplace omitted all text up to StartingPosition (broken by v1.0.90.00).

1.0.91.03 - December 27, 2010

Fixed: RegEx callout subpattern vars - broken by v1.0.90.00.

1.0.91.02 - December 26, 2010

COM: Added protection against NULL IDispatch/IUnknown pointers.

COM: Skip QueryInterface for IDispatch if VT_DISPATCH is passed explicitly.

Minor fix for maintainability: `obj.field := var` now correctly yields SYM_STRING, not SYM_OPERAND.

1.0.91.01 - December 24, 2010

Fixed: Unexpected results with `File.Write(Str)` after text-reading.

Fixed: UTF BOM check caused unexpected results in files without a BOM.

Fixed (ANSI): Parsing loops and `File.ReadLine()` treated `Chr(255)` as EOF.

Fixed (Unicode): `RegexReplace` errors where the UTF-8 and UTF-16 lengths differed.

Fixed: Disabling the last hook hotkey breaks Input.

Added: Simple optimization for `RegexMatch/Replace`.

1.0.91.00 - December 21, 2010

All changes in this release are COM-related.

Added: `ComObjError` now returns the previous setting.

Added: `ComObjType(co)` and `ComObjValue(co)`.

Added: `ComObjMissing()`.

Added: `ComObjArray()` and basic SAFEARRAY support.

Added: "Take ownership" parameter for `ComObjParameter()`.

Changed: Values passed to COM functions via `ComObjParameter` are no longer freed in the process of calling the function.

Changed: `ComObj.x()` now falls back to PROPERTYGET if member 'x' is not found. This fixes for-loops for some objects.

Changed: Wrap unhandled variant types in an object rather than returning integer value.

Changed: Manage VT_UNKNOWN/VT_ARRAY lifetime automatically, by default.

1.0.90.00 - November 27, 2010

Fixed: UrlDownloadToFile in Unicode builds on Windows < 7.

Fixed: Upper-ANSI characters were sent as Alt+0 in ANSI build.

Fixed: File.Pos was incorrect after attempting to read past EOF.

Fixed: Escape sequences in #If expressions and static initializers.

Fixed: ClipboardAll sometimes crashed the script with certain formats.

Fixed: Transform HTML calculated length incorrectly for &#NNN; entities.

Fixed: VarSetCapacity now correctly ignores #MaxMem for ByRef variables.

Fixed: `FileAppend, , file.txt` set ErrorLevel=1 even on success.

Fixed: Match length was miscalculated for RegEx callouts with the P) option.

Integrated Sean's improvements to number handling in COM interop.

Optimized RegExReplace and RegExMatch in Unicode builds.

Revision 61 - October 3, 2010

Added: `ObjClone(Object)`, forgotten when `Object.Clone()` was implemented.

Added: Support for RegEx Unicode properties in Unicode builds. Also upgraded PCRE to 8.10.

Added: `Object.Remove(int, "")` removes `Object[int]` without affecting other keys.

Changed: `ComObj.xyz()` is now always treated as a method-call.

Changed: `Var := 123` is now left as an expression, for consistency. This makes `Var := 123` and `Var := (123)` equivalent, whereas previously the former assigned only a string and the latter assigned both a string and a cached binary integer. In particular, this avoids some confusing type mismatch errors with COM objects.

Fixed: Dynamic variadic calls to functions with mandatory parameters.

Fixed: The final parameter of an assume-global variadic function had to be explicitly declared local.

Fixed: Static initializers interfering with setting of breakpoints.

Fixed: More pointer size-related errors with PCRE callouts on x64 builds.

Fixed: Input with 'M' option treated Ctrl-M (``r`) as Ctrl-J (``n`).

Fixed: `Object.Remove(n)` returned 0 (not "") if `Object[n]` didn't exist.

Revision 60 - September 24, 2010

Added: [File.Encoding](#) for changing or retrieving the codepage in use by a File object.

Added: [Variadic functions and function-calls](#) and [indirect mode](#) for callbacks.

Added: [Object.Clone\(\)](#)

Changed: ByRef parameters no longer require the caller to supply a variable.

Changed: `Obj.foo := ""` now frees foo's contents as originally intended (but still doesn't remove the field).

Changed: OnMessage functions now tolerate optional and ByRef parameters.

Changed: RegisterCallback now enables execute access on callbacks in 32-bit builds (already did in 64-bit builds).

Changed: RegisterCallback now treats explicit ParamCount="" as omitted.

Fixed: For-loop was treating non-expression "0" as true for scripted enumerators.

Fixed: ComEvent (ComObjConnect) not working on the ANSI build.

Fixed: ComEvent (ComObjConnect) not correctly backing up/restoring local vars when calling a function which is already running.

Fixed: Buffer overflow in A_EndChar when there is no end char.

Fixed: Func->mNextFunc not initialized to NULL (used only by LowLevel scripts).

Debugger: Added the capability to retrieve an object's contents (with a supported debugger client).

Debugger: Fixed a few bugs.

Revision 59 - September 12, 2010

Fixed: #If *expression* should now evaluate each *expression* at most once per keypress in all cases.

Changed: SplashImage uses common image-loading routines; now supports PNG and similar, but results may differ from before for icon files.

Added: `For x,y in z`.

Added: `Loop .. Until expression`, usable with any Loop or For-loop.

Added: Named loops; `continue outer_loop`, `break 2`, etc.

Debugger: Encode stdout and stderr packets as UTF-8.

Debugger: Allow user to Abort, **Retry** or Ignore when the initial connection fails.

Debugger: Allow [attaching a debugger client](#) by sending the script a registered window message.

Debugger: Allow detaching debugger client by sending the "detach" DBGp command.

Revision 58 - September 5, 2010

Added: `static var := expression.`

Revision 57 - September 4, 2010

Fixed Str*, WStr* and AStr* DllCall return types on x64.

Added functionality to [InStr](#) to bring it in line with StringGetPos:

- If *StartingPos* is negative, the search is conducted right-to-left beginning at that offset from the end.
- An additional parameter is available to specify which *occurrence* of the string to find.

Added `#include <LibName>` for explicitly including a script from a [function library folder](#).

Added functionality to [IniRead](#) and [IniWrite](#) allowing an entire section to be read or written.

Added functionality to [IniRead](#) allowing a list of section names to be retrieved.

Added support for [custom verbs](#) to Run/RunWait: `Run *verb file`.

Made [improvements](#) to the way Run/RunWait extracts the action and its parameters from the *Target* parameter.

Changed [NumGet/NumPut](#) to allow *Offset* to be omitted entirely since *Type* is always non-numeric.

Removed the restriction that a label can't point to the end of a block.

Revision 56 - August 29, 2010

Added support for x64 compiled scripts. Requires Ahk2Exe_L and x64 AutoHotkeySC.bin.

Fixed: Deref operator (`*addr`) rejected addresses outside 32-bit range.

Fixed: `#If expression` memory allocation error on x64.

Fixed: Custom hotstring end chars with code points greater than U+00FF.

Fixed: Special characters being sent out of sequence by SendInput/Play.

Fixed: `*/` being discarded in continuation sections (see L54).

Revision 55 - August 19, 2010

Fixed: GroupActivate not setting ErrorLevel=1 if no Label was specified.

Fixed: Tab controls in x64 build.

Fixed: String-copy mode of StrPut when length is omitted; broken by L53.

Fixed: Data-alignment issues in x64 builds.

Changed: Set A_LastError for more File* commands to assist debugging.

Includes FileAppend, FileRead, FileReadLine, FileDelete, FileCopy, FileMove, FileGetAttrib/Time/Size/Version and FileSetAttrib/Time.

Excludes FileCopyDir, FileRecycle, FileMoveDir, FileRemoveDir, FileRecycleEmpty, FileCreateDir and FileInstall as the APIs used do not consistently set or return a Win32 error code.

Revision 54 - August 15, 2010

Fixed: RunAs in Unicode build. [jackieku]

Fixed: RegisterCallback in x64 build. [fincs]

Fixed: Executables failing to run on Win2k and WinXP prior to SP2. Win2k requires SP4 and updates (probably [KB 816542](#)). XP not tested.

Fixed: Dialogs such as MsgBox became unresponsive when interrupted by another script thread. Message filter/workaround added in L45 had to be removed.

Fixed: Multi-byte characters in default ANSI codepage; this was also causing a memory leak.

Fixed: ComObject now allows purely numeric property/method names.

Enhanced usability and capabilities of FileOpen/File object.

- FileOpen's *Flags* parameter accepts a human-readable string of flags.
- FileOpen's "update" mode has been renamed "read/write" and now creates the file if it didn't already exist.
- FileOpen sets `A_LastError` as appropriate.
- `File := FileOpen(handle, "h")` wraps an existing file handle in an object.
- `File.Seek(n)` defaults to `SEEK_END` when `n` is negative.
- `File.Pos` (or `File.Position`) can be used in place of `Tell` and `Seek`.
- `File.Pos`, `File.Length`, `File.AtEOF` and `File.__Handle` can be used without empty parentheses `()`.
- `File.Length` can be set, as in `File.Length:=n`.
- `File.ReadNum()` and `File.WriteNum(n)` read or write a number, where *Num* is a `NumGet`-compatible type name.
- `File.WriteLine(s)` is equivalent to `File.Write(s "`n")`, for symmetry with `File.ReadLine()`.
- `File.Read()` reads and returns the remainder of the file when used without parameters.

- File object now returns "" for unrecognized methods and invalid syntax rather than the numeric *address* of "".

Changed: GroupAdd's Label parameter now applies to the whole group.

[\[Discussion\]](#)

Changed: GroupActivate sets ErrorLevel on success/failure. (Same thread as above.)

Changed: */ at the beginning of a line is ignored if there was no /*.

[\[Discussion\]](#)

Removed ToCodePage and FromCodePage subcommands of Transform.

Revision 53 - August 8, 2010

Merged AutoHotkey64 - COM support and x64 compatibility.

x64 (*changes since AutoHotkey64*):

- Fixed floating-point return values and exception handling for DllCall.
- Fixed RegEx callouts.
- Re-enabled GetProcAddress optimisation for DllCall.
- NumPut and NumGet default to "Int64" in x64 builds.

COM (*changes since AutoHotkey64*):

- Allow ComObjParameter to wrap 64-bit values in 32-bit builds.
- Implemented more standard ref counting conventions in ComObjActive.
- Prevent extraneous AddRef for new ComObjects in certain cases.
- Don't treat pdispVal==NULL return value from Invoke as an error.
- Fail more predictably when given wrong type of object.
- Require explicit second param in ComObjType, don't assume "IID".
- Free local vars after ComEvent calls a function.
- Remove pVarResult param from ComEvent; use return value instead.
- Pass the original wrapper ComObject in each ComEvent call.
- Add context to ComError dialog; let the buttons make more sense.

Implemented common syntax validation code for enumerators. `Enum[]` and `Enum.Next()` are now supported with or without parameters. `Enum[]:=val` and similar obscure forms which worked in AutoHotkey_COM/AutoHotkey64 are not allowed.

"Ptr" type is still signed (unlike AutoHotkey64), for maximum flexibility and consistency.

Other changes:

Added: `Object.HasKey(key)`.

Added: `Object(obj)` increments obj's reference count and returns its address.

Added: `ObjAddRef()` and `ObjRelease()`.

Fixed: `A_ThisHotkey` now has the expected value when used with `#If`.

Fixed: `RunAs` in Unicode build (fixed by jackieku).

Changed: Default script codepage for ANSI builds is now CP0 (ANSI).

Changed: `x.y[]` is now equivalent to `(x.y)[]` rather than `x["y"]`.

Changed: Built-in methods can now be called with or without the underscore prefix (e.g. `obj.MaxIndex()` or `obj._MaxIndex()`).

Revision 52 (again) - July 10, 2010

Added support for built-in and environment variables in double-derefs.

Improved support for multi-byte characters in certain codepages (previously only 932, 936, 949, 950 and 1361 were supported).

Fixed: StrPut failed for codepages which don't support WC_NO_BEST_FIT_CHARS, such as UTF-7.

Fixed: Double-deref as object - `literal%var%.literal`.

Fixed: `StrPut("", Encoding)` null-terminated an invalid address.

Revision 52 - June 12, 2010

Fixed: ObjRemove's Key parameter is now optional, as with [Object._Remove](#).

Fixed: Files were improperly locked while reading, unlike in older versions.

Fixed: Uppercase Hex format caused negative integers to become unsigned.

Fixed: RegExMatch was corrupting output array items when specific conditions were met. See [bug report](#).

Revision 51 - April 11, 2010

Changed: Use <EXENAME>.ahk instead of AutoHotkey.ini or AutoHotkey.ahk.

Changed: Default to UTF-8 for script files; override with /CPnnn.

Fixed: Unpause when the script exits so that object __Delete meta-functions can run.

Revision 50 - March 27, 2010

Improved flexibility of [Object._Insert](#).

- `Object._Insert(x)` - inserts `x` at `_MaxIndex()="" ? 1 : _MaxIndex() + 1`.
- `Object._Insert(i, x, y, z)` - inserts `x, y, z` at `i, i+1, i+2`.

Improved flexibility of [Object._Remove](#).

- `Object._Remove(k)` - removes and returns the value associated with `k`.
- `Object._Remove()` - removes and returns the value at `_MaxIndex()`.

Added file share mode flags to `FileOpen` [by jackieku].

Fixed: 'P' option of `RegexMatch` incorrectly output 1 for subpatterns which did not match anything.

Fixed: `Object._SetCapacity(key, n)` was not null-terminating in some cases.

Fixed: `StatusBarGetText` returned only half of the text in Unicode builds.

Revision 49 - March 14, 2010

Added `obj._NewEnum()`.

Added `ObjMethod(obj)` for each built-in `obj._Method()`.

Changed: `ObjSet/Get/Call` can no longer be called/overridden by script.

Fixed: Potential crash caused by `VK_PACKET` check added in L48.

Fixed: Character codes in `VK_PACKET` events were potentially misinterpreted as scancodes.

Fixed: `ExcludeText` was ineffective after the first matching control.

Revision 48 - February 21, 2010

Fixed: Standalone `obj . ()` was not recognized as a valid expression.

ControlSend now uses WM_CHAR to support characters which have no associated keycode. For instance, this applies to "..." (en-US), "∞" in a Unicode build and `{U+221e}` in either build.

KeyHistory now shows the full 16-bit character code for VK_PACKET (VK=E7) events. [Send](#) may indirectly cause these to be generated for Unicode characters.

Revision 47 - February 13, 2010

Fixed (Unicode): ControlGet,List with ListView controls.

Revision 46 - February 11, 2010

Added `A_FileEncoding`, which returns the current default file encoding.

Added `StrPut` and extended `StrGet` with additional features.

Revision 45 - February 8, 2010

Added `_GetAddress`; `_SetCapacity`/`_GetCapacity` may now operate on a given field. [\[more info\]](#)

Added workaround for `WM_PAINT` infinite loop often caused by subclassing.

Allow `obj . ()` as a means to call the default method of an object; equivalent to `obj [""] ()`.

Dev: Added solution and project files for MSBuild 4.0 & VS2010. [\[more info\]](#)

Fixed (Unicode): 'Running' column in ListHotkeys.

Revision 44 - February 7, 2010

Fixed: Attempting to set "nothing" crashed the script. For instance, `x[] := z`.

Fixed (Unicode): Unable to find default script (Documents\AutoHotkey.ahk).

Fixed (Unicode): A_Desktop, A_ProgramFiles and several other built-in variables which read their value from the registry caused strings to be incorrectly terminated.

Fixed (ANSI): Characters in the range 0x80-0xFF couldn't be found by InStr or StringReplace.

Changed (ANSI): Treat ToCodePage/FromCodePage like other unsupported/invalid subcommands.

Revision 43 - January 29, 2010

Fixed: obj.Func() resulting in a random integer when it should be an empty string.

Revision 42 - January 28, 2010

Merged with AutoHotkeyU.

Numerous bug-fixes and improvements, some relating to the merge.

Revision 41 - December 20, 2009

Fixed: Floating-point numbers were equivalent to an empty string when used as keys in objects. They are now converted to numeric strings in the current [float format](#), as originally intended.

Revision 40 - December 13, 2009

Changed meta-functions and multi-param behaviour to improve flexibility. Meta-functions are now invoked **before** searching for an appropriate field in each base object; however, values in x.base will override meta-functions defined in x.base.base. Chaining meta-functions (inheritence) is now possible by avoiding "Return"

Improved Default Base functionality to be more consistent with real objects.

Changed (Debugger): While stepping through code, skip all braces except those which end a function.

Changed (Debugger): When stepping onto the hidden "Exit" at end of the script, consider it to be **below** the last physical line to avoid confusion.

Revision 39 - December 2, 2009

Fixed (Debugger): Error in Base64 decoding used by property_set.

Revision 38 - November 17, 2009

Added: `#MenuMaskKey` directive to change which key is used to mask Alt/Win keyup events. See [this thread](#) for background information.

Changed: If `x[1, 2] := 3` creates an object for `x[1]`, its base is no longer automatically set to `x.base`. See [the documentation](#) for a method of controlling this behaviour.

Revision 37 - November 7, 2009

Added: Support for `obj[method_name](params)` object-call syntax.

Revision 36 - November 4, 2009

Changed: If `x.y` or similar is immediately followed by an open-bracket (`[`), the right-most identifier preceding the bracket is treated as the first parameter of the operation. For instance, `a.b.c[d]` is equivalent to `ObjGet(a.b, "c", d)` and `x.y[z] := v` is equivalent to `ObjSet(x, "y", z, v)`. Previously each sub-expression preceding a bracket was evaluated separately.

Changed: If a method-call such as `x.y()` is followed immediately by an assignment, the parentheses are treated as brackets. For instance, `x.y(z) := v` is now equivalent to `ObjSet(x, "y", z, v)`.

Fixed: `If(expr)` and `While(expr)` with no delimiting space or comma.

Revision 35 - October 25, 2009

Fixed: Standalone expressions beginning with two or more "dots", such as `x.y.z()`. (Broken by L34.) Note that '(' or '[' or ':=' is still required.

Revision 34 - October 24, 2009

Changed: Setting a value within an object to an empty string now stores the empty string rather than removing the key-value pair from the object. `_Remove` can still be used to completely remove key-value pairs.

Changed: Command names must be terminated with a space, tab or comma. The following characters no longer bypass this requirement: `<>:+-*/!~&|^[]`. For instance, syntax errors such as `MsgBox< foo` and `If!foo` are now caught at load-time.

Fixed: `Return` now properly handles expressions which result in a variable containing an object. For instance, `Return x:=y`, `Return (x)`, `Return x, . . .` and similar should work now (`Return x` already worked).

Fixed: Multi-parameter `get/set` did not correctly support meta-functions for multiple objects (such as for `x` and `x[y]` in the expression `x[y, z]`).

Fixed: Cascading object-assignments such as the `x[y]:=z` in `r:=x[y]:=z` could not yield numbers or objects - broken by L33.

Fixed: `x._Remove(y)` crashed the script or removed the wrong item if `x` did not contain `y`.

Fixed: `x.=y`, `if x.y=...` and similar. May affect other expressions.

Fixed: Standalone ternary expressions no longer requires spaces. For instance, `x? F(1) : F(2)` is now allowed.

Debugger: On script exit, disconnect debugger *after* releasing objects to allow debugging of `__Delete` handlers.

Revision 33 - October 3, 2009

Fixed: Local variables were not automatically freed for functions which return objects or pure numbers.

Fixed: Two separate reference-counting errors related to ternary operator and Object().

Fixed: If a string returned by a scripted object function was assigned directly to a variable, the variable's length was set incorrectly.

Fixed: If the last operation in an expression retrieved a string from a temporary object, the object and string were freed prematurely.

Fixed: Numeric strings with special formatting were not preserved when returned from a function via a variable or passed to a recursive function.

Fixed: If the final result of an expression (other than for Return) is an object, correctly yield an empty string instead of silently aborting the thread.

Fixed: __Delete meta-functions sometimes overwrote temporary values in the deref buffer.

Added: An address retrieved via &object may be cast back into an object reference by using Object(address).

Revision 32 - September 26, 2009

Fixed: Send/PostMessage crash when less than three parameters were specified.

Revision 31 - September 26, 2009

Added: Object/array support and numerous minor changes to make this possible.

Added: Support for While(expression) with no delimiting space.

Added: Trim, LTrim, RTrim.

Added: A ~= B; equivalent to RegExMatch(A, B). May be removed in a future revision.

Fixed: An incompatibility with LowLevel.

Changed: Characters [,] and ? are no longer valid in variable names. Consequently, ? (ternary) no longer requires a space on either side.

Changed: Optional parameters may now be omitted at any position in the parameter list of a non-dynamic function call. Since this works by automatically inserting the parameter's default value at load-time, it is not supported or allowed for dynamic function-calls.

Debugger: Various minor changes to make program flow easier to follow while stepping through code.

Optimization: If DllCall's first parameter is a literal string which identifies a function already present in memory, it is replaced with the actual address of the function.

Updated from v1.0.48.03 to v1.0.48.04.

Revision 30 - May 31, 2009

Updated from v1.0.48.02 to v1.0.48.03.

Revision 29 - May 2, 2009

All supported image formats may now be used as menu icons. Currently it is necessary to specify "actual size" when setting the icon to preserve transparency on Windows Vista and later. For example:

```
Menu, Icon, MenuItemName, Filename.png,, 0
```

Revision 28 - May 2, 2009

Improved average-case performance of dynamic function calls by implementing binary search for function name look-ups. This change also applies to other areas, such as load-time resolution of function references in expressions and run-time resolution by OnMessage, RegisterCallback, etc.

Revision 27 - April 26, 2009

Updated from v1.0.48.00 to v1.0.48.02.

Revision 26 - April 11, 2009

Fixed: Menu icons were not drawn on items which have sub-menus if owner-drawing was in use.

Fixed: Menu icons were not freed if all menu items were deleted at once.

Changed (Source): Renamed AutoHotkey.sln to AutoHotkey_L.sln to allow VS window to be identified more easily.

Revision 25 - March 30, 2009

Fixed: Send {U+xxxx} correctly sets modifier key-state as with any normal key.

Fixed: Send {U+..xx} no longer triggers hotkey SCxx.

Revision 24 - March 30, 2009

Added: Support for Send {U+xxxx}. SendInput() is used where available for proper unicode support, otherwise falls back to Alt+Numpad.

Revision 23 - March 30, 2009

Fixed: GuiControl, Enable/Disable now only resets focus if the focused control was disabled.

Revision 22 - March 26, 2009

Optimized If var [not] in/contains MatchList, especially for long lists containing shorter fields.

Revision 21 - March 16, 2009

Minor optimizations to While. [thanks Chris]

Fixed (Debugger): Revisions 19 and 20 omitted timer threads from the call stack.

Fixed (Debugger): XML-reserved characters were not escaped in most situations.

Fixed (Debugger): Incorrect command and transaction_id were used in the final response before exiting the script.

Revision 20 - February 27, 2009

Changed: If an icon cannot be loaded with the new method, fall back to the old method.

Revision 19 - February 26, 2009

Added: Gui, Font, qN option to set font quality/control anti-aliasing, where N is between 0 and 5.

Fixed: IL_Add now loads icons at the most appropriate size when multiple sizes exist.

Merged with AutoHotkey v1.0.48, introducing bug fixes, optimizations and other changes by Chris. Some AutoHotkey_L features were integrated into the mainstream release; changes to these features since previous revisions are as follows:

- Legacy behaviour is no longer applied to While - e.g. "While %x%" is correctly interpreted as a double-deref rather than as "While x".
- One-True-Brace style is now supported for While.
- Static vars may now be declared and initialized in assume-static functions.
- Passing too *few* parameters in a dynamic function call is no longer allowed, as it allows the function's caller to second-guess the function's designer, reduces the ability to detect script bugs at runtime, and may cause the application to crash if calling a built-in function.

Revision 18 - February 21, 2009

Fixed: Incomplete bug-fix in previous revision causing concat followed by assignment to skip the rest of the expression in some cases.

Revision 17 - February 20, 2009

Added: Menu icons via Icon and NoIcon sub-commands of the Menu command.

Changed: Negative icon numbers can now be used to indicate a resource ID within an executable.

Changed: Set default style of menus to include MNS_CHECKORBMP.

Changed: Load both small and large versions of custom tray icons to allow the correct icons to be shown in the appropriate places.

Fixed: Loading of icons from executable files at sizes other than the system small/large.

Revision 16 - February 4, 2009

Fixed: Deleting a menu item partially deleted its associated sub-menu.

Revision 15 - February 1, 2009

Fixed: SetFormat's presence in a *compiled* script was not disabling binary number write-caching. SetFormat should now work correctly in compiled scripts.

Revision 14 - February 1, 2009

Numerous bug-fixes and optimizations by Chris (pre-v1.0.48 beta).

Added: Support for regex callouts and the auto-callout option 'C'.

Changed: A_AhkVersion now includes the revision number. For instance, this is 1.0.47.07.L14.

Fixed: HWND's/window ID's were sign-extended to 64-bit. For instance, 0xb482057e became 0xffffffffb482057e.

Revision 13 - November 29, 2008

Fixed: Invalid DllCall types were treated as INT in some build environments.

Added: Workaround for GuiEscape issue when disabling focused control.

Changed version info: InternalName, ProductName and FileDescription are now "AutoHotkey_L" instead of "AutoHotkey".

Revision 12 - September 6, 2008

Fixed: Potential access violation when using `property_get` on a built-in variable.
-- DBGP

Revision 11 - September 5, 2008

Fixed: Remove checksum from AutoHotkeySC.bin via post-build script (AutoHotkey must be installed).

Added: .L suffix to A_AhkVersion.

Added: Preliminary support for DBGp (interactive debugging).

Revision 10 - August 2, 2008

Fixed: WheelLeft/WheelRight support, which was broken in revision 9.

Revision 9 - July 29, 2008

Fixed: Allow Break/Continue in While.

Fixed: TrayTip in non-English Windows and possibly other side-effects of using Vista headers.

Revision 8 - July 27, 2008

Added: #IfTimeout directive to set the timeout for evaluation of #If expressions, in milliseconds. Default is 1000.

Added: Assume-static mode for functions. "Static" must precede any local/global variable declarations.

Added: One-true-brace support for While.

Changed: While now sets A_Index to the iteration about to begin.

Revision 7 - July 26, 2008

Added: IsFunc(FuncName) - Returns a non-zero number if FuncName exists in the script or as a built-in function.

Revision 5 - July 19, 2008

Cleaned up obsolete references to test scripts.

Added basic default test script.

Revision 4 - July 18, 2008

Added: #if (expression) - Similar to #IfWinActive, but for arbitrary expressions.

Added: WheelLeft, WheelRight - Support for WM_MOUSEHWHEEL, which was introduced with Windows Vista. (Requires Vista.)

Added: While, expression - Loop while a condition is true.

Added: A_IsPaused - True if the underlying thread is paused.

Added: A_IsCritical - True if the current thread has been marked uninterruptible by the "Critical" command.

Changed: Allow any number of parameters to be passed in dynamic function calls.

Fixed: Access Violation caused by WinGetClass and subclassed windows.

Fixed: Access Violation caused by empty dynamic function references.

AutoHotkey Script Showcase

NiftyWindows -- by Enovatic-Solutions: This script gives you easy control of all basic window interactions such as dragging, resizing, maximizing, minimizing and closing. Its most powerful feature is activated by dragging with the right mouse button. Visualize each window divided into a virtual 9-cell grid with three columns and rows. The center cell is the largest one: you can grab and move a window around by clicking and holding it with the right mouse button. The other eight cells are used to resize a window in the same manner. NiftyWindows also offers snap-to-grid, "keep window aspect ratio", rolling up a window to its title bar, transparency control, and other useful shortcuts.

Screen Magnifier -- by Holomind: This screen magnifier has the several advantages over the one included with the operating system, including: Customizable refresh interval and zoom level (including shrink/de-magnify); antialiasing to provide nicer output; and it is open-source (as a result, there are several variations to choose from, or you can tweak the script yourself).

LiveWindows: Watch Dialog-boxes in Thumbnail -- by Holomind: This script allows you to monitor the progress of downloads, file-copying, and other dialogs by displaying a small replica of each dialog and its progress bar (dialogs are automatically detected, even if they're behind other windows). The preview window stays always-on-top but uses very little screen space (it can also be resized by dragging its edges). You can also monitor any window by dragging a selection rectangle around the area of interest (with control-shift-drag), then press `Win` `W` display that section in the preview window with real-time updates.

Mouse Gestures -- by deguix: This script watches how you move the mouse whenever the right mouse button is being held down. If it sees you "draw" a recognized shape or symbol, it will launch a program or perform another custom action of your choice (just like hotkeys). See the included README file for how to define gestures.

Context Sensitive Help in Any Editor -- by Rajat: This script makes `Ctrl` `2` (another hotkey of your choice) show the help file page for the selected AutoHotkey command or keyword. If nothing is selected, the command name

will be extracted from the beginning of the 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 [redacted] mouse button while clicking, then drag the window to a new position.

[Easy Window Dragging -- KDE style \(requires XP/2k/NT\) -- by Jonny](#): This script makes it much easier to move or resize a window: 1) Hold down the

Alt [redacted] and LEFT-click anywhere inside a window to drag it to a new location; 2) Hold down **Alt** [redacted] and RIGHT-click-drag anywhere inside a window to easily resize it; 3) Press **Alt** [redacted] but before releasing it the second time, left-click to minimize the window under the mouse cursor, right-click to maximize it, or middle-click to close it.

[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.

[IntelliSense -- by Rajat \(requires XP/2k/NT\)](#): This script watches while you edit an AutoHotkey script. When it sees you type a command followed by a comma or space, it displays that command's parameter list to guide you. In addition, you can press **Ctrl** [redacted] **F1** [redacted] (another hotkey of your choice) to display that

command's page in the help file. To dismiss the parameter list, press **Escape** [redacted] or **Enter** [redacted]

[Using a Joystick as a Mouse](#): This script converts a joystick into a three-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 helps determine the button numbers and other attributes 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 unhidden 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 unhidden 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  at 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.

WinLIRC Client: This script receives notifications from [WinLIRC](#) whenever you press a button on your remote control. It can be used to automate Winamp, Windows Media Player, etc. It's easy to configure. For example, if WinLIRC recognizes a button named "VolUp" on your remote control, create a label named VolUp and beneath it use the command `SoundSet +5` to increase the soundcard's volume by 5%.

1 Hour Software -- by skrommel: This is a large collection of useful scripts, professionally presented with short descriptions and screenshots.

Toralf's Scripts: This collection includes useful scripts such as:

- 1) AHK Window Info: Reveals information on windows, controls, etc.
- 2) Electronic Program Guide: Browses the TV programs/schedules of your region (supports several countries).
- 3) Auto-Syntax-Tidy: Changes indentation and case of commands in a script to give it a consistent format/style.

Sean's Scripts: Includes useful scripts such as:

- 1) [Network Download/Upload Meter](#): Displays the network download/upload KB in a small, always-on-top progress bar.
- 2) [StdoutToVar](#): Redirects the output of a command or application into one of the

script's variables.

3) [Capture a Screen Rectangle](#): A callable function that captures a portion of the screen and saves it as a file (BMP/JPG/PNG/GIF/TIF). It can also capture transparent windows and the mouse cursor.

4) [Color Zoomer/Picker](#): Magnifies the area near the cursor, allowing a single pixel to be selected and its color identified.

[SKAN's Tips N Tricks](#): Contains sample code and techniques for achieving useful effects and often-requested capabilities.

[Scripts and Functions Forum \(Current\)](#): This is a searchable collection of many ready-to-run scripts and functions. Built and maintained by AutoHotkey users, this archive grows and improves daily.

[Scripts and Functions Forum \(Archive\)](#): This is an archive of an older forum containing many more scripts, but some scripts might not run as-is on AutoHotkey v1.1.

[-- Home --](#)

Script Compatibility

Although many scripts written for AutoHotkey 1.0 do not require changes to run on AutoHotkey 1.1, some may function incorrectly due to necessary differences between the two versions. As the most problematic differences only affect advanced functionality like DllCall, most users do not need to be concerned.

AutoHotkey 1.1 is also known as "AutoHotkey_L", while AutoHotkey 1.0 was retrospectively labelled "AutoHotkey Basic". Some older versions of AutoHotkey_L used 1.0.* version numbers, so for clarity, this document refers to the two branches of AutoHotkey by name rather than version number.

Note: Some of the most common problems are caused by changes required to support Unicode text, and can be avoided by simply using the ANSI version of AutoHotkey_L.

Basic

High impact:

- Certain syntax errors are no longer tolerated
- FileRead may return corrupt binary data
- Variable and function names do not allow [,] or ?
- DPI scaling is enabled by default for GUIs

Medium impact:

- Transform's *Unicode* sub-command is unavailable on Unicode versions
- AutoHotkey.ahk is launched instead of AutoHotkey.ini
- SetFormat, Integer, **H** is case-sensitive
- A_LastError is modified by more commands
- MsgBox's handles commas more consistently
- Gui +Owner overrides additional styles
- SoundSet and SoundGet work better on Vista and later
- ~Tilde affects how custom modifier keys work
- `x & y::` causes both `x::` and `x up::` to fire when x is released

Low impact:

- If *var* is *type* ignores the system locale by default
- GroupActivate sets ErrorLevel and GroupAdd's *Label* works differently
- Run and RunWait interpret *Target* differently
- Control-Z is not interpreted as end-of-file
- Compatibility mode may cause confusion
- *A_IsCompiled* is always read-only
- Leading and trailing ``t` sequences are no longer discarded

Advanced

- Unicode vs ANSI
 - VarSetCapacity
 - DllCall
 - NumPut / NumGet
- Pointer Size

Basic

Syntax Errors

Certain syntax errors which were tolerated by AutoHotkey Basic are not tolerated by AutoHotkey_L. Most such errors can be easily corrected once they are identified. The following errors are detected immediately upon launching a script in AutoHotkey_L, and must be corrected for the script to run:

- A space, tab or comma is required between each command and its parameters. For example, `MsgBox< foo` and `If!foo` are not tolerated.
- `Hotkey, IfSomething`, where *Something* is invalid, is not tolerated.

Some other syntax errors are detected while the script is running. These cause an error message to be displayed prior to exiting the current thread:

- **Common:** Unrecognized or badly formatted `Gui`, `Gui Show` or `GuiControl` options.
- `GroupAdd` with a blank group name. Previously this caused the thread to *silently* exit.
- `Gui` option `+LastFoundExist` must not be combined with another option, since that would cause it to act the same as `+LastFound`.

Some other syntax errors are currently not detected, but cause problems with AutoHotkey_L:

- `Auto-concat` with `(` is more selective, so some invalid expressions like `12(34)` no longer work.

FileRead

`FileRead` translates text between code pages in certain common cases and therefore might output corrupt binary data. To avoid this, add the `*C` option or use `FileOpen` instead.

Variable and Function Names

The characters `[,]` and `?` are reserved for use in [expressions](#), so are no longer valid in variable names. Consequently, `?` (used in [ternary operations](#)) no longer requires a space on either side. See also [object syntax](#).

Errors may or may not be detected automatically:

- If a script used these characters in variable names in expressions, generally the script will run without displaying an error message, but will misbehave as the characters will be interpreted as operators rather than as part of a variable name.
- If these characters are used in a [double-deref](#) (such as `Array%n%` where `n` contains one of the above characters), an error message is displayed when the double-deref is evaluated, while the script is running.
- If these characters are used in other contexts, such as on the left hand side of an assignment, in the name of a command's input/output variable or between `%percent%` signs, an error message is displayed and the script is prevented from launching.

DPI Scaling

[DPI scaling](#) is enabled by default for script GUIs to ensure they scale according to the [system DPI setting](#). If enabled and the system DPI setting is not 96 (100%), positions and sizes accepted by or returned from Gui commands are not compatible with other commands. To disable DPI scaling, use `Gui -DPIScale`.

Transform

Some *Transform* sub-commands are altered or unavailable in Unicode versions of AutoHotkey_L:

- [Transform, Unicode](#) is unavailable. To assign Unicode text to the clipboard, use a regular assignment. See also: [StrPut/StrGet](#).
- [Transform, HTML](#) supports additional features.

Default Script

When AutoHotkey_L is launched without specifying a script, an `.ahk` file is

loaded by default instead of an .ini file. The name of this file depends on the filename of the current executable. For more details, see [Passing Command Line Parameters to a Script](#).

SetFormat, Integer[Fast], H

When an uppercase H is used, hexadecimal digits A-F will also be in uppercase. AutoHotkey Basic always uses lowercase digits. See [SetFormat](#).

A_LastError

The following commands now set [A_LastError](#) to assist with debugging: FileAppend, FileRead, FileReadLine, FileDelete, FileCopy, FileMove, FileGetAttrib/Time/Size/Version, FileSetAttrib/Time, FileCreateDir, RegRead, RegWrite, RegDelete. Using any of these commands causes the previous value of A_LastError to be overwritten.

MsgBox

[MsgBox](#)'s smart comma handling has been changed to improve flexibility and consistency with all other commands. In most cases, MsgBox will just work as intended. In some rare cases, scripts relying on the old quirky behaviour may observe a change in behaviour. For instance:

```
; This is now interpreted as an expression
(Options) followed by text (Title)
; instead of as a single expression (Text) with
multiple sub-expressions:
MsgBox % x, y
; Parentheses can be added to force the old
interpretation:
MsgBox % (x, y)

; This now shows an empty dialog instead of the
text "0, Title":
MsgBox 0, Title
; These behave as expected in both AutoHotkey_L
```

```
and AutoHotkey Basic:
MsgBox 0, Title, % ""      ; Shows an empty dialog
MsgBox 0`, Title         ; Shows the text "0,
Title"

; This now shows an empty dialog instead of the
text ", Title":
MsgBox,, Title
```

Gui +Owner

Applying the `+Owner` option to a Gui also removes the `WS_CHILD` style and sets the `WS_POPUP` style. This may break scripts which used `+Owner` to set the parent window of a Gui *after* setting the styles.

Sound Commands on Windows Vista and later

`SoundSet`, `SoundGet`, `SoundSetWaveVolume` and `SoundGetWaveVolume` have improved support for Windows Vista and later. Typical changes in behaviour include:

- Scripts affecting the whole system (as is usually intended) instead of just the script itself.
- Devices being numbered differently - each output or input is considered a separate device.

~Tilde and Custom Combination Hotkeys

As of [v1.1.14], the `tilde prefix` affects how a key works when used as a modifier key in a custom combination.

Custom Combinations and Down/Up Hotkeys

Except when the tilde prefix is used, if both a key-down and a key-up hotkey are defined for a custom modifier key, they will both fire when the key is released. For example, `x & y::` causes both `x::` and `x up::` to fire when x is released, where previously `x::` never fired.

If var is type

If *var is type* ignores the system locale unless [StringCaseSense](#), [Locale](#) has been used.

Window Groups

[GroupActivate](#) sets `ErrorLevel` to 1 if no window was found to activate or 0 otherwise. Previously, `ErrorLevel` was left unchanged.

[GroupAdd](#)'s *Label* parameter applies to the window group as a whole instead of to one particular window specification within the group. A discussion of this change can be found [on the forums](#). However, using this parameter is **not recommended**; check `ErrorLevel` after calling [GroupActivate](#) instead.

Run / RunWait

[AutoHotkey_L](#) includes some enhancements to the way the [Run](#) and [RunWait](#) commands interpret the *Target* parameter. This allows some things that didn't work previously, but in some very rare cases, may also affect scripts which were already working in AutoHotkey Basic. The new behaviour is as follows:

- If *Target* begins with a quotation mark, everything up to the next quotation mark is considered the action, typically an executable file.
- Otherwise the first substring which ends at a space and is either an existing file or ends in `.exe`, `.bat`, `.com`, `.cmd` or `.hta` is considered the action. This allows file types such as `.ahk`, `.vbs` or `.lnk` to accept parameters while still allowing "known" executables such as `wordpad.exe` to be launched without an absolute path as in previous versions.

Control-Z

[Loop Read](#) and [FileReadLine](#) no longer interpret the character  (ASCII 26) as an end-of-file marker. Any  (ASCII 26) appearing at the very end of the file, is loaded as-is. [FileRead](#) already ignored this character, so is not affected by this issue.

Compatibility Mode

If [Compatibility mode](#) is set to Windows 95, 98/ME or NT4 in the properties of the EXE file used to run the script, the script may not behave correctly. This is because compatibility mode causes a specific version of Windows to be reported to the application, but AutoHotkey_L omits support for these versions. For example, setting compatibility mode to Windows 95 or 98/ME will cause `MsgBox %A_OSVersion%` to report `WIN_NT4`.

A_IsCompiled

[A_IsCompiled](#) is defined as an empty string if the script has not been compiled. Previously it was left undefined, which meant that assignments such as `A_IsCompiled := 1` were valid if the script hadn't been compiled. Now it is treated as a read-only built-in variable in all cases.

Escaped Whitespace

Escaped whitespace characters such as ``t` and ``n` are no longer trimmed from the beginning and end of each arg. For example, `StringReplace s, s, `t` is now valid and will remove all tab characters from `s`.

Unicode vs ANSI

A text value is often referred to as a *string*, as each text value is stored as a sequence or *string* of characters. The numeric character code and size (in bytes) of each character depends on which version of AutoHotkey you are using: *Unicode* or *ANSI*. These details are typically important for scripts which do any of the following:

- Pass strings to external functions via [DllCall](#).
- Pass strings via [PostMessage/SendMessage](#).
- Manipulate strings directly via [NumPut/NumGet](#).
- Use [VarSetCapacity](#) to ensure a variable can hold a specific number of characters.

Scripts designed with one particular format in mind will often encounter problems when run on the wrong version of AutoHotkey. For instance, some scripts written for AutoHotkey Basic will function correctly on the ANSI version of AutoHotkey_L but fail on Unicode versions. If you aren't sure which version you are using, run the following script:

```
MsgBox % A_IsUnicode ? "Unicode" : "ANSI"
```

ANSI: Each character is **one byte** (8 bits). Character codes above 127 depend on your system's language settings.

Unicode: Each character is **two bytes** (16 bits). Character codes are as defined by the [UTF-16](#) format.

Semantic note: Technically, some Unicode characters are represented by *two* 16-bit code units, collectively known as a "surrogate pair." Similarly, some [ANSI code pages](#) (commonly known as [Double Byte Character Sets](#)) contain some double-byte characters. However, for practical reasons these are almost always treated as two individual units (referred to as "characters" for simplicity).

VarSetCapacity

VarSetCapacity sets the capacity of a var *in bytes*. To set a variable's capacity based on the number of characters, the size of a character must be taken into account. For example:

```

VarSetCapacity(ansi_var,    capacity_in_chars)
VarSetCapacity(unicode_var, capacity_in_chars *
2)
VarSetCapacity(native_var,  capacity_in_chars *
(A_IsUnicode ? 2 : 1))
VarSetCapacity(native_var,
t_size(capacity_in_chars)) ; see below

```

There are two main uses for VarSetCapacity:

1. Expand a variable to hold an estimated number of characters, to enhance performance when building a string by means of gradual concatenation. For example, `VarSetCapacity(var, 1000)` allows for 1000 bytes, which is only 500 characters on Unicode versions of AutoHotkey_L. This could affect performance, but the script should still function correctly.
2. Resize a variable to hold a binary structure. If the structure directly contains text, the format of that text must be taken into account. This depends on the structure - sometimes ANSI text will be used even in a Unicode version of AutoHotkey_L. If the variable is too small, the script may crash or otherwise behave unpredictably (depending on how the structure is used).

DllCall

When the "Str" type is used, it means a string in the native format of the current build. Since some functions may require or return strings in a particular format, the following string types are available:

	Char Size	C / Win32 Types	Encoding
WStr	16-bit	wchar_t*, WCHAR*, LPWSTR, LPCWSTR	UTF-16
		char*, CHAR*,	ANSI (the system default ANSI

AStr	8-bit	LPSTR, LPCSTR	code page)
Str	--	TCHAR*, LPTSTR, LPCTSTR	Equivalent to WStr in Unicode builds and AStr in ANSI builds.

If "Str" or the equivalent type for the current build is used as a parameter, the address of the string or var is passed to the function, otherwise a temporary copy of the string is created in the desired format and passed instead. As a general rule, "AStr" and "WStr" should not be used if the function writes a value into that parameter.

Note: "AStr" and "WStr" are equally valid for parameters and the function's return value.

In general, if a script calls a function via DllCall which accepts a string as a parameter, one of the following approaches must be taken:

1. If both Unicode (W) and ANSI (A) versions of the function are available, call the appropriate one for the current build. In the following example, "DeleteFile" is internally known as "DeleteFileA" or "DeleteFileW". Since "DeleteFile" itself doesn't really exist, DllCall automatically tries "A" or "W" as appropriate for the current build:

```
DllCall("DeleteFile", "Ptr", &filename)
DllCall("DeleteFile", "Str", filename)
```

In this example, `&filename` passes the address of the string exactly as-is, so the function must expect a string in the same format as the "Str" type. Note that "UInt" must be used in place of "Ptr" in AutoHotkey Basic, but the resulting code may not be 64-bit compatible.

Note: If the function cannot be found exactly as specified, AutoHotkey_L appends the "A" or "W" suffix regardless of which DLL is specified. However, AutoHotkey Basic appends the "A" suffix only for functions in User32.dll, Kernel32.dll, ComCtl32.dll, or Gdi32.dll.

2. If the function only accepts a specific type of string as input, the script may use the appropriate string type:

```
DllCall("DeleteFileA", "AStr", filename)
DllCall("DeleteFileW", "WStr", filename)
```

3. If the function must modify a string (in a non-native format), the script must allocate a buffer as described [above](#) and pass its address to the function. If the parameter accepts input, the script must also convert the input string to the appropriate format; this can be done using [StrPut](#).

NumPut / NumGet

When NumPut or NumGet are used with strings, the offset and type must be correct for the given type of string. The following may be used as a guide:

```
; 8-bit/ANSI strings: size_of_char=1
type_of_char="Char"
; 16-bit/UTF-16 strings: size_of_char=2
type_of_char="UShort"
nth_char := NumGet(var, (n-1)*size_of_char,
type_of_char)
NumPut(nth_char, var, (n-1)*size_of_char,
type_of_char)
```

If `var` contains a string in the native format, the appropriate values may be determined based on the value of `A_IsUnicode`:

```
nth_char := NumGet(var, t_size(n-1), t_char())
NumPut(nth_char, var, t_size(n-1), t_char())

; Define functions for convenience and clarity:
t_char() {
    return A_IsUnicode ? "UShort" : "Char"
}
t_size(char_count=1) {
```

```
    return A_IsUnicode ? char_count*2 :  
    char_count  
}
```

Pointer Size

Pointers are 4 bytes in 32-bit builds (including AutoHotkey Basic) and 8 bytes in 64-bit builds. Scripts using structures or DllCalls may need to account for this to run correctly on both platforms. Specific areas which are affected include:

- Offset calculation for fields in structures which contain one or more pointers.
- Size calculation for structures containing one or more pointers.
- Type names used with [DllCall](#), [NumPut](#) or [NumGet](#).

For size and offset calculations, use [A_PtrSize](#). For DllCall, NumPut and NumGet, use the [Ptr](#) type where appropriate.

Remember that the offset of a field is usually the total size of all fields preceding it. Also note that handles (including types like HWND and HBITMAP) are essentially pointer-types.

```
/*
    typedef struct _PROCESS_INFORMATION {
        HANDLE hProcess;    // Ptr
        HANDLE hThread;
        DWORD  dwProcessId; // UInt (4 bytes)
        DWORD  dwThreadId;
    } PROCESS_INFORMATION, *LPPROCESS_INFORMATION;
*/
VarSetCapacity(pi, A_PtrSize*2 + 8) ; Ptr + Ptr
+ UInt + UInt
DllCall("CreateProcess", <omitted for brevity>,
"Ptr", &pi, <omitted>)
hProcess      := NumGet(pi, 0)           ; Defaults
to "Ptr".
hThread       := NumGet(pi, A_PtrSize) ;
dwProcessId   := NumGet(pi, A_PtrSize*2,
"UInt")
dwProcessId   := NumGet(pi, A_PtrSize*2 + 4,
```

```
"UInt")
```

Alphabetical Command and Function Index

Click on a command or function name for details. Entries in **large font** are the most commonly used.

{ ... } (Block)	A pair of braces denotes a block. Blocks are typically used with functions , Else , Loop , While-loop , and IF-commands.
{ ... } / Object()	Creates a scriptable associative array.
[...] / Array()	Creates a scriptable associative array with integer keys.
Abs()	Returns the absolute value of <i>Number</i> .
AutoTrim	Determines whether traditional assignments like <code>Var1 = %Var2%</code> omit spaces and tabs from the beginning and end of Var2.
Asc()	Returns the numeric value of the first byte or UTF-16 code unit in the specified string.
ASin()	Returns the arcsine (the number whose sine is <i>Number</i>) in radians.
ACos()	Returns the arccosine (the number whose cosine is <i>Number</i>) in radians.
ATan()	Returns the arctangent (the number whose tangent is <i>Number</i>) in radians.
BlockInput	Disables or enables the user's ability to interact with the computer via keyboard and mouse.
Break	Exits (terminates) a loop . Valid inside any kind of loop .

Catch	Specifies the code to execute if an exception is raised during execution of a try statement.
Ceil()	Returns <i>Number</i> rounded up to the nearest integer (without any .00 suffix).
Chr()	Returns the string (usually a single character) corresponding to the character code indicated by the specified number.
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.
ClipWait	Waits until the clipboard contains data.
ComObjActive()	Retrieves a registered COM object.
ComObjArray()	Creates a SAFEARRAY for use with COM.
ComObjConnect()	Connects a COM object's event sources to functions with a given prefix.
ComObjCreate()	Creates a COM object.
ComObject()	Creates an object representing a typed value to be passed as a parameter or return value.
ComObjEnwrap() / ComObjUnwrap()	Wraps/unwraps a COM object.
ComObjError()	Enables or disables notification of COM errors.
ComObjFlags()	Retrieves or changes flags which control a COM wrapper object's behaviour.
ComObjGet()	Returns a reference to an object provided by a COM component.
ComObjMissing()	Creates a "missing parameter" object to pass to a COM method.

ComObjParameter()	Wraps a value and type to pass as a parameter to a COM method.
ComObjQuery()	Queries a COM object for an interface or service.
ComObjType()	Retrieves type information from a COM object.
ComObjValue()	Retrieves the value or pointer stored in a COM wrapper object.
Continue	Skips the rest of the current loop iteration and begins a new one. Valid inside any kind of 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 relative to either the active window or the screen.

Cos()	Returns the trigonometric cosine of <i>Number</i> .
Critical	Prevents the current thread from being interrupted by other threads, or enables it to be interrupted.
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.
DllCall()	Calls a function inside a DLL, such as a standard Windows API function.
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.
Edit	Opens the current script for editing in the associated editor.
Else	Specifies the command(s) to perform if an IF-statement evaluates to FALSE. When more than one command is present, enclose them in a block (braces).
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: <code>Var += Value</code> .
EnvDiv	Sets a variable to itself divided by the given value. Synonymous with: <code>Var /= Value</code> .
EnvGet	Retrieves an environment variable.

EnvMult	Sets a variable to itself times the given value. Synonymous with: <code>Var *= Value</code> .
EnvSet	Writes a value to a variable contained in the environment.
EnvSub	Sets a variable to itself minus the given value (can also compare date-time values). Synonymous with: <code>Var -= Value</code> .
EnvUpdate	Notifies the OS and all running applications that environment variable(s) have changed.
Exception()	Creates an object which can be used to throw a custom exception.
Exit	Exits the current thread or (if the script is not persistent and contains no hotkeys) the entire script.
ExitApp	Terminates the script unconditionally.
Exp()	Returns e (which is approximately 2.71828182845905) raised to the N th power.
FileAppend	Writes text to the end of a file (first creating the file, if necessary).
FileCopy	Copies one or more files.
FileCopyDir	Copies a folder along with all its sub-folders and files (similar to xcopy).
FileCreateDir	Creates a folder.
FileCreateShortcut	Creates a shortcut (.lnk) file.
FileDelete	Deletes one or more files.
FileEncoding	Sets the default encoding for FileRead , FileReadLine , Loop Read , FileAppend , and FileOpen . <i>See also: Text Encodings</i>

FileExist()	Checks for the existence of a file or folder and returns its attributes.
FileInstall	Includes the specified file inside the compiled version of the script.
FileGetAttrib	Reports whether a file or folder is read-only, hidden, etc.
FileGetShortcut	Retrieves information about a shortcut (.lnk) file, such as its target file.
FileGetSize	Retrieves the size of a file.
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 folder along with all its sub-folders and files. It can also rename a folder.
FileOpen	Provides object-oriented file I/O.
FileRead	Reads a file's contents into a variable .
FileReadLine	Reads the specified line from a file and stores the text in a variable .
FileRecycle	Sends a file or directory to the recycle bin, if possible.
FileRecycleEmpty	Empties the recycle bin.
FileRemoveDir	Deletes a folder.
FileSelectFile	Displays a standard dialog that allows the user to open or save 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.
Finally	Ensures that a block of code is always executed after a Try statement finishes.
Floor()	Returns <i>Number</i> rounded down to the nearest integer (without any .00 suffix).
For	Repeats a series of commands once for each key-value pair in an object.
Format	Formats a variable number of input values according to a format string.
FormatTime	Transforms a YYYYMMDDHH24MISS timestamp into the specified date/time format.
Func()	Retrieves a reference to a function.
GetKeyName()	Retrieves the name or text of a key.
GetKeyVK()	Retrieves the virtual key code of a key.
GetKeySC()	Retrieves the scan code of a key.
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.
Hotstring()	Creates, modifies, enables, or disables a hotstring while the script is running.
if	Specifies the command(s) to perform if the comparison of a variable to a value evaluates to TRUE. When more than one command is present, enclose them in a block (braces).
if (expression)	Specifies the command(s) to perform if an expression evaluates to TRUE.
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 folder.
IfGreater / IfGreaterOrEqual	Compares a variable to a value. Synonymous with: <code>if Var > Value</code> <code>if Var >= Value</code> .
IfInString / IfNotInString / InStr()	Checks if a variable contains the specified string.
IfLess / IfLessOrEqual	Compares a variable to a value. Synonymous with: <code>if Var < Value</code> <code>if Var <= Value</code> .
IfMsgBox	Checks which button was pushed by the user during the most recent MsgBox command.
IfWinActive / IfWinNotActive	Checks if the specified window exists and is currently active (foremost).
IfWinExist / IfWinNotExist	Checks if the specified window exists.
IL_Create() IL_Add() IL_Destroy()	The means by which icons are added to a ListView or TreeView control.
ImageSearch	Searches a region of the screen for an image.
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.
InputBox	Displays an input box to ask the user to enter a string.
InStr()	Searches for a given <i>occurrence</i> of a string, from the left or the right.
IsByRef()	Returns a non-zero number if a ByRef parameter of a function was supplied with the specified variable.
IsFunc()	Returns a non-zero number if the specified function exists in the script.
IsLabel()	Returns a non-zero number if the specified label exists in the script.
IsObject()	Returns a non-zero number if the specified value is an object.
KeyHistory	Displays script info and a history of the most recent keystrokes and mouse clicks.
KeyWait	Waits for a key or mouse/joystick button to be released or pressed down.
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.
LoadPicture()	Loads a picture from file and returns a bitmap or icon handle.
Log()	Returns the logarithm (base 10) of <i>Number</i> .
Ln()	Returns the natural logarithm (base e) of

	<i>Number.</i>
Loop (normal)	Perform a series of commands 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 at a time.
Loop (read file contents)	Retrieves the lines in a text file, one at a time (performs better than FileReadLine).
Loop (registry)	Retrieves the contents of the specified registry subkey, one item at a time.
LV_Add() LV_Delete() LV_DeleteCol() LV_GetCount() LV_GetNext() LV_GetText() LV_Insert() LV_InsertCol() LV_Modify() LV_ModifyCol() LV_SetImageList()	The means by which the rows and columns of a ListView control are added, modified or retrieved.
Max()	Returns the highest value of one or more numbers.
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.
MenuGetHandle	Retrieves the Win32 menu handle of a menu.
MenuGetName	Retrieves the name of a menu given a handle to its underlying Win32 menu .

Min()	Returns the lowest value of one or more numbers.
Mod()	Modulo. Returns the remainder when <i>Dividend</i> is divided by <i>Divisor</i> .
MouseClicked	Clicks or holds down a mouse button, or turns the mouse wheel. NOTE: The Click command is generally more flexible and easier to use.
MouseClickedDrag	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 the specified text in a small window containing one or more buttons (such as Yes and No).
NumGet()	Returns the binary number stored at the specified address+offset.
NumPut()	Stores a number in binary format at the specified address+offset.
ObjAddRef() / ObjRelease()	Increments or decrements an object's reference count .
ObjBindMethod()	Creates a BoundFunc object which calls a method of a given object.
ObjClone() ObjDelete() ObjGetAddress() ObjGetCapacity() ObjHasKey() ObjInsert() ObjInsertAt()	

ObjLength() ObjMaxIndex() ObjMinIndex() ObjNewEnum() ObjPop() ObjPush() ObjRemove() ObjRemoveAt() ObjSetCapacity()	<p>These functions are equivalent to built-in methods of the Object type. It is usually recommended to use the corresponding method instead.</p>
ObjRawSet()	Stores or overwrites a key-value pair in an object, bypassing the object's meta-functions .
OnClipboardChange()	Registers a function or function object to run whenever the clipboard's content changes.
OnExit	Specifies a subroutine to run automatically when the script exits.
OnMessage()	Specifies a function to call automatically when the script receives the specified message.
Ord()	Returns the ordinal value (numeric character code) of the first character in the specified string.
OutputDebug	Sends a string to the debugger (if any) for display.
Pause	Pauses the script's current thread .
PixelGetColor	Retrieves the color of the pixel at the specified x,y coordinates.
PixelSearch	Searches a region of the screen for a pixel of the specified color.
PostMessage	Places a message in the message queue of a window or control.
	Performs one of the following operations on a

Process	process: checks if it exists; changes its priority; closes it; waits for it to close.
Progress	Creates or updates a window containing a progress bar.
Random	Generates a pseudo-random number.
RegexMatch()	Determines whether a string contains a pattern (regular expression).
RegexReplace()	Replaces occurrences of a pattern (regular expression) inside a string.
RegDelete	Deletes a subkey or value from the registry.
RegRead	Reads a value from the registry.
RegWrite	Writes a value to the registry.
RegisterCallback()	Creates a machine-code address that when called, redirects the call to a function in the script.
Reload	Replaces the currently running instance of the script with a new one.
Return	Returns from a subroutine to which execution had previously jumped via function-call , Gosub , Hotkey activation, GroupActivate , or other means.
Round()	If <i>N</i> is omitted or 0, <i>Number</i> is rounded to the nearest integer. If <i>N</i> is positive number, <i>Number</i> is rounded to <i>N</i> decimal places. If <i>N</i> is negative, <i>Number</i> is rounded by <i>N</i> digits to the left of the decimal point.
Run	Runs an external program.
RunAs	Specifies a set of user credentials to use for all subsequent uses of Run and RunWait .
	Runs an external program and waits until it

RunWait	finishes.
SB_SetIcon() SB_SetParts() SB_SetText()	The means by which the bar of a StatusBar control is modified.
Send / SendRaw / SendInput / SendPlay / SendEvent	Sends simulated keystrokes and mouse clicks to the active window.
SendLevel	Controls which artificial keyboard and mouse events are ignored by hotkeys and hotstrings.
SendMessage	Sends a message to a window or control and waits for acknowledgement.
SendMode	Makes Send synonymous with SendInput or SendPlay rather than the default (SendEvent). Also makes Click and MouseMove/Click/Drag use the specified method.
SetBatchLines	Determines how fast a script will run (affects CPU utilization).
SetCapsLockState	Sets the state of the CapsLock  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 Click and MouseMove/Click/Drag .
SetEnv (Var = Value)	Assigns the specified value to a variable .
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 <code>NumLock</code>  also force the key to stay on or off.
SetScrollLockState	Sets the state of the <code>ScrollLock</code>  Can also force the key to stay on or off.
SetRegView	Allows registry commands in a 32-bit script to access the 64-bit registry view and vice versa.
SetStoreCapsLockMode	Whether to restore the state of <code>CapsLock</code>  after a Send .
SetTimer	Causes a subroutine to be launched automatically and repeatedly at a specified time interval.
SetTitleMatchMode	Sets the matching behavior of the <code>WinTitle</code> parameter in commands such as WinWait .
SetWinDelay	Sets the delay that will occur after each windowing command, such as WinActivate .
SetWorkingDir	Changes the script's current working directory.
Shutdown	Shuts down, restarts, or logs off the system.
Sin()	Returns the trigonometric sine of <i>Number</i> .
Sleep	Waits the specified amount of time before continuing.
Sort	Arranges a variable's contents in alphabetical, numerical, or random order (optionally removing duplicates).

SoundBeep	Emits a tone from the PC speaker.
SoundGet	Retrieves various settings from a sound device (master mute, master volume, etc.)
SoundGetWaveVolume	Retrieves the wave output volume from a sound device.
SoundPlay	Plays 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 updates a window containing a JPG, GIF, or BMP image.
SplashTextOn	Creates a customizable text popup window.
SplashTextOff	Closes the above window.
SplitPath	Separates a file name or URL into its name, directory, extension, and drive.
Sqrt()	Returns the square root of <i>Number</i> .
StatusBarGetText	Retrieves the text from a standard status bar control.
StatusBarWait	Waits until a window's status bar contains the specified string.
StrPut() / StrGet()	Copies a string to or from a memory address, optional converting it between code pages.
StringCaseSense	Determines whether string comparisons are case sensitive (default is "not case sensitive").
StringGetPos / InStr()	Retrieves the position of the specified substring within a string.

StringLeft	Retrieves a number of characters from the left-hand side of a string.
StringLen / StrLen()	Retrieves the count of how many characters are in a string.
StringLower	Converts a string to lowercase.
StringMid / SubStr()	Retrieves one or more characters from the specified position in a string.
StringReplace / StrReplace()	Replaces the specified substring with a new string.
StringRight	Retrieves a number of characters from the right-hand side of a string.
StringSplit / StrSplit()	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 or enables all or selected hotkeys and hotstrings .
SysGet	Retrieves screen resolution, multi-monitor info, dimensions of system objects, and other system properties.
Tan()	Returns the trigonometric tangent of <i>Number</i> .
Thread	Sets the priority or interruptibility of threads . It can also temporarily disable all timers .
	Signals the occurrence of an error. This signal

Throw	can be caught by a try-catch statement.
ToolTip	Creates an always-on-top window anywhere on the screen.
Transform	Performs miscellaneous math functions, bitwise operations, and tasks such as ASCII/Unicode conversion.
TrayTip	Creates a balloon message window or toast notification near the tray icon.
Trim() / LTrim() / RTrim()	Trims certain characters from the beginning and/or end of a string.
Try	Guards one or more statements (commands or expressions) against runtime errors and exceptions thrown by the throw command.
TV_Add() TV_Delete() TV_Get() TV_GetChild() TV_GetCount() TV_GetNext() TV_GetParent() TV_GetPrev() TV_GetSelection() TV_GetText() TV_Modify() TV_SetImageList()	The means by which the items of a TreeView control are added, modified or retrieved.
Until	Applies a condition to the continuation of a Loop or For-loop.
UrlDownloadToFile	Downloads a file from the Internet.
Var = value	Assigns the specified value to a variable .
Var := expression	Evaluates an expression and stores the result in a variable .

VarSetCapacity()	Enlarges a variable's holding capacity or frees its memory. Normally, this is necessary only for unusual circumstances such as DllCall .
While-loop	Performs a series of commands repeatedly until the specified expression evaluates to false.
WinActivate	Activates the specified window.
WinActivateBottom	Same as WinActivate except that it activates the bottommost matching window rather than the topmost.
WinActive()	Returns the Unique ID (HWND) of the active window if it matches the specified criteria.
WinClose	Closes the specified window.
WinExist()	Returns the Unique ID (HWND) of the first matching window.
WinGetActiveStats	Combines the functions of WinGetActiveTitle and WinGetPos into one command.
WinGetActiveTitle	Retrieves the title of the active window.
WinGetClass	Retrieves the specified window's class name.
WinGet	Retrieves the specified window's unique ID, process ID, process name, or a list of its controls. It can also retrieve a list of all windows matching the specified criteria.
WinGetPos	Retrieves the position and size of the specified window.
WinGetText	Retrieves the text from the specified window.
WinGetTitle	Retrieves the title of the specified window.
WinHide	Hides the specified window.

WinKill	Forces the specified window to close.
WinMaximize	Enlarges the specified window to its maximum size.
WinMenuSelectItem	Invokes a menu item from the menu bar of the specified window.
WinMinimize	Collapses the specified window into a button on the task bar.
WinMinimizeAll	Minimizes all windows.
WinMinimizeAllUndo	Reverses the effect of a previous WinMinimizeAll .
WinMove	Changes the position and/or size of the specified window.
WinRestore	Unminimizes or unmaximizes the specified window if it is minimized or maximized.
WinSet	Makes a variety of changes to the specified window, such as "always on top" and transparency.
WinSetTitle	Changes the title of the specified window.
WinShow	Unhides the specified window.
WinWait	Waits until the specified window exists.
WinWaitActive	Waits until the specified window is active.
WinWaitClose	Waits until the specified window does not exist.
WinWaitNotActive	Waits until the specified window is not active.
#ClipboardTimeout	Changes how long the script keeps trying to access the clipboard when the first attempt fails.
	Changes the script's comment symbol from

#CommentFlag	semicolon to some other string.
#Delimiter	Changes the script's command parameter delimiter from comma to some other character.
#DerefChar	Changes the script's dereference symbol from percent to some other character.
#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: backslash vs. accent).
#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: 
#Hotstring	Changes hotstring options or ending characters.
#If	Similar to #IfWinActive , but for arbitrary expressions.
#IfTimeout	Sets the maximum time that may be spent evaluating a single #If expression.
#IfWinActive / #IfWinNotActive / #IfWinExist / #IfWinNotExist	Creates context-sensitive hotkeys and hotstrings . Such hotkeys perform a different action (or none at all) depending on the type of window that is active or exists.
#Include / #IncludeAgain	Causes the script to behave as though the specified file's contents are present at this

	exact position.
<code>#InputLevel</code>	Controls which artificial keyboard and mouse events are ignored by hotkeys and hotstrings.
<code>#InstallKeybdHook</code>	Forces the unconditional installation of the keyboard hook.
<code>#InstallMouseHook</code>	Forces the unconditional installation of the mouse hook.
<code>#KeyHistory</code>	Sets the maximum number of keyboard and mouse events displayed by the <code>KeyHistory</code> window. You can set it to 0 to disable key history.
<code>#LTrim</code>	Enables or disables trimming of indentation in continuation sections.
<code>#MaxHotkeysPerInterval</code>	Along with <code>#HotkeyInterval</code> , specifies the rate of <code>hotkey</code> activations beyond which a warning dialog will be displayed.
<code>#MaxMem</code>	Sets the maximum capacity of each <code>variable</code> to the specified number of megabytes.
<code>#MaxThreads</code>	Sets the maximum number of simultaneous <code>threads</code> .
<code>#MaxThreadsBuffer</code>	Causes some or all <code>hotkeys</code> to buffer rather than ignore keypresses when their <code>#MaxThreadsPerHotkey</code> limit has been reached.
<code>#MaxThreadsPerHotkey</code>	Sets the maximum number of simultaneous <code>threads</code> per <code>hotkey</code> or <code>hotstring</code> .
<code>#MenuMaskKey</code>	Changes which key is used to mask <code>Win</code>  or <code>Alt</code>  events.
<code>#NoEnv</code>	Avoids checking empty variables to see if they are environment variables (recommended for all new scripts).

#NoTrayIcon	Disables the showing of a tray icon.
#Persistent	Keeps a script permanently running (that is, until the user closes it or ExitApp is encountered).
#SingleInstance	Determines whether a script is allowed to run again when it is already running.
#UseHook	Forces the use of the hook to implement all or some keyboard hotkeys .
#Warn	Enables or disables warnings for selected conditions that may be indicative of developer errors.
#WinActivateForce	Skips the gentle method of activating a window and goes straight to the forceful method.

ClipWait

Waits until the [clipboard](#) contains data.

```
ClipWait [, SecondsToWait, 1]
```

Parameters

SecondsToWait

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

1

If this parameter is omitted, the command is more selective, waiting specifically for text or files to appear ("text" includes anything that would produce text when you paste into Notepad). If this parameter is 1 (can be an [expression](#)), the command waits for data of any kind to appear on the clipboard.

ErrorLevel

If the wait period expires, [ErrorLevel](#) will be set to 1. Otherwise (i.e. the clipboard contains data), 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 considers anything convertible to text (e.g. HTML) to be text. It also considers files, such as those copied in an Explorer window via

 text. Such files are automatically converted to their

filenames (with full path) whenever the clipboard variable (%clipboard%) is referred to in the script. See [Clipboard](#) for details.

When 1 is present as the last parameter, the command will be satisfied when any data appears on the clipboard. This can be used in conjunction with [ClipboardAll](#) to save non-textual items such as pictures.

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
{
    MsgBox, The attempt to copy text onto the
clipboard failed.
    return
}
MsgBox, clipboard = %clipboard%
return
```

EnvGet [v1.0.43.08+]

Retrieves an environment variable.

```
EnvGet, OutputVar, EnvVarName
```

Parameters

OutputVar

The name of the variable in which to store the string.

EnvVarName

The name of the [environment variable](#) to retrieve. For example:
EnvGet, OutputVar, Path.

Remarks

If the specified environment variable is empty or does not exist, *OutputVar* is made blank.

The operating system limits each environment variable to 32 KB of text.

Related

[EnvSet](#), [#NoEnv](#), [environment variables](#), [EnvUpdate](#), [SetEnv](#), [Run](#), [RunWait](#)

Example

```
EnvGet, OutputVar, LogonServer
```

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

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

Remarks

The operating system limits each environment variable to 32 KB of text.

An environment variable created or changed with this command will be accessible only to programs the script launches via [Run](#) or [RunWait](#). See [environment variables](#) for more details.

This command exists separately from [SetEnv](#) because [normal script variables](#) are not stored in the environment. This is because performance would be worse and also because the OS limits environment variables to 32 KB.

Related

[EnvGet](#), [#NoEnv](#), [environment variables](#), [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

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[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, EnvUpdate could be used to broadcast the change:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\Environment
```

Related

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

Example

```
EnvUpdate
```

OnClipboardChange

OnClipboardChange() [v1.1.20+]

Registers a [function](#) or [function object](#) to run whenever the clipboard's content changes.

```
OnClipboardChange(Func [, AddRemove])
```

Parameters

Func

A function name or [function object](#) to call. The function's parameter and return value are described [below](#).

AddRemove

One of the following values:

- 1** (the default): Call the function after any previously registered functions.
- 1**: Call the function before any previously registered functions.
- 0**: Do not call the function.

If an OnClipboardChange label exists, it is always called first.

Func

```
FunctionName(Type)
```

Type

Contains one of the following values:

- 0** if the clipboard is now empty;
- 1** if it contains something that can be expressed as text (this includes [files copied](#) from an Explorer window);
- 2** if it contains something entirely non-text such as a picture.

Return Value

If this is the last or only OnClipboardChange function, the return value is ignored. Otherwise, the function can return a non-zero integer to prevent subsequent functions from being called.

Example

This example is equivalent to [the one above](#), except that the function is not called when the script first starts; only when the content of the Clipboard changes.

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

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

OnClipboardChange Label

Deprecated: This approach is not recommended for use in new scripts. Use the [OnClipboardChange](#) function instead.

A label named OnClipboardChange (if it exists) is launched automatically whenever any application (even the script itself) has changed the contents of the clipboard. The label also runs once when the script first starts.

The built-in variable A_EventInfo contains:

0 if the clipboard is now empty;

1 if it contains something that can be expressed as text (this includes [files copied](#) from an Explorer window);

2 if it contains something entirely non-text such as a picture.

The following example is a working script. Whenever it is running, it will briefly display a ToolTip for each clipboard change.

```
#Persistent
return

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

Remarks

If the clipboard changes while an OnClipboardChange function or label is already running, that notification event is lost. If this is undesirable, specify **Critical** as the label's first line. However, this will also buffer/defer other **threads** (such as the press of a hotkey) that occur while the OnClipboardChange thread is running.

If the script itself changes the clipboard, its OnClipboardChange function or label is typically not executed immediately; that is, commands immediately below the command that changed the clipboard are likely to execute beforehand. To force the function or label to execute immediately, use a short delay such as **Sleep 20** after changing the clipboard.

Related

[Clipboard](#), [OnExit](#), [OnMessage\(\)](#), [RegisterCallback\(\)](#)

SysGet

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.

MonitorPrimary: Retrieves the number of the primary monitor, which will be 1 in a single-monitor system.

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. For example:

```
SysGet, Mon2, Monitor, 2
MsgBox, Left: %Mon2Left% -- Top: %Mon2Top% --
Right: %Mon2Right% -- Bottom %Mon2Bottom%.
```

Within a [function](#), to create a set of variables that is global instead of local, [declare](#) Mon2 as a global variable prior to using this command (the converse is true for [assume-global](#) functions). However, it is often also necessary to declare each variable in the set, due to a [common source of confusion](#).

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").
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.
	SM_CXMIN, SM_CYMIN: Minimum width and height of a

28, 29	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.
19	SM_MOUSEPRESENT: Nonzero if a mouse is installed; zero otherwise.
75	SM_MOUSEWHEELPRESENT: Nonzero if a mouse with a wheel is installed; zero otherwise.
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.
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.

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: 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.

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.
	SM_CXDOUBLECLK, SM_CYDOUBLECLK: 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

36, 37	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_CXFUSBORDER, SM_CYFUSBORDER: 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: The retrieved value is always 0.
21, 3	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.
	SM_CXMENUCHECK, SM_CYMENUCHECK: Dimensions of

71, 72	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_CXMINSPPACING SM_CYMINSPPACING: 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.
30, 31	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_CYSMCACTION: 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.
22	SM_DEBUG: Nonzero if the debug version of User.exe is installed; zero otherwise.
82	<p>SM_IMMENABLED: Nonzero if Input Method Manager/Input Method Editor features are enabled; zero otherwise.</p> <p>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.</p>
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.
	SM_TABLETPC: Nonzero if the current operating system is the

Remarks

The built-in variables [A_ScreenWidth](#) and [A_ScreenHeight](#) contain the dimensions of the primary monitor, in pixels.

Related

[DllCall](#), [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% (%MonitorWorkAreaLeft%
work)`nTop: `t%MonitorTop% (%MonitorWorkAreaTop%
work)`nRight: `t%MonitorRight%
```

```
(%MonitorWorkAreaRight%  
work) `nBottom: `t%MonitorBottom%  
(%MonitorWorkAreaBottom% work)  
}
```

DllCall()

Calls a function inside a DLL, such as a standard Windows API function.

```
Result := DllCall("[DllFile\]Function" [, Type1, Arg1, Type2, Arg2, "Cdecl Return Type"])
```

Parameters

Result

DllCall returns the actual value returned by the function. If the function is of a type that does not return a value, the result is an undefined integer. If the function cannot be called due to an [error](#), the return value is blank (an empty string).

[DllFile\]Function

The DLL or EXE file name followed by a backslash and the name of the function. For example: `"MyDLL\MyFunction"` (the file extension ".dll" is the default when omitted). If an absolute path isn't specified, *DllFile* is assumed to be in the system's PATH or [A_WorkingDir](#).

DllFile may be omitted when calling a function that resides in User32.dll, Kernel32.dll, ComCtl32.dll, or Gdi32.dll. For example, `"User32\IsWindowVisible"` produces the same result as `"IsWindowVisible"`.

If no function can be found by the given name, an A (ANSI) or W (Unicode) suffix is automatically appended based on which version of AutoHotkey is running the script. For example, `"MessageBox"` is the same as `"MessageBoxA"` in ANSI versions and `"MessageBoxW"` in Unicode versions.

Performance can be dramatically improved when making *repeated* calls to a DLL by [loading it beforehand](#).

[v1.0.46.08+]: This parameter may also consist solely of an integer, which is interpreted as the address of the function to call. Sources of such addresses include [COM](#) and [RegisterCallback\(\)](#).

Type1, Arg1

Each of these pairs represents a single parameter to be passed to the function. The number of pairs is unlimited. For *Type*, see the [types table](#) below. For *Arg*, specify the value to be passed to the function.

Cdecl ReturnType

The word *Cdecl* is normally omitted because most functions use the standard calling convention rather than the "C" calling convention (functions such as `wprintf` that accept a varying number of arguments are one exception to this). If you omit *Cdecl* but the call yields [ErrorLevel An](#) -- where **n** is the total size of the arguments you passed -- *Cdecl* might be required. Note that most object-oriented C++ functions use the *thiscall* convention, which is not supported.

If present, the word *Cdecl* should be listed before the return type (if any). Separate each word from the next with a space or tab. For example:

```
"Cdecl Str" .
```

[AHK_L 53+]: Since a separate "C" calling convention does not exist in 64-bit code, *Cdecl* may be specified but has no effect on 64-bit builds of AutoHotkey.

ReturnType: If the function returns a 32-bit signed integer (Int), BOOL, or nothing at all, *ReturnType* may be omitted. Otherwise, specify one of the argument types from the [types table](#) below. The [asterisk suffix](#) is also supported.

Types of Arguments and Return Values

A string such as "Blue" or MyVar. If the called function modifies the string and the argument is a naked variable, its contents will be updated. For example, the following call would convert the contents of *MyVar* to uppercase: `DllCall("CharUpper",`

Str	<p><code>"Str", MyVar)</code>.</p> <p>However, if the function is designed to store a string larger than a variable's current capacity, ensure that the variable is large enough before calling the function. This can be achieved by calling <code>VarSetCapacity(MyVar, 123)</code>, where 123 is the length that <i>MyVar</i> must be able to hold.</p> <p>A <i>Str</i> argument must not be an expression that evaluates to a number (such as <code>i+1</code>). If it is, the function is not called and <code>ErrorLevel</code> is set to -2.</p> <p>The asterisk variable "Str*" is supported but rarely used. It can be used with functions that expect something like "TCHAR **" or "LPTSTR *".</p> <div style="background-color: #e0ffe0; padding: 10px; border: 1px solid #c0ffc0;"> <p>Note: When passing a string to a function, be aware what <i>type of string</i> the function expects.</p> </div>
AStr WStr	<p>[AHK_L 42+]: An ANSI or Unicode (Wide character) string. See Script Compatibility for equivalent Win32 types and other details.</p>
Int64	<p>A 64-bit integer, whose range is -9223372036854775808 (-0x8000000000000000) to 9223372036854775807 (0x7FFFFFFFFFFFFFFF).</p>
Int	<p>A 32-bit integer (the most common integer type), whose range is -2147483648 (-0x80000000) to 2147483647 (0x7FFFFFFF). An Int is sometimes called a "Long".</p> <p>An Int should also be used for each BOOL argument expected by a function (a BOOL value should be either 1 or 0).</p> <p>An unsigned Int (UInt) is also used quite frequently, such as for DWORD.</p>
	<p>A 16-bit integer, whose range is -32768 (-0x8000) to 32767 (0x7FFF). An unsigned Short (UShort) can be used with functions</p>

Short	that expect a WORD.
Char	An 8-bit integer, whose range is -128 (-0x80) to 127 (0x7F). An unsigned character (UChar) can be used with functions that expect a BYTE.
Float	A 32-bit floating point number, which provides 6 digits of precision.
Double	A 64-bit floating point number, which provides 15 digits of precision.
Ptr	<p>[AHK_L 42+]: A pointer-sized integer, equivalent to Int or Int64 depending on whether the exe running the script is 32-bit or 64-bit. <i>Ptr</i> should be used for pointers to arrays or structures (such as RECT* or LPPOINT) and almost all handles (such as HWND, HBRUSH or HBITMAP). If the parameter is a pointer to a single numeric value such as LPDWORD or int*, generally the * or P suffix should be used instead of "Ptr".</p> <p><i>Ptr</i> can also be used with the * or P suffix; it should be used with functions that output a pointer via LPVOID* or similar.</p> <p><i>UPtr</i> is also valid, but is only unsigned in 32-bit builds as AutoHotkey does not support unsigned 64-bit integers.</p> <p>If compatibility with older versions of AutoHotkey is required, use a variable type as shown below:</p> <pre>Ptr := A_PtrSize ? "Ptr" : "UInt" ; If A_PtrSize is not defined, use UInt instead. DllCall("DeleteFile", Ptr, &filename) ; Omit the quote marks around Ptr.</pre> <p>Note: To pass a NULL handle or pointer, pass the integer 0.</p>

<p>* or P (suffix)</p>	<p>Append an asterisk (with optional preceding space) to any of the above types to cause the address of the argument to be passed rather than the value itself (the called function must be designed to accept it). Since the value of such an argument might be modified by the function, whenever a naked variable is passed as the argument, that variable's contents will be updated. For example, the following call would pass the contents of MyVar to MyFunction by address, but would also update MyVar to reflect any changes made to it by MyFunction: <code>DllCall("MyDll\MyFunction", "Int*", MyVar)</code>.</p> <p>In general, an asterisk is used whenever a function has an argument type or return type that starts with "LP". The most common example is LPDWORD, which is a pointer to a DWORD. Since a DWORD is an unsigned 32-bit integer, use "UInt*" or "UIntP" to represent LPDWORD. An asterisk should not be used for string types such as LPTSTR, pointers to structures such as LPRECT, or arrays; for these, "Str" or "Ptr" should be used, depending on whether you pass a variable or its address.</p> <p>Note: "Char*" is not the same as "Str" because "Char*" passes the address of an 8-bit number, whereas "Str" passes the address of a series of characters which may be either 8-bit (ANSI) or 16-bit (Unicode), depending on the version of AutoHotkey. Similarly, "UInt*" passes the address of a 32-bit number, so should not be used if the function expects an array of values or a structure larger than 32 bits.</p> <p>Since variables in AutoHotkey have no fixed type, the address passed to the function points to temporary memory rather than the variable itself. It is not necessary to call <code>VarSetCapacity</code> on the variable as <code>DllCall</code> will update it correctly after the function returns.</p>

U (prefix)	<p>Prepend the letter U to any of the integer types above to interpret it as an unsigned integer (UInt64, UInt, UShort, and UChar). Strictly speaking, this is necessary only for return values and asterisk variables because it does not matter whether an argument passed by value is unsigned or signed (except for Int64).</p> <p>If a negative integer is specified for an unsigned argument, the integer wraps around into the unsigned domain. For example, when -1 is sent as a UInt, it would become 0xFFFFFFFF.</p> <p><i>Unsigned</i> 64-bit integers produced by a function are not supported. Therefore, to work with numbers greater or equal to 0x8000000000000000, omit the U prefix and interpret any negative values received from the function as large integers. For example, a function that yields -1 as an Int64 is really yielding 0xFFFFFFFFFFFFFFFF if it is designed to yield a UInt64.</p>
---------------	---

Note: When specifying an argument type or return type that does not contain a space or asterisk, the quotes around it may be omitted. For example, `Str` can be used in place of `"Str"` and `CDecl` in place of `"CDecl"`. In addition, the letter P may be used in place of asterisk to allow the quotes to be omitted there as well. For example: `UIntP`.

ErrorLevel

[v1.1.04+]: This function is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[ErrorLevel](#) is set to one of the following values to indicate whether the call succeeded or failed.

0: Success.

-1 (negative 1): The `[DllFile]Function` parameter is a floating point number. A string or positive integer is required.

-2: The [return type](#) or one of the specified [arg types](#) is invalid. This error can also

be caused by passing an [expression](#) that evaluates to a number to a string ([str](#)) argument.

-3: The specified *DllFile* could not be accessed or loaded. If no explicit path was specified for *DllFile*, the file must exist in the system's PATH or [A_WorkingDir](#). This error might also occur if the user lacks permission to access the file, or if AutoHotkey is 32-bit and the DLL is 64-bit or vice versa.

-4: The specified function could not be found inside the DLL.

N (any positive number): The function was called but it aborted with fatal exception number **N** (for example, 0xC0000005 means "access violation"). In such cases, the function returns a blank value (empty string), but any [asterisk variables](#) are still updated. An example of a fatal exception is dereferencing an invalid pointer such as NULL. Since a [Cdecl](#) function never produces the "*An*" error in the next paragraph, it may generate an exception when too few arguments are passed to it.

An (the letter A followed by an integer **n**): The function was called but was passed too many or too few arguments. "**n**" is the number of bytes by which the argument list was incorrect. If **n** is positive, too many arguments (or arguments that were too large) were passed, or the call requires [CDecl](#). If **n** is negative, too few arguments were passed. This situation should be corrected to ensure reliable operation of the function. The presence of this error may also indicate that an exception occurred, in which case the function returns a blank value. Note that due to the x64 calling convention, 64-bit builds never set ErrorLevel to **An**.

Exceptions and [A_LastError](#)

In spite of the built-in exception handling, it is still possible to crash a script with `DllCall`. This can happen when a function does not directly generate an exception but yields something inappropriate, such as a bad pointer or a string that is not terminated. This might not be the function's fault if the script passed it an unsuitable value such as a bad pointer or a "[str](#)" with insufficient capacity. A script can also crash when it specifies an inappropriate argument type or return type, such as claiming that an ordinary integer yielded by a function is an [asterisk variable](#) or [str](#).

The built-in variable **[A_LastError](#)** contains the result of the operating system's

GetLastError() function, which is called immediately after the function is called (this has no measurable impact on performance). A_LastError is a number between 0 and 4294967295 (always formatted as decimal, not hexadecimal). Like [ErrorLevel](#), A_LastError is a per-thread setting; that is, interruptions by other [threads](#) cannot change it. However, A_LastError is also set by [Run/RunWait](#).

Performance

When making repeated calls to a DLL, performance can be dramatically improved by loading it explicitly (*this is not necessary for a [standard DLL](#) such as User32 because it is always resident*). This practice avoids the need for DllCall to internally call LoadLibrary and FreeLibrary each time. For example:

```
hModule := DllCall("LoadLibrary", "Str",  
"MyFunctions.dll", "Ptr") ; Avoids the need for  
DllCall() in the loop to load the library.  
Loop, C:\My Documents\*.*, , 1  
    result := DllCall("MyFunctions\BackupFile",  
"Str", A_LoopFileFullPath)  
DllCall("FreeLibrary", "Ptr", hModule) ; To  
conserve memory, the DLL may be unloaded after  
using it.
```

[v1.0.46.08+]: Even faster performance can be achieved by looking up the function's address beforehand. For example:

```
; In the following example, if the DLL isn't yet  
loaded, use LoadLibrary in place of  
GetModuleHandle.  
MulDivProc := DllCall("GetProcAddress", Ptr,  
DllCall("GetModuleHandle", Str, "kernel32",  
"Ptr"), AStr, "MulDiv", "Ptr")  
Loop 500  
    DllCall(MulDivProc, Int, 3, Int, 4, Int, 3)
```

[AHK_L 31+]: If DllCall's first parameter is a literal string such as "MulDiv" and the DLL containing the function is ordinarily loaded before the script starts, the string is automatically resolved to a function address. This built-in optimization is more effective than the example shown above.

Also, adding the line #NoEnv anywhere in the script improves DllCall's performance when unquoted parameter types are used (e.g. Int vs. "Int").

Finally, when passing a string-variable to a function that will not change the length of the string, performance is improved by passing the variable by address (e.g. &MyVar) rather than as a "str" (especially when the string is very long).

The following example converts a string to uppercase:

```
DllCall("CharUpper", Ptr, &MyVar, Ptr).
```

Structures and Arrays

A structure is a collection of *members* (fields) stored adjacently in memory. Most members tend to be integers.

Functions that accept the address of a structure (or a memory-block array) can be called by storing the structure's raw binary data in a normal variable. The following steps are generally used:

1) Call `VarSetCapacity(MyStruct, 123, 0)` to ensure that the target variable is large enough to hold the structure's data. Replace 123 with a number that is at least as large as the size of the structure. Specifying zero as the last parameter is optional; it initializes all members to be binary zero, which is typically used to avoid calling NumPut() as often in the next step.

2) If the target function uses the values initially in the structure, call `NumPut(123, MyStruct, 4, "UInt")` to initialize any members that should be non-zero. Replace 123 with the integer to be put into the target member (or specify &Var to store the address of a variable). Replace 4 with the offset of the target member (see step #4 for description of "offset"). Replace "UInt" with the appropriate type or omit it if the member is a pointer or handle.

3) Call the target function, passing the address of MyStruct as a UInt (or Ptr in [AHK_L 42+]) argument. For example, `DllCall("MyDll\MyFunc",`

`Ptr, &MyStruct)` . The function will examine and/or change some of the members.

4) Use `MyInteger := NumGet(MyStruct, 4, "UInt")` to retrieve any desired integers from the structure. Replace `4` with the offset of the target member in the structure. The first member is always at offset 0. The second member is at offset 0 plus the size of the first member (typically 4). Members beyond the second are at the offset of the previous member plus the size of the previous member. Most members -- such as `DWORD`, `Int`, and [other types of 32-bit integers](#) -- are 4 bytes in size. Replace `"UInt"` with the appropriate type or omit it if the member is a pointer or handle.

See [Structure Examples](#) for actual usages.

Known Limitations

When a [variable's address](#) (e.g. `&MyVar`) is passed to a function and that function alters the length of the variable's contents, subsequent uses of the variable may behave incorrectly. To fix this, do one of the following: 1) Pass `MyVar` as a `"Str"` argument rather than as a `Ptr/address`; 2) [v1.0.44.03+]: Call `VarSetCapacity(MyVar, -1)` to update the variable's internally-stored length after calling `DllCall`.

Any binary zero stored in a variable by a function hides all data to the right of the zero; that is, such data cannot be accessed or changed by most commands and functions. However, such data can be manipulated by the [address operator](#) and `NumPut/NumGet`, as well as `DllCall` itself.

A function that returns the address of one of the strings that was passed into it might return an identical string in a different memory address than expected. For example calling `CharLower(CharUpper(MyVar))` in a programming language would convert `MyVar`'s contents to lowercase. But when the same is done with `DllCall()`, `MyVar` would be uppercase after the following call because `CharLower` would have operated on a different/temporary string whose contents were identical to `MyVar`:

```
MyVar = ABC
result := DllCall("CharLower", str,
```

```
DllCall("CharUpper", Str, MyVar, Str), Str)
```

To work around this, change the two underlined "Str" values above to Ptr. This interprets CharUpper's return value as a pure address that will get passed as an integer to CharLower.

Certain limitations may be encountered when dealing with strings. For details, see [Script Compatibility](#).

Component Object Model (COM)

COM objects which are accessible to VBScript and similar languages are typically also accessible to AutoHotkey via [ComObjCreate](#), [ComObjGet](#) or [ComObjActive](#) and the built-in [object syntax](#).

COM objects which don't support [IDispatch](#) can be used with DllCall by retrieving the address of a function from the virtual function table of the object's interface. For more details, see [the example](#) further below.

Much of the .NET Framework is also accessible via COM and DllCall. See [.NET Framework Interop](#).

Related

[Script Compatibility](#), [PostMessage](#), [OnMessage\(\)](#), [RegisterCallback\(\)](#), [Run](#), [VarSetCapacity](#), [Functions](#), [SysGet](#), [MSDN Library](#)

Examples

```
; Example: Calls the Windows API function  
"MessageBox" and report which button the user  
presses.
```

```
WhichButton := DllCall("MessageBox", "Int", "0",  
"Str", "Press Yes or No", "Str", "Title of box",  
"Int", 4)  
MsgBox You pressed button #%WhichButton%.
```

; Example: Changes the desktop wallpaper to the specified bitmap (.bmp) file.

```
DllCall("SystemParametersInfo", UInt, 0x14, UInt, 0, Str, A_WinDir . "\winnt.bmp", UInt, 2)
```

; Example: Calls the API function "IsWindowVisible" to find out if a Notepad window is visible.

```
DetectHiddenWindows On  
if not DllCall("IsWindowVisible", "Ptr",  
WinExist("Untitled - Notepad")) ; WinExist()  
returns an HWND.  
    MsgBox The window is not visible.
```

; Example: Calls the API's wsprintf() to pad the number 432 with leading zeros to make it 10 characters wide (0000000432).

```
VarSetCapacity(ZeroPaddedNumber, 20) ; Ensure the  
variable is large enough to accept the new string.  
DllCall("wsprintf", "Str", ZeroPaddedNumber,  
"Str", "%010d", "Int", 432, "Cdecl") ; Requires  
the Cdecl calling convention.
```

```
MsgBox %ZeroPaddedNumber%
```

; [v1.1.17+]: The same can be achieved via Format function in conjunction with the zero flag:

```
MsgBox % Format("{:010}", 432)
```

; Example: Demonstrates QueryPerformanceCounter(), which gives more precision than A_TickCount's 10ms.

```
DllCall("QueryPerformanceCounter", "Int64*",  
CounterBefore)  
Sleep 1000  
DllCall("QueryPerformanceCounter", "Int64*",  
CounterAfter)  
MsgBox % "Elapsed QPC time is " . CounterAfter -  
CounterBefore
```

```
; Example: This is a hotkey that temporarily  
reduces the mouse cursor's speed, which  
facilitates precise positioning.  
; Hold down the F1 key to slow down the cursor.  
Release it to return to original speed.
```

```
F1::  
SPI_GETMOUSESPEED = 0x70  
SPI_SETMOUSESPEED = 0x71  
; Retrieve the current speed so that it can be  
restored later:  
DllCall("SystemParametersInfo", UInt,  
SPI_GETMOUSESPEED, UInt, 0, UIntP, OrigMouseSpeed,  
UInt, 0)  
; Now set the mouse to the slower speed specified  
in the next-to-last parameter (the range is 1-20,  
10 is default):  
DllCall("SystemParametersInfo", UInt,  
SPI_SETMOUSESPEED, UInt, 0, Ptr, 3, UInt, 0)  
KeyWait F1 ; This prevents keyboard auto-repeat  
from doing the DllCall repeatedly.  
return
```

```
F1 up::DllCall("SystemParametersInfo", UInt, 0x71,  
UInt, 0, Ptr, OrigMouseSpeed, UInt, 0) ; Restore  
the original speed.
```

```
; Example: When passed a window's Unique ID and
the text or ClassNN of one of its controls,
; the following function returns the HWND (unique
ID) of that control.
; [v1.0.43.06+]: This function has been superseded
by the following command, which is more accurate.
```

`ControlGet, OutputVar, Hwnd,, ClassNN, WinTitle`

```
; Example: Monitors the active window and display
the position of its vertical scroll bar in its
; focused control (with real-time updates). This
requires [v1.0.43.06+] because it uses ControlGet
Hwnd.
```

```
#Persistent
```

```
SetTimer, WatchScrollBar, 100
return
```

```
WatchScrollBar:
```

```
ActiveWindow := WinExist("A")
```

```
if not ActiveWindow ; No active window.
    return
```

```
ControlGetFocus, FocusedControl, ahk_id
```

```
%ActiveWindow%
```

```
if not FocusedControl ; No focused control.
    return
```

```
; Display the vertical or horizontal scroll bar's
position in a Tooltip:
```

```
ControlGet, ChildHWND, Hwnd,, %FocusedControl%,
ahk_id %ActiveWindow%
```

```
ToolTip % DllCall("GetScrollPos", "Ptr",
ChildHWND, "Int", 1) ; Last parameter is 1 for
SB_VERT, 0 for SB_HORZ.
```

```
return
```

```
; Example: This is a working script that writes
some text to a file then reads it back into memory
(requires v1.0.34+).
; This method can be used to help performance in
cases where multiple files are being read or
written simultaneously.
; [AHK_L 42+]: The same can be achieved using
FileOpen.
```

```
FileSelectFile, FileName, S16,, Create a new file:
if FileName =
    return
GENERIC_WRITE = 0x40000000 ; Open the file for
writing rather than reading.
CREATE_ALWAYS = 2 ; Create new file (overwriting
any existing file).
hFile := DllCall("CreateFile", Str, FileName,
UInt, GENERIC_WRITE, UInt, 0, Ptr, 0, UInt,
CREATE_ALWAYS, UInt, 0, Ptr, 0, Ptr)
if not hFile
{
    MsgBox Can't open "%FileName%" for writing.
    return
}
TestString = This is a test string.`r`n ; When
writing a file this way, use `r`n rather than `n
to start a new line.
DllCall("WriteFile", Ptr, hFile, Str, TestString,
UInt, StrLen(TestString), UIntP,
BytesActuallyWritten, Ptr, 0)
DllCall("CloseHandle", Ptr, hFile) ; Close the
file.

; Now that the file was written, read its contents
back into memory.
GENERIC_READ = 0x80000000 ; Open the file for
reading rather than writing.
```

```
OPEN_EXISTING = 3 ; This mode indicates that the
file to be opened must already exist.
FILE_SHARE_READ = 0x1 ; This and the next are
whether other processes can open the file while we
have it open.
FILE_SHARE_WRITE = 0x2
hFile := DllCall("CreateFile", Str, FileName,
UInt, GENERIC_READ, UInt,
FILE_SHARE_READ|FILE_SHARE_WRITE, Ptr, 0, UInt,
OPEN_EXISTING, UInt, 0, Ptr, 0)
if not hFile
{
    MsgBox Can't open "%FileName%" for reading.
    return
}
; Make the variable empty for testing purposes,
but ensure it retains sufficient capacity:
BytesToRead := VarSetCapacity(TestString,
StrLen(TestString))
DllCall("ReadFile", Ptr, hFile, Str, TestString,
UInt, BytesToRead, UIntP, BytesActuallyRead, Ptr,
0)
DllCall("CloseHandle", Ptr, hFile) ; Close the
file.
MsgBox The following string was read from the
file: %TestString%
```

```
; Example: Hides the mouse cursor when you press
Win+C. To later show the cursor, press Win+C
again.
```

```
; This script is from
www.autohotkey.com/forum/topic6107.html
```

```
OnExit, ShowCursor ; Ensure the cursor is made
visible when the script exits.
return
```

```

ShowCursor:
SystemCursor("On")
ExitApp

#c::SystemCursor("Toggle") ; Win+C hotkey to
toggle the cursor on and off.

SystemCursor(OnOff=1) ; INIT = "I","Init"; OFF =
0,"Off"; TOGGLE = -1,"T","Toggle"; ON = others
{
    static AndMask, XorMask, $, h_cursor

, c0, c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13 ;
system cursors

b1, b2, b3, b4, b5, b6, b7, b8, b9, b10, b11, b12, b13 ;
blank cursors

h1, h2, h3, h4, h5, h6, h7, h8, h9, h10, h11, h12, h13 ;
handles of default cursors
    if (OnOff = "Init" or OnOff = "I" or $ = "")
; init when requested or at first call
    {
        $ = h
; active default cursors
        VarSetCapacity( h_cursor, 4444, 1 )
        VarSetCapacity( AndMask, 32*4, 0xFF )
        VarSetCapacity( XorMask, 32*4, 0 )
        system_cursors =
32512, 32513, 32514, 32515, 32516, 32642, 32643, 32644, 32
645, 32646, 32648, 32649, 32650
        StringSplit c, system_cursors, ` ,
        Loop %c0%
        {
            h_cursor := DllCall( "LoadCursor",
"Ptr", 0, "Ptr", c%A_Index% )

```

```

        h%A_Index% := DllCall( "CopyImage",
"Ptr",h_cursor, "UInt",2, "Int",0, "Int",0,
"UInt",0 )
        b%A_Index% := DllCall( "CreateCursor",
"Ptr",0, "Int",0, "Int",0
, "Int",32, "Int",32,
"Ptr",&AndMask, "Ptr",&XorMask )
    }
}
if (OnOff = 0 or OnOff = "Off" or $ = "h" and
(OnOff < 0 or OnOff = "Toggle" or OnOff = "T"))
    $ = b ; use blank cursors
else
    $ = h ; use the saved cursors

Loop %c0%
{
    h_cursor := DllCall( "CopyImage",
"Ptr",%$%A_Index%, "UInt",2, "Int",0, "Int",0,
"UInt",0 )
    DllCall( "SetSystemCursor",
"Ptr",h_cursor, "UInt",c%A_Index% )
}
}

```

; Structure Example: Pass the address of a RECT structure to GetWindowRect(), which sets the structure's members to the positions of the left, top, right, and bottom sides of a window (relative to the screen).

Run Notepad

```

WinWait Untitled - Notepad ; This also sets the
"last found window" for use with WinExist() below.
VarSetCapacity(Rect, 16) ; A RECT is a struct

```

```
consisting of four 32-bit integers (i.e. 4*4=16).
DllCall("GetWindowRect", Ptr, WinExist(), Ptr,
&Rect) ; WinExist() returns an HWND.
MsgBox % "Left " . NumGet(Rect, 0, "Int") . " Top
" . NumGet(Rect, 4, "Int")
      . " Right " . NumGet(Rect, 8, "Int") . "
Bottom " . NumGet(Rect, 12, "Int")
```

```
; Structure Example: Pass to FillRect() the
address of a RECT structure that indicates a part
of the
; screen to temporarily paint red.
```

```
VarSetCapacity(Rect, 16, 0) ; Set capacity to
hold four 4-byte integers and initialize them all
to zero.
```

```
NumPut(A_ScreenWidth//2, Rect, 8, "Int") ; The
third integer in the structure is "rect.right".
NumPut(A_ScreenHeight//2, Rect, 12, "Int") ; The
fourth integer in the structure is "rect.bottom".
hDC := DllCall("GetDC", "Ptr", 0, "Ptr") ; Pass
zero to get the desktop's device context.
hBrush := DllCall("CreateSolidBrush", "UInt",
0x0000FF, "Ptr") ; Create a red brush (0x0000FF
is in BGR format).
DllCall("FillRect", "Ptr", hDC, "Ptr", &Rect,
"Ptr", hBrush) ; Fill the specified rectangle
using the brush above.
DllCall("ReleaseDC", "Ptr", 0, "Ptr", hDC) ;
Clean-up.
DllCall("DeleteObject", "Ptr", hBrush) ; Clean-
up.
```

```
; Structure Example: Change the system's clock to
the specified date and time. Use caution when
; changing to a date in the future as it may cause
```

scheduled tasks to run prematurely!

```
SetSystemTime("20051008142211") ; Pass it a  
timestamp (local, not UTC).
```

```
SetSystemTime(YYYYMMDDHHMISS)  
; Sets the system clock to the specified date and  
time.  
; Caller must ensure that the incoming parameter  
is a valid date-time stamp  
; (local time, not UTC). Returns non-zero upon  
success and zero otherwise.
```

```
{  
    ; Convert the parameter from local time to UTC  
for use with SetSystemTime().  
    UTC_Delta -= A_NowUTC, Seconds ; Seconds is  
more accurate due to rounding issue.  
    UTC_Delta := Round(-UTC_Delta/60) ; Round to  
nearest minute to ensure accuracy.  
    YYYYMMDDHHMISS += UTC_Delta, Minutes ; Apply  
offset to convert to UTC.
```

```
    VarSetCapacity(SystemTime, 16, 0) ; This  
struct consists of 8 UShorts (i.e. 8*2=16).
```

```
    StringLeft, Int, YYYYMMDDHHMISS, 4 ; YYYY  
(year)  
    NumPut(Int, SystemTime, 0, "UShort")  
    StringMid, Int, YYYYMMDDHHMISS, 5, 2 ; MM  
(month of year, 1-12)  
    NumPut(Int, SystemTime, 2, "UShort")  
    StringMid, Int, YYYYMMDDHHMISS, 7, 2 ; DD  
(day of month)  
    NumPut(Int, SystemTime, 6, "UShort")  
    StringMid, Int, YYYYMMDDHHMISS, 9, 2 ; HH  
(hour in 24-hour time)  
    NumPut(Int, SystemTime, 8, "UShort")
```

```

    StringMid, Int, YYYYMMDDHHMISS, 11, 2 ; MI
(minute)
    NumPut(Int, SystemTime, 10, "UShort")
    StringMid, Int, YYYYMMDDHHMISS, 13, 2 ; SS
(second)
    NumPut(Int, SystemTime, 12, "UShort")

    return DllCall("SetSystemTime", Ptr,
&SystemTime)
}

```

/* More Structure Examples:

1) See the [WinLIRC client script](#) for a demonstration of how to use DllCall() to make a network connection to a TCP/IP server and receive data from it.

2) The operating system offers standard dialog boxes that prompt the user to pick a font and/or color, or an icon.

These dialogs use structures and are demonstrated at www.autohotkey.com/forum/topic17230.html.

*/

/*

Example: Temporarily remove the active window from the taskbar by using COM.

Methods in ITaskbarList's VTable:

IUnknown:

0	QueryInterface	-- use ComObjQuery instead
1	AddRef	-- use ObjAddRef instead
2	Release	-- use ObjRelease instead

ITaskbarList:

```

        3 HrInit
        4 AddTab
        5 DeleteTab
        6 ActivateTab
        7 SetActiveAlt
*/
IID_ITaskbarList := "{56FDF342-FD6D-11d0-958A-006097C9A090}"
CLSID_TaskbarList := "{56FDF344-FD6D-11d0-958A-006097C9A090}"

; Create the TaskbarList object and store its
address in tbl.
tbl := ComObjCreate(CLSID_TaskbarList,
IID_ITaskbarList)

activeHwnd := WinExist("A")

DllCall(vtable(tbl,3), "ptr", tbl)
; tbl.HrInit()
DllCall(vtable(tbl,5), "ptr", tbl, "ptr",
activeHwnd) ; tbl.DeleteTab(activeHwnd)
Sleep 3000
DllCall(vtable(tbl,4), "ptr", tbl, "ptr",
activeHwnd) ; tbl.AddTab(activeHwnd)

; Non-dispatch objects must always be manually
freed.
ObjRelease(tbl)

vtable(ptr, n) {
    ; NumGet(ptr+0) returns the address of the
object's virtual function
    ; table (vtable for short). The remainder of
the expression retrieves
    ; the address of the nth function's address
from the vtable.

```

```
    return NumGet(NumGet(ptr+0), n*A_PtrSize)  
}
```

NumGet

Returns the binary number stored at the specified address+offset.

```
Number := NumGet(VarOrAddress [, Offset := 0][, Type := "UPtr"])
```

Parameters

VarOrAddress

A memory address or variable. If *VarOrAddress* is a variable such as `MyVar`, the address of the variable's string buffer is used. This is usually equivalent to passing `&MyVar`, but omitting the "&" performs better and ensures that the target address + offset is [valid](#).

Do not pass a variable reference if the variable *contains* the target address; in that case, pass an expression such as `MyVar+0`.

Offset

An offset - in bytes - which is added to *VarOrAddress* to determine the target address.

Type

One of the following strings (defaults to UPtr if omitted):
UInt, Int, Int64, Short, UShort, Char, UChar, Double, Float, Ptr or UPtr

Unlike DllCall, these must be enclosed in quotes when used as literal strings.

For details see [DllCall Types](#).

Return Value

If the target address is invalid, an empty string is returned. However, some invalid addresses cannot be detected as such and may cause unpredictable

behaviour.

Otherwise, the number at the specified address+offset is returned.

General Remarks

If only two parameters are present, the second parameter can be either *Offset* or *Type*. For example, `NumGet(var, "int")` is valid.

Related

[NumPut](#), [DllCall](#), [VarSetCapacity](#)

NumPut

Stores a number in binary format at the specified address+offset.

```
NumPut(Number, VarOrAddress [, Offset := 0][, Type := "UPtr"])
```

Parameters

Number

The number to store.

VarOrAddress

A memory address or variable. If *VarOrAddress* is a variable such as `MyVar`, the address of the variable's string buffer is used. This is usually equivalent to passing `&MyVar`, but omitting the "&" performs better and ensures that the target address + offset is [valid](#).

Do not pass a variable reference if the variable *contains* the target address; in that case, pass an expression such as `MyVar+0`.

Offset

An offset - in bytes - which is added to *VarOrAddress* to determine the target address.

Type

One of the following strings (defaults to `UPtr` if omitted):
`UInt`, `Int`, `Int64`, `Short`, `UShort`, `Char`, `UChar`, `Double`, `Float`, `Ptr` or `UPtr`

Unlike `DllCall`, these must be enclosed in quotes when used as literal strings.

For details see [DllCall Types](#).

Return Value

If the target address is invalid, an empty string is returned. However, some invalid addresses cannot be detected as such and may cause unpredictable behaviour.

Otherwise, the address to the right of the item just written is returned. This is often used when writing a sequence of numbers of different types, such as in a structure for use with DllCall.

General Remarks

If an integer is too large to fit in the specified *Type*, its most significant bytes are ignored; e.g. `NumPut(257, var, 0, "Char")` would store the number 1.

If only three parameters are present, the third parameter can be either *Offset* or *Type*. For example, `NumPut(x, var, "int")` is valid.

Related

[NumGet](#), [DllCall](#), [VarSetCapacity](#)

RegisterCallback() [v1.0.47+]

Creates a machine-code address that when called, redirects the call to a [function](#) in the script.

```
Address := RegisterCallback("FunctionName" [, Options := "", ParamCount := FormalCount, EventInfo := Address])
```

Parameters

Address

Upon success, RegisterCallback() returns a numeric address that may be called by [DllCall\(\)](#) or anything else capable of calling a machine-code function. Upon failure, it returns an empty string. Failure occurs when *FunctionName*: 1) does not exist; 2) accepts too many or too few parameters according to *ParamCount*; or 3) accepts any [ByRef parameters](#).

FunctionName

A [function](#)'s name, which must be enclosed in quotes if it is a literal string. This function is called automatically whenever *Address* is called. The function also receives the parameters that were passed to *Address*.

[v1.1.06+]: A [function reference](#) can be passed instead of a function name.

Options

Specify zero or more of the following words. Separate each option from the next with a space (e.g. `C Fast`).

Fast or **F**: Avoids starting a new [thread](#) each time *FunctionName* is called. Although this performs better, it must be avoided whenever the thread from which *Address* is called varies (e.g. when the callback is triggered by an incoming message). This is because *FunctionName* will

be able to change global settings such as [ErrorLevel](#), [A_LastError](#), and the [last-found window](#) for whichever thread happens to be running at the time it is called. For more information, see [Remarks](#).

CDecl or **C**: Makes *Address* conform to the "C" calling convention. This is typically omitted because the standard calling convention is much more common for callbacks.

ParamCount

The number of parameters that *Address*'s caller will pass to it. If entirely omitted, it defaults to the number of mandatory parameters in the [definition](#) of *FunctionName*. In either case, ensure that the caller passes exactly this number of parameters.

EventInfo

An integer that *FunctionName* will see in [A_EventInfo](#) whenever it is called via this *Address*. This is useful when *FunctionName* is called by more than one *Address*. If omitted, it defaults to *Address*. Note: Unlike other global settings, the [current thread](#)'s [A_EventInfo](#) is not disturbed by the [fast mode](#).

If the exe running the script is 32-bit, this parameter must be between 0 and 4294967295. If the exe is 64-bit, this parameter can be a 64-bit integer. Although [A_EventInfo](#) usually returns an unsigned integer, AutoHotkey does not fully support unsigned 64-bit integers and therefore some operations may cause the value to wrap into the signed range.

The Callback Function's Parameters

A [function](#) assigned to a callback address may accept up to 31 parameters. [Optional parameters](#) are permitted, which is useful when the function is called by more than one caller.

Interpreting the parameters correctly requires some understanding of how the x86 calling conventions work. Since AutoHotkey does not have typed parameters, the callback's parameter list is assumed to consist of integers, and some reinterpretation may be required.

AutoHotkey 32-bit: All incoming parameters are unsigned 32-bit integers.

Smaller types are padded out to 32 bits, while larger types are split into a series of 32-bit parameters.

If an incoming parameter is intended to be a signed integer, any negative numbers can be revealed by following either of the following examples:

```
; Method #1
if wParam > 0x7FFFFFFF
    wParam := -(~wParam) - 1

; Method #2: Relies on the fact that AutoHotkey
natively uses signed 64-bit integers.
wParam := wParam << 32 >> 32
```

AutoHotkey 64-bit: All incoming parameters are signed 64-bit integers. AutoHotkey does not natively support unsigned 64-bit integers. Smaller types are padded out to 64 bits, while larger types are always passed by address.

AutoHotkey 32-bit/64-bit: If an incoming parameter is intended to be 8-bit or 16-bit (or 32-bit on x64), the upper bits of the value might contain "garbage" which can be filtered out by using bitwise-and, as in the following examples:

```
Callback(UCharParam, UShortParam, UIntParam) {
    UCharParam &= 0xFF
    UShortParam &= 0xFFFF
    UIntParam &= 0xFFFFFFFF
    ; ...
}
```

If an incoming parameter is intended by its caller to be a string, what it actually receives is the address of the string. To retrieve the string itself, use [StrGet](#):

```
MyString := StrGet(MyParameter) ; Requires
[AHK_L 46+]
```

If an incoming parameter is the address of a structure, the individual members may be extracted by following the steps at [DllCall structures](#).

Receiving parameters by address [AHK_L 60+]: If the function is declared as [variadic](#), its final parameter is assigned the *address* of the first callback parameter which was not assigned to a script parameter. For example:

```
callback := RegisterCallback("TheFunc", "F", 3)
; Parameter list size must be specified.
TheFunc("TheFunc was called directly.")
; Call TheFunc directly.
DllCall(callback, float, 10.5, int64, 42)
; Call TheFunc via callback.
TheFunc(params*) {
    if IsObject(params)
        MsgBox % params[1]
    else
        MsgBox % NumGet(params+0, "float") ", "
        NumGet(params+A_PtrSize, "int64")
}
```

Most callbacks use the *stdcall* calling convention, which requires a fixed number of parameters. In those cases, *ParamCount* must be set to the size of the parameter list, where *Int64* and *Double* count as two 32-bit parameters.

With *Cdecl* or the 64-bit calling convention, *ParamCount* only affects how many script parameters are assigned values. If omitted, all optional parameters receive their default values and are excluded from the calculations for the address stored in *params*.

What the Function Should Return

If the function uses [Return](#) without any parameters, or it specifies a blank value such as "" (or it never uses [Return](#) at all), 0 is returned to the caller of the callback. Otherwise, the function should return an integer, which is then returned to the caller. AutoHotkey 32-bit truncates return values to 32-bit, while AutoHotkey 64-bit supports 64-bit return values. Returning structs larger than

this (by value) is not supported.

Fast vs. Slow

The default/slow mode causes the function to start off fresh with the default values for settings such as [SendMode](#) and [DetectHiddenWindows](#). These defaults can be changed in the [auto-execute section](#).

By contrast, the [fast mode](#) inherits global settings from whichever [thread](#) happens to be running at the time the function is called. Furthermore, any changes the function makes to global settings (including [ErrorLevel](#) and the [last-found window](#)) will go into effect for the [current thread](#). Consequently, the fast mode should be used only when it is known exactly which thread(s) the function will be called from.

To avoid being interrupted by itself (or any other thread), a callback may use [Critical](#) as its first line. However, this is not completely effective when the function is called indirectly via the arrival of a message less than 0x312 (increasing [Critical's interval](#) may help). Furthermore, [Critical](#) does not prevent the function from doing something that might indirectly result in a call to itself, such as calling [SendMessage](#) or [DllCall](#).

Memory

Each use of [RegisterCallback\(\)](#) allocates a small amount of memory (32 bytes plus system overhead). Since the OS frees this memory automatically when the script exits, any script that allocates a small, *fixed* number of callbacks does not have to explicitly free the memory. By contrast, a script that calls [RegisterCallback\(\)](#) an indefinite/unlimited number of times should explicitly call the following on any unused callbacks:

```
DllCall("GlobalFree", "Ptr", Address, "Ptr")
```

Related

[DllCall\(\)](#), [OnMessage\(\)](#), [OnExit](#), [OnClipboardChange](#), [Sort's callback](#), [Critical](#), [Post/SendMessage, Functions](#), [List of Windows Messages](#), [Threads](#)

Examples

```
; Example: The following is a working script that
displays a summary of all top-level windows.
```

```
; For performance and memory conservation, call
RegisterCallback() only once for a given callback:
if not EnumAddress ; Fast-mode is okay because it
will be called only from this thread:
```

```
EnumAddress :=
RegisterCallback("EnumWindowsProc", "Fast")
```

```
DetectHiddenWindows On ; Due to fast-mode, this
setting will go into effect for the callback too.
```

```
; Pass control to EnumWindows(), which calls the
callback repeatedly:
```

```
DllCall("EnumWindows", Ptr, EnumAddress, Ptr, 0)
MsgBox %Output% ; Display the information
accumulated by the callback.
```

```
EnumWindowsProc(hwnd, lParam)
{
    global Output
    WinGetTitle, title, ahk_id %hwnd%
    WinGetClass, class, ahk_id %hwnd%
    if title
        Output .= "HWND: " . hwnd . "`tTitle: " .
title . "`tClass: " . class . "`n"
    return true ; Tell EnumWindows() to continue
until all windows have been enumerated.
}
```

```
; Example: The following is a working script that
```

demonstrates how to subclass a GUI window by
; redirecting its WindowProc to a new WindowProc
in the script. In this case, the background
; color of a text control is changed to a custom
color.

```
TextBackgroundColor := 0xFFBBBB ; A custom color  
in BGR format.
```

```
TextBackgroundBrush := DllCall("CreateSolidBrush",  
UInt, TextBackgroundColor)
```

```
Gui, Add, Text, HwndMyTextHwnd, Here is some text  
that is given`na custom background color.
```

```
Gui +LastFound
```

```
GuiHwnd := WinExist()
```

```
; 64-bit scripts must call SetWindowLongPtr  
instead of SetWindowLong:
```

```
SetWindowLong := A_PtrSize=8 ? "SetWindowLongPtr"  
: "SetWindowLong"
```

```
WindowProcNew := RegisterCallback("WindowProc", ""  
; Specify "" to avoid fast-mode for subclassing.  
    , 4, MyTextHwnd) ; Must specify exact  
ParamCount when EventInfo parameter is present.
```

```
WindowProcOld := DllCall(SetWindowLong, Ptr,  
GuiHwnd, Int, -4 ; -4 is GWL_WNDPROC  
    , Ptr, WindowProcNew, Ptr) ; Return value must  
be set to Ptr or UPtr vs. Int.
```

```
Gui Show  
return
```

```
WindowProc(hwnd, uMsg, wParam, lParam)  
{  
    Critical  
    global TextBackgroundColor,
```

```
TextBackgroundBrush, WindowProcOld
    if (uMsg = 0x138 && lParam = A_EventInfo) ;
0x138 is WM_CTLCOLORSTATIC.
    {
        DllCall("SetBkColor", Ptr, wParam, UInt,
TextBackgroundColor)
        return TextBackgroundBrush ; Return the
HBRUSH to notify the OS that we altered the HDC.
    }
    ; Otherwise (since above didn't return), pass
all unhandled events to the original WindowProc.
    return DllCall("CallWindowProc", Ptr,
WindowProcOld, Ptr, hwnd, UInt, uMsg, Ptr, wParam,
Ptr, lParam)
}
```

```
GuiClose:
ExitApp
```

StrPut() / StrGet() [AHK_L 46+]

Copies a string to or from a memory address, optionally converting to or from a given code page.

```
StrPut(String [, Encoding := None])
StrPut(String, Address [, Length] [, Encoding := None])
StrGet(Address [, Length] [, Encoding := None])
```

Parameters

String

Any string. Numbers are also acceptable.

Address

The address at which the string will be written to/read from.

Length

The maximum number of characters to read/write, including the null-terminator if required.

StrPut: See *Return Value* below for more details.

Encoding

The source encoding for StrGet or target encoding for StrPut; for example, "UTF-8", "UTF-16" or "CP936". If *Address* and *Length* are not specified, numeric identifiers must be prefixed with "CP". Specify an empty string or "CP0" to use the system default ANSI code page.

Return Value

For either function, invalid parameters cause an empty string to be returned.

StrPut returns the number of characters written, the required buffer size in characters if no *Address* was given, or 0 if an error occurred. If *Length* is less than the length of the converted string, the function fails and returns 0. If *Length* is exactly the length of the converted string, the string is not null-terminated; otherwise the returned count includes the null-terminator.

StrGet returns the requested string after performing any necessary conversion.

Remarks

Note that the *String* parameter of StrPut and return value of StrGet are always in the [native encoding](#) of the current executable, whereas *Encoding* specifies the encoding of the string written to or read from the given *Address*. If no *Encoding* is specified, the string is simply measured or copied without any conversion taking place.

If conversion between code pages is necessary, the required buffer size may differ from the size of the source *String*.

Scripts which are required to be compatible with AutoHotkey Basic can still use StrPut and StrGet provided that [the appropriate script files](#) are installed in a [function library](#). These scripts can be found at the [AutoHotkey Community Forum](#).

Related

[Script Compatibility](#), [FileEncoding](#), [VarSetCapacity\(\)](#)

Examples

Either *Length* or *Encoding* may be specified directly after *Address*, but in those cases *Encoding* must be non-numeric:

```
strA := StrGet(addressA, "cp0")      ; OK
strA := StrGet(addressA, length, 0) ; OK
strA := StrGet(addressA, 0)         ; Error
```

StrPut may be called once to calculate the required buffer size for a string in a

particular encoding, then again to encode and write the string into the buffer. If you frequently use variables with `StrPut`, consider adding this function to your [library](#):

```
StrPutVar(string, ByRef var, encoding)
{
    ; Ensure capacity.
    VarSetCapacity( var, StrPut(string, encoding)
        ; StrPut returns char count, but
    VarSetCapacity needs bytes.
        * ((encoding="utf-16" || encoding="cp1200")
? 2 : 1) )
    ; Copy or convert the string.
    return StrPut(string, &var, encoding)
}
```

VarSetCapacity()

Enlarges a variable's holding capacity or frees its memory. Normally, this is necessary only for unusual circumstances such as [DllCall](#).

```
GrantedCapacity := VarSetCapacity(TargetVar [,  
RequestedCapacity, FillByte])
```

Parameters

TargetVar

A reference to a variable. For example: `VarSetCapacity(MyVar, 1000)`. This can also be a dynamic variable such as `Array%i%` or a function's [ByRef](#) parameter.

RequestedCapacity

If omitted, the variable's current capacity will be returned and its contents will not be altered. Otherwise, anything currently in the variable is lost (the variable becomes blank).

Specify for *RequestedCapacity* the number of bytes that the variable should be able to hold after the adjustment. For Unicode strings, this should be the length times two. *RequestedCapacity* does not include the internal zero terminator. For example, specifying 1 would allow the variable to hold up to one byte in addition to its internal terminator. Note: the variable will auto-expand if the script assigns it a larger value later.

Since this function is often called simply to ensure the variable has a certain minimum capacity, for performance reasons, it shrinks the variable only when *RequestedCapacity* is 0. In other words, if the variable's capacity is already greater than *RequestedCapacity*, it will not be reduced (but the variable will still be made blank for consistency).

Therefore, to explicitly shrink a variable, first free its memory with `VarSetCapacity(Var, 0)` and then use

`VarSetCapacity(Var, NewCapacity)` -- or simply let it auto-expand from zero as needed.

For performance reasons, freeing a variable whose previous capacity was less than 64 characters (128 bytes in Unicode builds) might have no effect because its memory is of a permanent type. In this case, the current capacity will be returned rather than 0.

For performance reasons, the memory of a variable whose capacity is less than 4096 bytes is not freed by storing an empty string in it (e.g. `Var := ""`). However, `VarSetCapacity(Var, 0)` does free it.

[v1.0.44.03+]: Specify -1 for *RequestedCapacity* to update the variable's internally-stored string length to the length of its current contents. This is useful in cases where the variable has been altered indirectly, such as by passing its [address](#) via `DllCall()`. In this mode, `VarSetCapacity()` returns the length in bytes rather than the capacity.

FillByte

This parameter is normally omitted, in which case the memory of the target variable is not initialized (instead, the variable is simply made blank as described above). Otherwise, specify a number between 0 and 255. Each byte in the target variable's memory area (its current capacity, which might be greater than *RequestedCapacity*) is set to that number. Zero is by far the most common value, which is useful in cases where the variable will hold raw binary data such as a [DllCall structure](#).

Return Value

This function returns the number of bytes that `Var` can now hold, which will be greater than or equal to *RequestedCapacity*. If *VarName* is not a valid variable name (such as a literal string or number), 0 is returned. If the system has insufficient memory to make the change (very rare), an error dialog is displayed and the current capacity is returned - this behaviour may change in a future version.

Remarks

In addition to its uses described at [DllCall](#), this function can also be used to

enhance performance when building a string by means of gradual concatenation. This is because multiple automatic resizings can be avoided when you have some idea of what the string's final length will be. In such a case, *RequestedCapacity* need not be accurate: if the capacity is too small, performance is still improved and the variable will begin auto-expanding when the capacity has been exhausted. If the capacity is too large, some of the memory is wasted, but only temporarily because all the memory can be freed after the operation by means of `VarSetCapacity(Var, 0)` or `Var := ""`.

`#MaxMem` restricts only the automatic expansion that a variable does on its own. It does not affect `VarSetCapacity`.

Related

[DllCall](#), [#MaxMem](#), [NumPut](#), [NumGet](#)

Example

```
; Optimize by ensuring MyVar has plenty of space
to work with.
VarSetCapacity(MyVar, 10240000) ; ~10 MB
Loop
{
    ...
    MyVar = %MyVar%%StringToConcatenate%
    ...
}
```

```
; Calculate required buffer space for a string.
bytes_per_char := A_IsUnicode ? 2 : 1
max_chars := 500
max_bytes := max_chars * bytes_per_char

Loop 2
{
    ; Allocate space for use with DllCall.
```

```

VarSetCapacity(buf, max_bytes)

if A_Index = 1
    ; Alter the variable indirectly via
DllCall.
    DllCall("wsprintf", "Ptr", &buf, "Str",
"0x%08x", "UInt", 4919)
    else
        ; Use "str" to update the length
automatically:
        DllCall("wsprintf", "Str", buf, "Str",
"0x%08x", "UInt", 4919)

        ; Concatenate a string to demonstrate why the
length needs to be updated:
wrong_str := buf . "<end>"
wrong_len := StrLen(buf)

        ; Update the variable's length.
VarSetCapacity(buf, -1)

right_str := buf . "<end>"
right_len := StrLen(buf)

MsgBox,
(
Before updating
    String: %wrong_str%
    Length: %wrong_len%

After updating
    String: %right_str%
    Length: %right_len%
)
}

```

ComObjActive() [AHK_L 53+]

Retrieves a running object that has been registered with OLE.

```
ComObject := ComObjActive(CLSID)
```

Creates an object representing a typed value to be passed as a parameter or return value.

```
ParamObj := ComObject(VarType, Value [, Flags])
```

DEPRECATED:

The usages shown below are deprecated and may be altered or unavailable in a future release.

Creates an object which may be used in place of an optional parameter's default value when calling a method of a COM object. [v1.1.12+]: This function is obsolete. Instead, simply write two consecutive commas, as in `Obj.Method(1, , 3)`

```
ParamObj := ComObjMissing()
```

Wraps or unwraps a raw [IDispatch](#) pointer in a usable object and automatically calls `AddRef`.

```
ComObject := ComObjEnwrap(DispPtr)  
DispPtr := ComObjUnwrap(ComObject)
```

To write more future-proof code, use the following instead:

```
ComObject := ComObject(9, DispPtr, 1),  
ObjAddRef(DispPtr)  
DispPtr := ComObjValue(ComObject),  
ObjAddRef(DispPtr)
```

Parameters

CLSID

CLSID or human-readable Prog ID of the COM object to retrieve.

ComObject

COM object usable with [object syntax](#).

VarType

An integer indicating the type of value. See [ComObjType\(\)](#) for a list of types.

Value

The value to wrap. Currently only integer and pointer values are supported.

Flags

Flags affecting the behaviour of this function and the wrapper object; see below.

DispPtr

Raw IDispatch pointer.

Flags

0

Default behaviour. [AddRef](#) is called automatically for IUnknown and IDispatch pointers, so the caller should use [ObjRelease](#) to release their copy of the pointer if appropriate.

As the default behaviour may be changing in a future release, it is

	recommended to always set <i>Flags</i> to 1 when wrapping an interface pointer, and call <code>ObjAddRef()</code> if needed.
1	Take ownership of an IUnknown, IDispatch or SAFEARRAY pointer. <code>AddRef</code> is not called. If the wrapper object contains a SAFEARRAY (excluding VT_BYREF), <code>SafeArrayDestroy</code> is called automatically when the wrapper object is freed.

ByRef [v1.1.17+]

If a wrapper object's *VarType* includes the VT_BYREF (0x4000) flag, empty brackets `[]` can be used to read or write the referenced value.

When creating a reference, *Value* must be the memory address of a variable or buffer with sufficient capacity to store a value of the given type. For example, the following can be used to create a variable which a VBScript function can write into:

```

VarSetCapacity(var, 24, 0)
vref := ComObject(0x400C, &var) ; 0x400C is a
combination of VT_BYREF and VT_VARIANT.

vref[] := "in value"
sc.Run("Example", vref) ; sc should be
initialized as in the example below.
MsgBox % vref[]

```

General Remarks

In current versions, any function-call beginning with "ComObj" that does not match one of the other COM functions actually calls `ComObjActive`. For example, `ComObjEnwrap(DispPtr)` and `ComObjActive(DispPtr)` are both equivalent to `ComObject(DispPtr)` (*VarType* 9 is implied). However, this behaviour will be unavailable in a future release, so it is best to use only `ComObject()` and `ComObjActive()` as shown on this page.

If ComObjActive cannot retrieve an active object, it may throw an exception, exit the script or return an empty string, depending on the current [ComObjError](#) setting and [other factors](#).

When this function is used to wrap or retrieve an IDispatch or IUnknown interface pointer, the default behaviour is to increment the COM object's reference count. Therefore, the interface pointer must be [manually released](#) when it is no longer needed. When the wrapper object is freed, the reference it contains is automatically released.

Known limitation: Each time a COM object is wrapped, a new wrapper object is created. Comparisons and assignments such as `obj1 == obj2` and `array[obj1] := value` treat the two wrapper objects as unique, even though they contain the same COM object.

Related

[ComObjCreate](#), [ComObjGet](#), [ComObjConnect](#), [ComObjError](#), [ComObjFlags](#), [ObjAddRef/ObjRelease](#), [ComObjQuery](#), [GetActiveObject \(MSDN\)](#)

Examples

ComObjUnwrap: See [ComObjConnect](#).

```
; Preamble - ScriptControl requires a 32-bit
version of AutoHotkey.
code =
(
Sub Example(Var)
    MsgBox Var
    Var = "out value!"
End Sub
)
sc := ComObjCreate("ScriptControl"), sc.Language
:= "VBScript", sc.AddCode(code)
```

```
; Example: Pass a VARIANT ByRef to a COM function.
var := ComVar()
var[] := "in value"
sc.Run("Example", var.ref)
MsgBox % var[]
```

```
; ComVar: Creates an object which can be used to
pass a value ByRef.
; ComVar[] retrieves the value.
; ComVar[] := Val sets the value.
; ComVar.ref retrieves a ByRef object for
passing to a COM function.
ComVar(Type=0xC)
{
    static base := { __Get: "ComVarGet", __Set:
"ComVarSet", __Delete: "ComVarDel" }
    ; Create an array of 1 VARIANT. This method
allows built-in code to take
    ; care of all conversions between VARIANT and
AutoHotkey internal types.
    arr := ComObjArray(Type, 1)
    ; Lock the array and retrieve a pointer to the
VARIANT.
    DllCall("oleaut32\SafeArrayAccessData", "ptr",
ComObjValue(arr), "ptr*", arr_data)
    ; Store the array and an object which can be
used to pass the VARIANT ByRef.
    return { ref: ComObject(0x4000|Type,
arr_data), _: arr, base: base }
}
```

```
ComVarGet(cv, p*) { ; Called when script accesses
an unknown field.
    if p.MaxIndex() = "" ; No name/parameters,
i.e. cv[]
        return cv._[0]
```

```
}
```

```
ComVarSet(cv, v, p*) { ; Called when script sets  
an unknown field.
```

```
    if p.MaxIndex() = "" ; No name/parameters,  
i.e. cv[]:=v
```

```
        return cv._[0] := v
```

```
}
```

```
ComVarDel(cv) { ; Called when the object is being  
freed.
```

```
    ; This must be done to allow the internal  
array to be freed.
```

```
    DllCall("oleaut32\SafeArrayUnaccessData",  
"ptr", ComObjValue(cv._))
```

```
}
```

ComObjArray() [v1.0.91+]

Creates a SafeArray for use with COM.

```
ArrayObj := ComObjArray(VarType, Count1 [, Count2,  
... Count8])
```

Parameters

VarType

The base type of the array (the VARTYPE of each element of the array). The VARTYPE is restricted to a subset of the variant types. Neither the VT_ARRAY nor the VT_BYREF flag can be set. VT_EMPTY and VT_NULL are not valid base types for the array. All other types are legal. See [ComObjType](#) for a list of possible values.

CountN

The size of each dimension. Arrays containing up to 8 dimensions are supported.

ArrayObj

A wrapper object containing the SafeArray.

Methods

Array wrapper objects support the following methods:

Array.MaxIndex(n): Returns the upper bound of the *n*th dimension. If *n* is omitted, it defaults to 1.

Array.MinIndex(n): Returns the lower bound of the *n*th dimension. If *n* is omitted, it defaults to 1.

Array.Clone() [v1.0.96.00+]: Returns a copy of the array.

`Array._NewEnum()` [v1.0.96.00+]: Not typically called by script; allows [for-loops](#) to be used with `SafeArrays`.

General Remarks

Array wrapper objects may also be returned by COM methods and [ComObjActive](#). Scripts may determine if a value is an array as follows:

```
if ComObjType(obj) & 0x2000
    MsgBox % "Array subtype: " . ComObjType(obj)
    & 0xffff
else
    MsgBox Not an array.
```

Arrays with up to 8 dimensions are supported.

[v1.0.96.00+]: Since `SafeArrays` are not designed to support multiple references, when one `SafeArray` is assigned to an element of another `SafeArray`, a separate copy is created. However, this only occurs if the wrapper object has the `F_OWNVALUE` flag, which indicates it is responsible for destroying the array. This flag can be removed by using [ComObjFlags](#).

[v1.1.17.00+]: When a function or method called by a COM client returns a `SafeArray` with the `F_OWNVALUE` flag, a copy is created and returned instead, as the original `SafeArray` is automatically destroyed.

Related

[ComObjType](#), [ComObjValue](#), [ComObjActive](#), [ComObjFlags](#), [Array Manipulation Functions \(MSDN\)](#)

Examples

```
; Example #1: Simple usage.
```

```
arr := ComObjArray(VT_VARIANT:=12, 3)
```

```
arr[0] := "Auto"  
arr[1] := "Hot"  
arr[2] := "key"  
Loop % arr.MaxIndex() + 1  
    t := arr[A_Index-1]  
MsgBox % t
```

```
; Example #2: Multiple dimensions.
```

```
arr := ComObjArray(VT_VARIANT:=12, 3, 4)
```

```
; Get the number of dimensions:
```

```
dim := DllCall("oleaut32\SafeArrayGetDim", "ptr",  
ComObjValue(arr))
```

```
; Get the bounds of each dimension:
```

```
Loop %dim%  
    dims .= arr.MinIndex(A_Index) " .. "  
arr.MaxIndex(A_Index) "`n"  
MsgBox %dims%
```

```
; Simple usage:
```

```
Loop 3 {  
    x := A_Index-1  
    Loop 4 {  
        y := A_Index-1  
        arr[x, y] := x * y  
    }  
}  
MsgBox % arr[2, 3]
```

ComObjConnect() [AHK_L 53+]

Connects the object's event sources to functions with a given prefix.

```
ComObjConnect(ComObject [, Prefix])
```

Parameters

ComObject

An object which raises events.

If the object does not support the `IConnectionPointContainer` interface or type information about the object's class cannot be retrieved, an error message is shown. This can be suppressed or handled with [ComObjError](#) or `try/catch`.

[v1.1.22+]: The `IProvideClassInfo` interface is used to retrieve type information about the object's class if the object supports it. Otherwise, `ComObjConnect` attempts to retrieve type information via the object's `IDispatch` interface, which may be unreliable.

Prefix

A string to prefix to the event name to determine which function to call when an event occurs.

If omitted, the object is "disconnected"; that is, the script will no longer receive notification of its events.

[v1.1.01+]: This parameter can be an object defined by the script. When an event is raised, the corresponding method is called. The first parameter, which is usually the hidden `this` parameter, refers to the script-defined object, not the COM object. To catch all events without defining a method for each one, define a [__Call meta-function](#).

Usage

To make effective use of `ComObjConnect`, you must first write functions in the script to handle any events of interest. Such functions, or "event-handlers," have the following structure:

```
PrefixEventName( [Params..., ComObject] )  
{  
    ... event-handling code ...  
    return ReturnValue  
}
```

Prefix is a prefix of your choosing, while **EventName** is the name of whatever event the function should handle.

Params corresponds to whatever parameters the event has. If the event has no parameters, *Params* should be omitted entirely. *ComObject* is optional, and can only be used if the correct number of *Params* are defined; it contains a reference to the original wrapper object which was passed to `ComObjConnect`. "ComObject" should be replaced with a name more meaningful in the context of your script.

Note that event handlers may have return values. To return a COM-specific type of value, use `ComObject(type, value)`. For example, `return ComObject(0, 0)` returns a variant of type `VT_EMPTY`, which is equivalent to returning `undefined` (or not returning) from a JavaScript function.

Call `ComObjConnect(yourObject, "Prefix")` to enable event-handling.

Call `ComObjConnect(yourObject)` to disconnect the object (stop handling events).

If the number of parameters is not known, a [variadic function](#) can be used.

Remarks

On failure, the function may throw an exception, exit the script or simply return,

depending on the current [ComObjError](#) setting and [other factors](#).

Related

[ComObjCreate](#), [ComObjGet](#), [ComObjActive](#), [ComObjError](#),
[WScript.ConnectObject](#) (MSDN)

Examples

```
ie := ComObjCreate("InternetExplorer.Application")

; Connects events to corresponding script
; functions with the prefix "IE_".
ComObjConnect(ie, "IE_")

ie.Visible := true ; This is known to work
; incorrectly on IE7.
ie.Navigate("https://autohotkey.com/")
#Persistent

IE_DocumentComplete(ieEventParam, url,
ieFinalParam) {
    global ie
    if (ie != ieEventParam)
        s .= "First parameter is a new wrapper
object.`n"
    if (ie == ieFinalParam)
        s .= "Final parameter is the original
wrapper object.`n"
    if ((disp1:=ComObjUnwrap(ieEventParam)) ==
(disp2:=ComObjUnwrap(ieFinalParam)))
        s .= "Both wrapper objects refer to the
same IDispatch instance.`n"
    ObjRelease(disp1), ObjRelease(disp2)
    MsgBox % s . "Finished loading "
ie.Document.title " @ " url
ie.Quit()
```

```
} ExitApp
```

ComObjCreate() [AHK_L 53+]

Creates a COM object.

```
ComObject := ComObjCreate(CLSID [, IID])
```

Parameters

CLSID

CLSID or human-readable Prog ID of the COM object to create.

IID

[v1.0.96.00+]: The identifier of an interface the object supports.

Return Value

On failure, the function may throw an exception, exit the script or return an empty string, depending on the current [ComObjError](#) setting and [other factors](#).

If an IID is specified, an interface pointer is returned. The script must typically call [ObjRelease](#) when it is finished with the pointer.

Otherwise, a wrapper object usable by script is returned. See [object syntax](#).

Related

[ComObjGet](#), [ComObjActive](#), [ComObjConnect](#), [ComObjArray](#), [ComObjError](#), [ComObjQuery](#), [CreateObject \(MSDN\)](#)

Examples

For a constantly growing list of examples, see the following forum topic:
<http://www.autohotkey.com/forum/topic61509.html>.

```
ie := ComObjCreate("InternetExplorer.Application")
```

```
ie.Visible := true ; This is known to work  
incorrectly on IE7.  
ie.Navigate("https://autohotkey.com/")
```

ComObjGet() [AHK_L 53+]

Returns a reference to an object provided by a COM component.

```
ComObject := ComObjGet(Name)
```

Parameters

Name

The display name of the object to be retrieved. See [MkParseDisplayName \(MSDN\)](#) for more information.

Remarks

On failure, the function may throw an exception, exit the script or return an empty string, depending on the current [ComObjError](#) setting and [other factors](#).

Related

[ComObjCreate](#), [ComObjActive](#), [ComObjConnect](#), [ComObjError](#), [ComObjQuery](#), [CoGetObject \(MSDN\)](#)

Examples

```
; Example: Press Shift+Escape to show the command  
line which was used  
; to launch the active window's process.  
Requires XP or later.  
+Esc::  
    WinGet pid, PID, A  
    ; Get WMI service object.  
    wmi := ComObjGet("winmgmts:")  
    ; Run query to retrieve matching process(es).  
    queryEnum := wmi.ExecQuery("")
```

```
. "Select * from Win32_Process where  
ProcessId=" . pid)  
  ._NewEnum()  
  ; Get first matching process.  
  if queryEnum[process]  
    MsgBox 0, Command line, %  
process.CommandLine  
  else  
    MsgBox Process not found!  
  ; Free all global objects (not necessary when  
using local vars).  
  wmi := queryEnum := process := ""  
return  
; Win32_Process: http://msdn.microsoft.com/en-  
us/library/aa394372.aspx
```

ComObjError() [AHK_L 53+]

Enables or disables notification of COM errors.

```
Enabled := ComObjError([Enable])
```

Parameters

Enable

A boolean value (true or false). Optional.

Enabled

[v1.0.91+]: Receives the setting which was in effect before the function was called.

General Remarks

Notification of COM errors is enabled by default.

COM errors may be raised by [ComObjCreate](#), [ComObjGet](#), [ComObjActive](#), [ComObjConnect](#), [ComObjQuery](#) (if an invalid pointer is detected) or by a method or property of a COM object. Raising a COM error causes one of the following outcomes:

Setting	Outcome
0 (disabled)	The function, method or property returns a default value (typically an empty string), without showing an error dialog.
1 (enabled)	If there is an active Try block, an exception is thrown. Otherwise: An error dialog is shown, giving the user the option to continue the script. If the user chooses "Yes", the function, method or property returns a default value (typically an empty string) and

the script continues. Otherwise, the script exits.

After accessing a COM object, [A_LastError](#) contains the HRESULT code returned by the COM object's [IDispatch::Invoke](#) function. The script may consult [A_LastError](#) within a [Catch](#) block or after the method or property returns, provided that `ComObjError(false)` has been used or the user did not choose to exit the script.

Related

[ComObjCreate](#), [ComObjGet](#), [ComObjActive](#), [ComObjConnect](#), [ComObjQuery](#)

ComObjFlags() [v1.0.96.00+]

Retrieves or changes flags which control a COM wrapper object's behaviour.

```
Flags := ComObjFlags(ComObject [, NewFlags, Mask])
```

Parameters

ComObject

A COM wrapper object.

NewFlags

New values for the flags identified by *Mask*, or flags to add or remove.

Mask

A bitmask of flags to change.

Flags

All of *ComObject*'s flags (after applying *NewFlags*, if specified).

Flags

F_OWNVALUE 1	Currently only affects SafeArrays. If this flag is set, the SafeArray is destroyed when the wrapper object is freed. Since SafeArrays have no reference counting mechanism, if a SafeArray with this flag is assigned to an element of another SafeArray, a separate copy is created.
-----------------	---

General Remarks

If *Mask* is omitted, *NewFlags* specifies the flags to add (if positive) or remove (if negative). For example, `ComObjFlags(obj, -1)` removes the F_OWNVALUE flag. Do not specify any value for *Mask* other than 0 or 1; all other bits are reserved for future use.

Related

[ComObjActive](#), [ComObjArray](#)

Examples

```
; Example: Check for the presence of the
F_OWNVALUE flag.
```

```
arr := ComObjArray(0xC, 1)
if ComObjFlags(arr) & 1
    MsgBox arr will be automatically destroyed.
else
    MsgBox arr will not be automatically
    destroyed.
```

```
; Example: Change array-in-array behaviour.
```

```
arr1 := ComObjArray(0xC, 3)
arr2 := ComObjArray(0xC, 1)
arr2[0] := "original value"
arr1[0] := arr2           ; Assign implicit copy.
ComObjFlags(arr2, -1)    ; Remove F_OWNVALUE.
arr1[1] := arr2           ; Assign original array.
arr1[2] := arr2.Clone()  ; Assign explicit copy.
arr2[0] := "new value"
for arr in arr1
    MsgBox % arr[0]

arr1 := ""
; Not valid since arr2 == arr1[1], which has been
destroyed:
; arr2[0] := "foo"
```

ComObjQuery() [v1.0.96.00+]

Queries a COM object for an interface or service.

```
InterfacePointer := ComObjQuery(ComObject, [SID, ]  
IID)
```

Parameters

ComObject

A COM wrapper object or raw interface pointer.

IID

An interface identifier (GUID) in the form "{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}".

SID

A service identifier in the same form as IID. When omitting this parameter, also omit the comma.

General Remarks

In its two-parameter mode, this function is equivalent to [IUnknown::QueryInterface](#). When SID and IID are both specified, it internally queries for the [IServiceProvider](#) interface, then calls [IServiceProvider::QueryService](#). In either case, the return value is either zero or a pointer to the requested interface. Generally this pointer must be [released](#) when the script is finished with it.

Related

[ObjRelease](#), [ComObjCreate](#), [ComObjGet](#), [ComObjActive](#), [ComObjError](#)

Examples

```
; Example: Determine the class name of an object.
```

```
obj := ComObjCreate("Scripting.Dictionary")
```

```
MsgBox % "Interface name: " ComObjType(obj,  
"name")
```

```
IID_IProvideClassInfo := "{B196B283-BAB4-101A-  
B69C-00AA00341D07}"
```

```
; Request a pointer to the object's  
IProvideClassInfo interface.
```

```
if !(pci := ComObjQuery(obj,  
IID_IProvideClassInfo))
```

```
{
```

```
    MsgBox IProvideClassInfo interface not  
supported.
```

```
    return
```

```
}
```

```
; Call GetClassInfo to retrieve a pointer to the  
ITypeInfo interface.
```

```
DllCall(vtable(pci, 3), "ptr", pci, "ptr*", ti)
```

```
; Call GetDocumentation to get the object's full  
type name.
```

```
DllCall(vtable(ti, 12), "ptr", ti, "int", -1,  
"ptr*", name, "ptr", 0, "ptr", 0, "ptr", 0)
```

```
; Convert the BSTR pointer to a usable string.
```

```
name := StrGet(name, "UTF-16")
```

```
; Release raw interface pointers.
```

```
ObjRelease(ti)
```

```
ObjRelease(pci)
```

```

; Display the type name!
MsgBox % "Class name: " name

vtable(ptr, n) {
    ; NumGet(ptr+0) returns the address of the
    object's virtual function
    ; table (vtable for short). The remainder of
    the expression retrieves
    ; the address of the nth function's address
    from the vtable.
    return NumGet(NumGet(ptr+0), n*A_PtrSize)
}

```

```

; Example: Automate an existing Internet Explorer
window.

```

```

sURL := "https://autohotkey.com/boards/"
if WebBrowser := GetWebBrowser()
    WebBrowser.Navigate(sURL)
return

GetWebBrowser()
{
    ; Get a raw pointer to the document object of
    the top-most IE window.
    static msg := DllCall("RegisterWindowMessage",
    "Str", "WM_HTML_GETOBJECT")
    SendMessage msg, 0, 0, Internet
    Explorer_Server1, ahk_class IEFram
    e
    if ErrorLevel = FAIL
        return ; IE not found.
    lResult := ErrorLevel
    DllCall("oleacc\ObjectFromLresult", "Ptr",
    lResult
    , "Ptr", GUID(IID_IHTMLDocument2, "
    {332C4425-26CB-11D0-B483-00C04FD90119}")
}

```

```

        , "Ptr", 0, "Ptr*", pdoc)

        ; Query for the WebBrowserApp service. In this
particular case,
        ; the SID and IID are the same, but it isn't
always this way.
        static IID_IWebBrowserApp := "{0002DF05-0000-
0000-C000-0000000000046}"
        static SID_SWebBrowserApp :=
IID_IWebBrowserApp
        pweb := ComObjQuery(pdoc, SID_SWebBrowserApp,
IID_IWebBrowserApp)

        ; Release the document object pointer.
ObjRelease(pdoc)

        ; Return the WebBrowser object, wrapped for
usability:
        static VT_DISPATCH := 9, F_OWNVALUE := 1
        return ComObject(VT_DISPATCH, pweb,
F_OWNVALUE)
}

GUID(ByRef GUID, sGUID) ; Converts a string to a
binary GUID and returns its address.
{
    VarSetCapacity(GUID, 16, 0)
    return DllCall("ole32\CLSIDFromString",
"WStr", sGUID, "Ptr", &GUID) >= 0 ? &GUID : ""
}

```

ComObjType() [v1.0.91+]

Retrieves type information from a COM object.

```
VarType := ComObjType(ComObject)
IName   := ComObjType(ComObject, "Name")
IID     := ComObjType(ComObject, "IID")
CName   := ComObjType(ComObject, "Class") ;
Requires [v1.1.26+]
CLSID   := ComObjType(ComObject, "CLSID") ;
Requires [v1.1.26+]
```

Parameters

ComObject

A wrapper object containing a COM object or typed value.

Param2

The second parameter is a string indicating the type information to retrieve.

Return Value

The return value depends on the value of *Param2*:

Param2	Return Value
Omitted	An integer variant type code indicating the type of value contained by the COM wrapper object.
"Name"	The name of the object's default interface.
"IID"	The globally unique identifier (GUID) of the object's default interface.

"Class"	[v1.1.26+]: The object's class name. Note that this is not the same as a Prog ID (a Prog ID is a name used to identify the class in the system registry, or for ComObjCreate).
"CLSID"	[v1.1.26+]: The globally unique identifier (GUID) of the object's class. Classes are often registered by CLSID under the <code>HKCR\CLSID</code> registry key.

An empty string is returned if either parameter is invalid or if the requested type information could not be retrieved.

Variant Type Constants

```

VT_EMPTY      = 0 ; No value
VT_NULL       = 1 ; SQL-style Null
VT_I2         = 2 ; 16-bit signed int
VT_I4         = 3 ; 32-bit signed int
VT_R4         = 4 ; 32-bit floating-point
number
VT_R8         = 5 ; 64-bit floating-point
number
VT_CY         = 6 ; Currency
VT_DATE       = 7 ; Date
VT_BSTR       = 8 ; COM string (Unicode
string with length prefix)
VT_DISPATCH   = 9 ; COM object
VT_ERROR      = 0xA ; Error code (32-bit
integer)
VT_BOOL       = 0xB ; Boolean True (-1) or
False (0)
VT_VARIANT    = 0xC ; VARIANT (must be
combined with VT_ARRAY or VT_BYREF)
VT_UNKNOWN    = 0xD ; IUnknown interface
pointer
VT_DECIMAL    = 0xE ; (not supported)
VT_I1         = 0x10 ; 8-bit signed int
VT_UI1        = 0x11 ; 8-bit unsigned int

```

```

VT_UI2      = 0x12 ; 16-bit unsigned int
VT_UI4      = 0x13 ; 32-bit unsigned int
VT_I8       = 0x14 ; 64-bit signed int
VT_UI8      = 0x15 ; 64-bit unsigned int
VT_INT      = 0x16 ; Signed machine int
VT_UINT     = 0x17 ; Unsigned machine int
VT_RECORD   = 0x24 ; User-defined type --
NOT SUPPORTED
VT_ARRAY    = 0x2000 ; SAFEARRAY
VT_BYREF    = 0x4000 ; Pointer to another type
of value
/*
VT_ARRAY and VT_BYREF are combined with another
value (using bitwise OR)
to specify the exact type. For instance, 0x2003
identifies a SAFEARRAY
of 32-bit signed integers and 0x400C identifies
a pointer to a VARIANT.
*/

```

General Remarks

In most common cases, return values from methods or properties of COM objects are converted to an appropriate data type supported by AutoHotkey. Types which aren't specifically handled are coerced to strings via [VariantChangeType](#); if this fails or if the variant type contains the VT_ARRAY or VT_BYREF flag, an object containing both the value and its type is returned instead.

For any variable *x*, if `ComObjType(x)` returns an integer, *x* contains a COM object wrapper.

If *Param2* is "Name" or "IID", type information is retrieved via the [IDispatch::GetTypeInfo](#) interface method. *ComObject*'s variant type must be VT_DISPATCH.

If *Param2* is "Class" or "CLSID", type information is retrieved via the

[IProvideClassInfo::GetClassInfo](#) interface method. *ComObject*'s variant type must be VT_DISPATCH or VT_UNKNOWN, and the object must implement the IProvideClassInfo interface (some objects do not).

Related

[ComObjValue](#), [ComObjCreate](#), [ComObjGet](#), [ComObjActive](#)

Examples

```
d := ComObjCreate("Scripting.Dictionary")
VarType := ComObjType(d)
IName    := ComObjType(d, "Name")
IID      := ComObjType(d, "IID")
CName    := ComObjType(d, "Class") ; Requires
[v1.1.26+]
CLSID    := ComObjType(d, "CLSID") ; Requires
[v1.1.26+]
MsgBox % "Variant type:`t" VarType
        . "`nInterface name:`t" IName "`nInterface
ID:`t" IID
        . "`nClass name:`t" CName "`nClass ID
(CLSID):`t" CLSID
```

ComObjValue() [v1.0.91+]

Retrieves the value or pointer stored in a COM wrapper object.

```
Value := ComObjValue(ComObject)
```

Parameters

Value

■ A 64-bit signed integer.

ComObject

■ A wrapper object containing a COM object or typed value.

General Remarks

This function is not intended for general use.

Calling `ComObjValue` is equivalent to `variant.l1Val`, where *ComObject* is treated as a [VARIANT structure](#). Any script which uses this function must be aware what [type of value](#) the wrapper object contains and how it should be treated. For instance, if an interface pointer is returned, [Release](#) should not be called, but [AddRef](#) may be required depending on what the script does with the pointer.

Related

[ComObjType](#), [ComObjCreate](#), [ComObjGet](#), [ComObjActive](#)

ObjAddRef() / ObjRelease() [AHK_L 53+]

Increments or decrements an object's [reference count](#).

```
ObjAddRef(Ptr)
ObjRelease(Ptr)
```

Parameters

Ptr

An unmanaged object pointer or COM interface pointer.

Return Value

These functions return the new reference count. This value should be used **only** for debugging purposes.

Related

Reference Counting

Although the following articles discuss reference counting as it applies to COM, they cover some important concepts and rules which generally also apply to AutoHotkey objects: [IUnknown::AddRef](#), [IUnknown::Release](#), [Reference Counting Rules](#).

Examples

See [ComObjConnect](#).

```
obj := Object()

; The following two lines are equivalent:
ptr1 := Object(obj)
```

```
ptr2 := ObjectToPointer(obj)
```

```
ObjectToPointer(obj) {  
    if !IsObject(obj)  
        return ""  
    ptr := &obj  
    ObjAddRef(ptr)  
    return ptr  
}
```

; Each pointer retrieved via Object() or
ObjectToPointer() must be manually released
; to allow the object to be eventually freed and
any memory used by it reclaimed.

```
ObjRelease(ptr2)  
ObjRelease(ptr1)
```

Drive

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). For example: `Drive, Label, C:, Seagate200`.

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

Lock, Drive: Prevents a drive's eject feature from working. For 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). 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: Reverses the above. *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 (to eject other types of media or devices, see the DllCall example at the bottom of this page).

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; for example: `Drive, Eject, D:, 1`.

Drive Eject waits for the ejection or retraction to complete before allowing the script to continue. If the tray is already in the correct state (open or closed), [ErrorLevel](#) is set to 0 (i.e. "no error").

Drive Eject 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, the following hotkey toggles the tray to the opposite state (open or closed):

```
#c::  
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
```

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

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

Remarks

The following is an alternate ejection method that also works on types of media/devices other than CD/DVD:

```
; Update the first line below to match the
desired drive letter (you can ignore all the
other lines below).
DriveLetter = I: ; Set this to the drive letter
you wish to eject.

hVolume := DllCall("CreateFile"
    , Str, "\\.\\" . DriveLetter
    , UInt, 0x80000000 | 0x40000000 ;
GENERIC_READ | GENERIC_WRITE
    , UInt, 0x1 | 0x2 ; FILE_SHARE_READ |
FILE_SHARE_WRITE
    , UInt, 0
    , UInt, 0x3 ; OPEN_EXISTING
    , UInt, 0, UInt, 0)
if hVolume <> -1
{
    DllCall("DeviceIoControl"
        , UInt, hVolume
        , UInt, 0x2D4808 ;
IOCTL_STORAGE_EJECT_MEDIA
        , UInt, 0, UInt, 0, UInt, 0, UInt, 0
        , UIntP, dwBytesReturned ; Unused.
        , UInt, 0)
    DllCall("CloseHandle", UInt, hVolume)
}
```

Related

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

Example

```
Drive, Label, D:, BackupDrive  
Drive, Eject,, 1 ; Retract (close) the tray of the  
default CD or DVD drive.
```

DriveGet

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 *Cmd*.

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. For 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, CDFS (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]: Retrieves the media status of a CD or DVD drive, where *Drive* is the drive letter followed by a colon (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 strings:

not ready	The drive is not ready to be accessed, perhaps due to being engaged in a write operation. Known limitation: "not ready" also occurs when the drive contains a DVD rather than a CD.
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 CD 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.

To eject or retract the tray, see the [Drive](#) command.

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[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, c:\
```

FileAppend

Writes text to the end of a file (first creating the file, if necessary).

```
FileAppend [, Text, Filename, Encoding]
```

Parameters

Text

The text to append to the file. This text may include linefeed characters (`\n`) to start new lines. In addition, a single long line can be broken up into several shorter ones by means of a [continuation section](#).

If *Text* is blank, *Filename* will be created as an empty file (but if the file already exists, its modification time will be updated).

If *Text* is `%ClipboardAll%` or a variable that was previously assigned the value of `ClipboardAll`, *Filename* will be unconditionally overwritten with the entire contents of the clipboard (i.e. [FileDelete](#) is not necessary).

Filename

The name of the file to be appended, which is assumed to be in `%A_WorkingDir%` if an absolute path isn't specified.

End of line (EOL) translation: To disable EOL translation, prepend an asterisk to the filename. This causes each linefeed character (`\n`) to be written as a single linefeed (LF) rather than the Windows standard of CR+LF. For example: `*C:\My Unix File.txt`.

If the file is not already open (due to being inside a [file-reading loop](#)), EOL translation is automatically disabled if *Text* contains any carriage return and linefeed pairs (`\r\n`). In other words, the asterisk option described in the previous paragraph is put into effect automatically. However, specifying the asterisk when *Text* contains `\r\n` improves performance because the program does not need to scan *Text* for `\r\n`.

Standard Output (stdout): Specifying an asterisk (*) for *Filename* causes *Text* to be sent to standard output (stdout). Such text can be redirected to a file, piped to another EXE, or captured by [fancy text editors](#). For example, the following would be valid if typed at a command prompt:

```
"%ProgramFiles%\AutoHotkey\AutoHotkey.exe"  
"My Script.ahk" >"Error Log.txt"
```

However, text sent to stdout will not appear at the command prompt it was launched from. This can be worked around by piping a script's output to another command or program. For example:

```
"%ProgramFiles%\AutoHotkey\AutoHotkey.exe"  
"My Script.ahk" |more
```

```
For /F "tokens=*" %L in  
('" "%ProgramFiles%\AutoHotkey\AutoHotkey.exe"  
e" "My Script .ahk"'') do @Echo %L
```

[v1.1.20+]: Specifying two asterisks (**) for *Filename* causes *Text* to be sent to the stderr stream.

Encoding

[AHK_L 42+]: Overrides the default encoding set by [FileEncoding](#), where *Encoding* follows the same format.

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

[A_LastError](#) is set to the result of the operating system's GetLastError() function.

Remarks

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

The target file is automatically closed after the text is appended (except when FileAppend is used in its single-parameter mode inside a [file-reading/writing loop](#)).

[AHK_L 42+]: [FileOpen\(\)](#) in append mode provides more control than FileAppend and allows the file to be kept open rather than opening and closing it each time. Once a file is opened in append mode, use `file.write(string)` to append the string. File objects also support binary I/O via [RawWrite/RawRead](#) or [WriteNum/ReadNum](#), whereas FileAppend supports only text.

Related

[FileOpen/File Object](#), [FileRead](#), [file-reading loop](#), [FileReadLine](#), [IniWrite](#), [FileDelete](#), [OutputDebug](#), [continuation sections](#)

Example

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

```
; The following example uses a continuation
section to enhance readability and
maintainability:
```

```
FileAppend,
```

```
(
```

```
A line of text.
```

```
By default, the hard carriage return (Enter)
between the previous line and this one will be
written to the file.
```

```
    This line is indented with a tab; by default,
that tab will also be written to the file.
```

```
Variable references such as %Var% are expanded by
```

```
default.  
) , C:\My File.txt
```

```
; The following example demonstrates how to  
automate FTP uploading using the operating  
; system's built-in FTP command. This script has  
been tested on Windows XP.
```

```
FTPCommandFile = %A_ScriptDir%\FTPCommands.txt  
FTPLogFile = %A_ScriptDir%\FTPLog.txt  
FileDelete %FTPCommandFile% ; In case previous  
run was terminated prematurely.
```

```
FileAppend, ; The comma is required in this case.
```

```
(  
open host.domain.com  
username  
password  
binary  
cd htdocs  
put %VarContainingNameOfTargetFile%  
delete SomeOtherFile.htm  
rename OldFileName.htm NewFileName.htm  
ls -l  
quit  
) , %FTPCommandFile%
```

```
RunWait %ComSpec% /c ftp.exe -s:"%FTPCommandFile%"  
>"%FTPLogFile%"
```

```
FileDelete %FTPCommandFile% ; Delete for security  
reasons.
```

```
Run %FTPLogFile% ; Display the log for review.
```

FileCopy

Copies one or more files.

```
FileCopy, SourcePattern, DestPattern [, Flag]
```

Parameters

SourcePattern

The name of a single file or folder, 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 copy -- retaining the existing file name(s) -- specify only the folder name as shown in these functionally identical examples:

```
FileCopy, C:\*.txt, C:\My Folder
```

```
FileCopy, 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

This parameter can be an [expression](#), even one that evaluates to true or false (since true and false are stored internally as 1 and 0).

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

In either case, if the source file is a single file (no wildcards) and it does not exist, [ErrorLevel](#) is set to 0. To detect this condition, use [IfExist](#) or [FileExist\(\)](#) on the source file prior to copying it.

Unlike [FileMove](#), copying a file onto itself is always counted as an error, even if the overwrite mode is in effect.

If files were found, [A_LastError](#) is set to 0 (zero) or the result of the operating system's [GetLastError\(\)](#) function immediately after the last failure. Otherwise [A_LastError](#) contains an error code that might indicate why no files were found.

Remarks

[FileCopy](#) copies files only. To instead copy the contents of a folder (all its files and subfolders), see the examples section below. To copy a single folder (including its subfolders), use [FileCopyDir](#).

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

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.

```
; The following example copies all files and
folders inside a folder to a different folder:
ErrorCount := CopyFilesAndFolders("C:\My
Folder\*.*", "D:\Folder to receive all files &
folders")
if ErrorCount <> 0
    MsgBox %ErrorCount% files/folders could not be
copied.
```

```
CopyFilesAndFolders(SourcePattern,
DestinationFolder, DoOverwrite = false)
; Copies all files and folders matching
SourcePattern into the folder named
DestinationFolder and
; returns the number of files/folders that could
not be copied.
{
    ; First copy all the files (but not the
folders):
    FileCopy, %SourcePattern%,
%DestinationFolder%, %DoOverwrite%
    ErrorCount := ErrorLevel
    ; Now copy all the folders:
    Loop, %SourcePattern%, 2 ; 2 means "retrieve
folders only".
    {
        FileCopyDir, %A_LoopFileFullPath%,
%DestinationFolder%\%A_LoopFileName%,
%DoOverwrite%
        ErrorCount += ErrorLevel
        if ErrorLevel ; Report each problem
folder by name.
```

```
        MsgBox Could not copy
%A_LoopFileFullPath% into %DestinationFolder%.
    }
    return ErrorCount
}
```

FileCopyDir

Copies a folder along with all its sub-folders 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 directory (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. The operation will fail and have no effect if *Dest* already exists as a file or directory.

1: Overwrite existing files. However, any files or subfolders inside *Dest* that do not have a counterpart in *Source* will not be deleted.

This parameter can be an [expression](#), even one that evaluates to true or false (since true and false are stored internally as 1 and 0).

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[ErrorLevel](#) is set to 1 if there was a problem or 0 otherwise. However, if the source directory contains any saved webpages consisting of a *PageName.htm* file and a corresponding directory named *PageName_files*, an error may be indicated even when the copy is successful.

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.

[FileCopyDir](#) copies a single folder. To instead copy the contents of a folder (all its files and subfolders), see the examples section of [FileCopy](#).

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?
IfMsgBox, No
    return
SplitPath, SourceFolder, SourceFolderName ;
Extract only the folder name from its full path.
FileCopyDir, %SourceFolder%,
%TargetFolder%\%SourceFolderName%
if ErrorLevel
    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

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

`A_LastError` is set to the result of the operating system's `GetLastError()` function.

Remarks

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

Related

[FileRemoveDir](#)

Example

```
FileCreateDir, C:\Test1\My Images\Folder2
```

FileCreateShortcut

Creates a shortcut (.lnk) file.

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

Parameters

Target

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.

LinkFile

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.

WorkingDir

Directory that will become *Target'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.

Args

Parameters that will be passed to *Target* when it is launched. Separate parameters with spaces. If a parameter contains spaces, enclose it in double quotes.

Description

Comments that describe the shortcut (used by the OS to display a tooltip,

etc.)

IconFile

The full path and name of the icon to be displayed for *LinkFile*. It must either be an ico file or the very first icon of an EXE or DLL.

ShortcutKey

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 s. For example, if the letter B is specified for this parameter, the shortcut key will be 

IconNumber

To use an icon in *IconFile* other than the first, specify that number here (can be an [expression](#)). For example, 2 is the second icon.

RunState

To have *Target* launched minimized or maximized, specify one of the following digits:

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

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[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.

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, %A_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 `C:\Temp*.tmp`. *FilePattern* is assumed to be in `%A_WorkingDir%` if an absolute path isn't specified.

To remove an entire folder, along with all its sub-folders and files, use [FileRemoveDir](#).

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

`ErrorLevel` is set to the number of files that failed to be deleted (if any) or 0 otherwise. Deleting a wildcard pattern such as `*.tmp` is considered a success even if it does not match any files; thus `ErrorLevel` is set to 0.

If files were found, `A_LastError` is set to 0 (zero) or the result of the operating system's `GetLastError()` function immediately after the last failure. Otherwise `A_LastError` contains an error code that might indicate why no files were found.

Remarks

To delete a read-only file, first remove the read-only attribute. For example: `FileSetAttrib, -R, C:\My File.txt`.

Related

FileRecycle, FileRemoveDir, FileCopy, FileMove

Example

```
FileDelete, C:\temp files\*.tmp
```

FileEncoding [AHK_L 42+]

Sets the default encoding for [FileRead](#), [FileReadLine](#), [Loop Read](#), [FileAppend](#), and [FileOpen](#).

```
FileEncoding [, Encoding]
```

Encoding can be one of the following values:

- `UTF-8` : Unicode UTF-8, equivalent to CP65001.
- `UTF-16` : Unicode UTF-16 with little endian byte order, equivalent to CP1200.
- `UTF-8-RAW` or `UTF-16-RAW` : As above, but no byte order mark is written when a new file is created.
- `CPnnn` : a code page with numeric identifier *nnn*. See [Code Page Identifiers](#).
- Empty or omitted: the system default ANSI code page, which is also the default setting.

Remarks

`A_FileEncoding` contains the current setting.

The default encoding is not used if a UTF-8 or UTF-16 byte order mark is present in the file, unless the file is being opened with write-only access (i.e. the previous contents of the file are being discarded).

Related

[FileOpen](#), [StrPut/StrGet](#), [Script Compatibility](#)

FileExist()

Checks for the existence of a file or folder and returns its attributes.

```
AttributeString := FileExist(FilePattern)
```

Parameters

FilePattern

The path, filename, or file pattern to check. *FilePattern* is assumed to be in `%A_WorkingDir%` if an absolute path isn't specified.

Return Value

This function returns the attribute string (a subset of "RASHNDOCT") of the first matching file or folder:

R = READONLY

A = ARCHIVE

S = SYSTEM

H = HIDDEN

N = NORMAL

D = DIRECTORY

O = OFFLINE

C = COMPRESSED

T = TEMPORARY

If the file has no attributes (rare), "X" is returned. If no file is found, an empty string is returned.

Remarks

This function is a combination of [IfExist](#) and [FileGetAttrib](#).

Since an empty string is seen as "false", the function's return value can always be

used as a quasi-boolean value. For example, the statement `if FileExist("C:\My File.txt")` would be true if the file exists and false otherwise. Similarly, the statement `if InStr(FileExist("C:\My Folder"), "D")` would be true only if the file exists *and* is a directory.

Since *FilePattern* may contain wildcards, `FileExist` may be unsuitable for validating a file path which is to be used with another function or program. For example, `FileExist("*.txt")` may return attributes even though `*.txt` is not a valid filename. In such cases, [FileGetAttrib](#) is preferred.

Related

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

Examples

```
if FileExist("D:\")
    MsgBox, The drive exists.
if FileExist("D:\Docs\*.txt")
    MsgBox, At least one .txt file exists.
if !FileExist("C:\Temp\FlagFile.txt")
    MsgBox, The target file does not exist.
```

; The following example shows how to check the file for a particular attribute:

```
if InStr(FileExist("C:\My File.txt"), "H")
    MsgBox The file is hidden.
```

FileGetAttrib

Reports whether a file or folder is read-only, hidden, etc.

```
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. Unlike [FileExist](#), this must be a true filename, not a pattern.

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

[A_LastError](#) is set to the result of the operating system's `GetLastError()` function.

Remarks

The string returned will contain a subset of the letters in the string "RASHNDOCT":

R = READONLY

A = ARCHIVE

S = SYSTEM

H = HIDDEN

N = NORMAL
D = DIRECTORY
O = OFFLINE
C = COMPRESSED
T = TEMPORARY

To check if a particular attribute is present in the retrieved string, following this example:

```
FileGetAttrib, Attributes, C:\My File.txt  
IfInString, Attributes, H  
    MsgBox The file is hidden.
```

On a related note, to retrieve a file's 8.3 short name, follow this example:

```
Loop, C:\My Documents\Address List.txt  
    ShortPathName = %A_LoopFileShortPath% ;  
Will yield something similar to  
C:\MYDOCU~1\ADDRES~1.txt
```

A similar method can be used to get the long name of an 8.3 short name.

Related

[FileExist\(\)](#), [FileSetAttrib](#), [FileGetTime](#), [FileSetTime](#), [FileGetSize](#),
[FileGetVersion](#), [File-loop](#)

Example

```
FileGetAttrib, OutputVar, C:\New Folder
```

FileGetShortcut

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). For example:
C:\WINDOWS\system32\notepad.exe

OutDir

Name of the variable in which to store the shortcut's working directory. For example: C:\My Documents. If environment variables such as `%WinDir%` are present in the string, one way to resolve them is via [StringReplace](#). For example: `StringReplace, OutDir, OutDir, `%WinDir`%, %A_WinDir%`.

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

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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, 32,, Pick a shortcut to
```

```
analyze., Shortcuts (*.lnk)
if file =
    return
FileGetShortcut, %file%, OutTarget, OutDir,
OutArgs, OutDesc, OutIcon, OutIconNum, OutRunState
MsgBox
%OutTarget%`n%OutDir%`n%OutArgs%`n%OutDesc%`n%OutI
con%`n%OutIconNum%`n%OutRunState%
```

FileGetSize

Retrieves the size of a file.

```
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

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

`A_LastError` is set to the result of the operating system's `GetLastError()` function.

Remarks

Files of any size are supported, even those over 4 gigabytes, and 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).

To calculate the size of folder, including all its files, follow this example:

```
SetBatchLines, -1 ; Make the operation run at
maximum speed.
FolderSize = 0
FileSelectFolder, WhichFolder ; Ask the user to
pick a folder.
Loop, %WhichFolder%\*.*, , 1
    FolderSize += %A_LoopFileSize%
MsgBox Size of %WhichFolder% is %FolderSize%
bytes.
```

Related

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

Example

```
FileGetSize, OutputVar, C:\My Documents\test.doc
; Retrieve the size in bytes.
FileGetSize, OutputVar, C:\My Documents\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 or folder, 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 parameter is omitted)

C = Creation time

A = Last access time

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

[A_LastError](#) is set to the result of the operating system's `GetLastError()` function.

Remarks

See [YYYYMMDDHH24MISS](#) for an explanation 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 version number/string.

Filename

The name of the target file. If a full path is not specified, this function uses the search sequence specified by the system [LoadLibrary](#) function. If omitted, the current file of the innermost enclosing [File-Loop](#) will be used instead.

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

[A_LastError](#) is set to the result of the operating system's `GetLastError()` function.

Remarks

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

Related

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

Example

```
FileGetVersion, version, C:\My Application.exe  
FileGetVersion, version,  
%A_ProgramFiles%\AutoHotkey\AutoHotkey.exe
```

FileInstall

Includes the specified file inside the [compiled version](#) of the script.

```
FileInstall, Source, Dest [, Flag]
```

Parameters

Source

The name of the file to be added to the compiled EXE. The file is assumed to be in (or relative to) the script's own directory if an absolute path isn't specified.

The file name **must not** contain double quotes, variable references (e.g. %A_ProgramFiles%), or wildcards. In addition, any special characters such as literal percent signs and commas must be [escaped](#) (just like in the parameters of all other commands). Finally, this parameter must be listed to the right of the FileInstall command (that is, not on a [continuation line](#) beneath it).

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. The destination directory must already exist. 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

This parameter can be an [expression](#), even one that evaluates to true or false (since true and false are stored internally as 1 and 0).

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[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 is added **during script compilation**. When the compiled script is executed and the same "FileInstall" command is 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 helps the testing of scripts that will eventually be compiled.

Related

[FileCopy](#), [#Include](#)

Example

```
FileInstall, C:\My Documents\My File.txt,  
%A_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

This parameter can be an [expression](#), even one that evaluates to true or false (since true and false are stored internally as 1 and 0).

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[ErrorLevel](#) is set to the number of files that could not be moved due to an error, or 0 otherwise. However, if the source file is a single file (no wildcards) and it does not exist, [ErrorLevel](#) is set to 0. To detect this condition, use [IfExist](#) or [FileExist\(\)](#) on the source file prior to moving it.

Unlike [FileCopy](#), moving a file onto itself is always considered successful, even if the overwrite mode is not in effect.

If files were found, [A_LastError](#) is set to 0 (zero) or the result of the operating system's `GetLastError()` function immediately after the last failure. Otherwise [A_LastError](#) contains an error code that might indicate why no files were found.

Remarks

`FileMove` moves files only. To instead move the contents of a folder (all its files and subfolders), see the examples section below. To move or rename a single folder, use [FileMoveDir](#).

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

Although this command is capable of moving files to a different volume, the operation will take longer than a same-volume move. This is because a same-volume move is similar to a rename, and therefore much faster.

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.
```

```
; The following example moves all files and  
folders inside a folder to a different folder:  
ErrorCount := MoveFilesAndFolders("C:\My  
Folder\*.*", "D:\Folder to receive all files &  
folders")  
if ErrorCount <> 0  
    MsgBox %ErrorCount% files/folders could not be  
moved.
```

```
MoveFilesAndFolders(SourcePattern,  
DestinationFolder, DoOverwrite = false)  
; Moves all files and folders matching  
SourcePattern into the folder named  
DestinationFolder and  
; returns the number of files/folders that could  
not be moved. This function requires [v1.0.38+]  
; because it uses FileMoveDir's mode 2.  
{  
    if DoOverwrite = 1  
        DoOverwrite = 2 ; See FileMoveDir for  
description of mode 2 vs. 1.  
        ; First move all the files (but not the  
folders):  
        FileMove, %SourcePattern%,  
%DestinationFolder%, %DoOverwrite%  
        ErrorCount := ErrorLevel  
        ; Now move all the folders:  
        Loop, %SourcePattern%, 2 ; 2 means "retrieve  
folders only".  
    }
```

```
        FileMoveDir, %A_LoopFileFullPath%,
%DestinationFolder%\%A_LoopFileName%,
%DoOverwrite%
        ErrorCount += ErrorLevel
        if ErrorLevel ; Report each problem
folder by name.
        MsgBox Could not move
%A_LoopFileFullPath% into %DestinationFolder%.
    }
    return ErrorCount
}
```

FileMoveDir

Moves a folder along with all its sub-folders and files. It can also rename a folder.

```
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

The new path and name of the 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`.

Note: *Dest* is the actual path and name that the directory will have after it is moved; it is *not* the directory into which *Source* is moved (except for the known limitation mentioned below).

Flag

(options) Specify one of the following single characters:

0 (default): Do not overwrite existing files. The operation will fail if *Dest* already exists as a file or directory.

1: Overwrite existing files. However, any files or subfolders inside *Dest* that do not have a counterpart in *Source* will not be deleted. **Known limitation:** If *Dest* already exists as a folder and it is on the same volume as *Source*, *Source* will be moved into it rather than overwriting it. To avoid this, see the next option.

2: The same as mode 1 above except that the limitation is absent.

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 when *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.

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

Remarks

FileMoveDir moves a single folder to a new location. To instead move the contents of a folder (all its files and subfolders), see the examples section of [FileMove](#).

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

Related

[FileCopyDir](#), [FileCopy](#), [FileMove](#), [FileDelete](#), [File-loops](#), [FileSelectFolder](#), [SplitPath](#)

Example

```
FileMoveDir, C:\My Folder, D:\My Folder ; Move to  
a new drive.  
FileMoveDir, C:\My Folder, C:\My Folder (renamed),  
R ; Simple rename.  
FileMoveDir, C:\My Folder, C:\New Location\My
```

Folder, R ; Folders can be "renamed into" another location as long as it's on the same volume.

FileOpen [AHK_L 42+]

Opens a file.

```
file := FileOpen(Filename, Flags [, Encoding])
```

Parameters

Filename

The path of the file to open, which is assumed to be in [A_WorkingDir](#) if an absolute path isn't specified.

[v1.1.17+]: Specify an asterisk (or two) as shown below to open the standard input/output/error stream:

```
FileOpen("*", "r")    ; for stdin
FileOpen("*", "w")    ; for stdout
FileOpen("***", "w")  ; for stderr
```

Flags

Either [in AHK_L 54+] a string of characters indicating the desired access mode followed by other options (with optional spaces or tabs in between); or [in AHK_L 42+] a combination (sum) of numeric flags. Supported values are described in the tables below.

Encoding

The code page to use for text I/O if the file does not contain a UTF-8 or UTF-16 [byte order mark](#), or if the `h` (handle) flag is used. If omitted, the current value of [A_FileEncoding](#) is used.

Flags

Access modes (mutually-exclusive)

Flag	Number	Description
r	0	<i>Read</i> : Fails if the file doesn't exist.
w	1	<i>Write</i> : Creates a new file, overwriting any existing file .
a	2	<i>Append</i> : Creates a new file if the file didn't exist, otherwise moves the file pointer to the end of the file.
rw	3	<i>Read/Write</i> : Creates a new file if the file didn't exist.
h		Indicates that <i>Filename</i> is a file handle to wrap in an object. Sharing mode flags are ignored and the file or stream represented by the handle is not checked for a byte order mark. The file handle is not closed automatically when the file object is destroyed and calling Close has no effect. Note that Seek , Tell and Length should not be used if <i>Filename</i> is a handle to a nonseeking device such as a pipe or a communications device.

Sharing mode flags

Flag	Number	Description
-rwd		Locks the file for read, write and/or delete access. Any combination of <code>r</code> , <code>w</code> and <code>d</code> may be used. Specifying <code>-</code> is the same as specifying <code>-rwd</code> . If omitted entirely, the default is to share all access.
	0	If <i>Flags</i> is numeric, the absence of sharing mode flags causes the file to be locked.
	0x100	Shares <i>read</i> access.
	0x200	Shares <i>write</i> access.
	0x400	Shares <i>delete</i> access.

End of line (EOL) options

--

Flag	Number	Description
<code>`n`</code>	4	Replace <code>`r`n`</code> with <code>`n`</code> when reading and <code>`n`</code> with <code>`r`n`</code> when writing.
<code>`r`</code>	8	Replace standalone <code>`r`</code> with <code>`n`</code> when reading.

Return Value

If the file is opened successfully, the return value is a [File object](#).

If the function fails, the return value is 0 and [in AHK_L 54+] [A_LastError](#) contains an error code.

Use `if file` or `IsObject(file)` to check if the function succeeded.

Remarks

When a UTF-8 or UTF-16 file is created, a byte order mark is written to the file **unless** *Encoding* (or [A_FileEncoding](#) if *Encoding* is omitted) contains `UTF-8-RAW` or `UTF-16-RAW`.

When a file containing a UTF-8 or UTF-16 byte order mark (BOM) is opened with read access, the BOM is excluded from the output by positioning the file pointer after it. Therefore, `File.Position` may report 3 or 2 immediately after opening the file.

Related

[FileEncoding](#), [File Object](#), [FileRead](#)

Examples

```
; Example: This is a working script that writes  
some text to a file then reads it back into  
memory.  
; It provides the same functionality as this  
DllCall-example.
```

```

FileSelectFile, FileName, S16,, Create a new file:
if (FileName = "")
    return
file := FileOpen(FileName, "w")
if !IsObject(file)
{
    MsgBox Can't open "%FileName%" for writing.
    return
}
TestString := "This is a test string.`r`n" ; When
writing a file this way, use `r`n rather than `n
to start a new line.
file.Write(TestString)
file.Close()

; Now that the file was written, read its contents
back into memory.
file := FileOpen(FileName, "r-d") ; read the file
("r"), share all access except for delete ("-d")
if !IsObject(file)
{
    MsgBox Can't open "%FileName%" for reading.
    return
}
CharsToRead := StrLen(TestString)
TestString := file.Read(CharsToRead)
file.Close()
MsgBox The following string was read from the
file: %TestString%

```

```

; Open the script in read-only mode and read its
first line:
file := FileOpen(A_ScriptFullPath, "r")
MsgBox % file.ReadLine()

```

```
; Open a console window for this demonstration:
DllCall("AllocConsole")
; Open the application's stdin/stdout streams in
newline-translated mode.
stdin := FileOpen("*", "r `n") ; Requires
[v1.1.17+]
stdout := FileOpen("*", "w `n")
; For older versions:
;   stdin := FileOpen(DllCall("GetStdHandle",
"int", -10, "ptr"), "h `n")
;   stdout := FileOpen(DllCall("GetStdHandle",
"int", -11, "ptr"), "h `n")
stdout.Write("Enter your query.`n\> ")
stdout.Read(0) ; Flush the write buffer.
query := RTrim(stdin.ReadLine(), "`n")
stdout.WriteLine("Your query was '" query "'. Have
a nice day.")
stdout.Read(0) ; Flush the write buffer.
Sleep 5000
```

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. Windows and Unix formats are supported; that is, the file's lines may end in either carriage return and linefeed (`\r\n`) or just linefeed (`\n`).

LineNum

Which line to read (1 is the first, 2 the second, and so on). This can be an [expression](#).

If the specified line number is greater than the number of lines in the file, [ErrorLevel](#) is set to 1 and *OutputVar* is not changed. This also happens when the specified line number is the last line in the file but that line is blank and does not end in a newline/CRLF.

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[ErrorLevel](#) is set to 0 upon success. Otherwise it is set to 1 and the original contents of *OutputVar* are not changed.

[A_LastError](#) is set to the result of the operating system's `GetLastError()` function.

Remarks

It is strongly recommended to use this command only for small files, or in cases where only a single line of text is needed. To scan and process a large number of lines (one by one), use a [file-reading loop](#) for best performance. To read an entire file into a variable, use [FileRead](#).

Although any leading and trailing tabs and spaces present in the line will be written to *OutputVar*, the linefeed 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 on (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 [FileRead](#) or a [file-reading loop](#) instead).

Related

[FileOpen/File.ReadLine\(\)](#), [FileRead](#), [FileAppend](#), [File-reading loop](#), [IniRead](#)

Example

```
Loop
{
    FileReadLine, line, C:\My
Documents\ContactList.txt, %A_Index%
    if ErrorLevel
        break
    MsgBox, 4, , Line #%A_Index% is "%line%".
Continue?
    IfMsgBox, No
        return
}
MsgBox, The end of the file has been reached or
there was a problem.
return
```

FileRead

Reads a file's contents into a [variable](#).

`FileRead, OutputVar, Filename`

Parameters

OutputVar

The name of the [variable](#) in which to store the retrieved data. *OutputVar* will be made blank if a problem occurs such as the file being "in use" or not existing (in which case [ErrorLevel](#) is set to 1). It will also be made blank if *Filename* is an empty file (in which case [ErrorLevel](#) is set to 0).

Filename

The name of the file to read, which is assumed to be in [%A_WorkingDir%](#) if an absolute path isn't specified.

Options: Zero or more of the following strings may be also be present immediately before the name of the file. Separate each option from the next with a single space or tab. For example: `*t *m5000 C:\Log Files\200601.txt`.

***c:** Load a [ClipboardAll](#) file or other binary data. All other options are ignored when ***c** is present.

***m1024:** If this option is omitted, the entire file is loaded unless there is insufficient memory, in which case an error message is shown and the thread exits (but [Try](#) can be used to avoid this). Otherwise, replace 1024 with a decimal or hexadecimal number of bytes. If the file is larger than this, only its leading part is loaded.

Note: This might result in the last line ending in a naked carriage return (`r`) rather than `r`n`.

***t:** Replaces any/all occurrences of carriage return & linefeed (`\r\n`) with linefeed (`\n`). However, this translation reduces performance and is usually not necessary. For example, text containing `\r\n` is already in the right format to be added to a [Gui Edit control](#). Similarly, [FileAppend](#) detects the presence of `\r\n` when it opens a new file; it knows to write each `\r\n` as-is rather than translating it to `\r\r\n`. Finally, the following [parsing loop](#) will work correctly regardless of whether each line ends in `\r\n` or just `\n`: `Loop, parse, MyFileContents, \n, \r`.

***Pnnn:** [AHK_L 42+]: Overrides the default encoding set by [FileEncoding](#), where *nnn* must be a numeric [code page identifier](#).

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[ErrorLevel](#) is set to 0 if the load was successful. It is set to 1 if a problem occurred such as: 1) file does not exist; 2) file is locked or inaccessible; 3) the system lacks sufficient memory to load the file.

[A_LastError](#) is set to the result of the operating system's `GetLastError()` function.

Reading Binary Data

Depending on the file, parameters and default settings, `FileRead` may interpret the file data as text and convert it to the [native encoding](#) used by the script. This is likely to cause problems if the file contains binary data, except in the following cases:

- If the `*C` option is present, all code page and end-of-line translations are unconditionally bypassed.
- If the `*Pnnn` option is present and *nnn* corresponds to the native string encoding, no code page translation occurs.
- If the current [file encoding](#) setting corresponds to the native string encoding, no code page translation occurs.

Note that once the data has been read into `OutputVar`, only the text before the first binary zero (if any are present) will be "seen" by most AutoHotkey

commands and functions. However, the entire contents are still present and can be accessed by advanced methods such as [NumGet\(\)](#).

Finally, [FileOpen\(\)](#) and [File.RawRead\(\)](#) or [File.ReadNum\(\)](#) may be used to read binary data without first reading the entire file into memory.

Remarks

When the goal is to load all or a large part of a file into memory, [FileRead](#) performs much better than using a [file-reading loop](#).

A file greater than 1 GB in size will cause [ErrorLevel](#) to be set to 1 and [OutputVar](#) to be made blank unless the ***m** option is present, in which case the leading part of the file is loaded.

[FileRead](#) does not obey [#MaxMem](#). If there is concern about using too much memory, check the file size beforehand with [FileGetSize](#).

[FileOpen\(\)](#) provides more advanced functionality than [FileRead](#), such as reading or writing data at a specific location in the file without reading the entire file into memory. See [File Object](#) for a list of functions.

Related

[FileEncoding](#), [FileOpen/File Object](#), [file-reading loop](#), [FileReadLine](#), [FileGetSize](#), [FileAppend](#), [IniRead](#), [Sort](#), [UrlDownloadToFile](#)

Examples

```
; Example #1: Read text file into OutputVar.  
FileRead, OutputVar, C:\My Documents\My File.txt
```

```
; Example #2: Quickly sort the contents of a file.  
FileRead, Contents, C:\Address List.txt  
if not ErrorLevel ; Successfully loaded.  
{  
    Sort, Contents
```

```
FileDelete, C:\Address List (alphabetical).txt  
FileAppend, %Contents%, C:\Address List  
(alphabetical).txt  
Contents = ; Free the memory.  
}
```

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 C:\Temp*.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

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

Remarks

[SHFileOperation](#) is used to do the actual work. This function may permanently delete the file if it is too large to be recycled; as of [v1.0.96], a warning should be shown before this occurs.

Related

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

Example

FileRecycle, C:\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

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

Remarks

This commands requires that MS Internet Explorer 4 or later be installed.

Related

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

Example

```
FileRecycleEmpty, C:\
```

FileRemoveDir

Deletes a 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 *DirName*. In this case, if *DirName* is not empty, no action will be taken and `ErrorLevel` will be set to 1.

1: Remove all files and subdirectories (like the Windows command "rmdir /S").

This parameter can be an [expression](#), even one that evaluates to true or false (since true and false are stored internally as 1 and 0).

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

`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 open or save file(s).

```
FileSelectFile, OutputVar [, Options,  
RootDir\Filename, Prompt, Filter]
```

Parameters

OutputVar

The name of the variable in which to store the filename(s) 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 options below.

M: Multi-select. Specify the letter M to allow the user to select more than one file via shift-click, control-click, or other means. **M** may optionally be followed by a number as described below (for example, both M and M1 are valid). To extract the individual files, see the example at the bottom of this page.

S: Save button. Specify the letter S to cause the dialog to always contain a Save button instead of an Open button. **S** may optionally be followed by a number (or sum of numbers) as described below (for example, both S and S24 are valid).

Even if **M** and **S** are absent, the following numbers can be used. To put more than one of them into effect, add them up. For example, to use 8 and 16, specify the number 24.

- 1: File Must Exist
- 2: Path Must Exist

8: Prompt to Create New File

16: Prompt to Overwrite File

32 [v1.0.43.09+]: Shortcuts (.lnk files) are selected as-is rather than being resolved to their targets. This option also prevents navigation into a folder via a folder shortcut.

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.

RootDir\Filename

If present, this parameter contains one or both of the following:

RootDir: The root (starting) directory, which is assumed to be a subfolder in `%A_WorkingDir%` if an absolute path is not specified. If omitted or blank, the starting directory will be a default that might depend on the OS version (it will likely be the directory most recently selected by the user during a prior use of `FileSelectFile`). [v1.0.43.10+]: On Windows XP/2003 and earlier, a **CLSID** such as `::{20d04fe0-3aea-1069-a2d8-08002b30309d}` (i.e. My Computer) may also be specified, in which case any subdirectory present after the CLSID should end in a backslash (otherwise, the string after the last backslash will be interpreted as the default filename, below).

Filename: The default filename to initially show in the dialog's edit field. Only the naked filename (with no path) will be shown. To ensure that the dialog is properly shown, ensure that no illegal characters are present (such as `/<|:`).

Examples:

```
C:\My Pictures\Default Image Name.gif ;  
Both RootDir and Filename are present.  
C:\My Pictures ; Only RootDir is present.  
My Pictures ; Only RootDir is present, and  
it's relative to the current working  
directory.  
My File ; Only Filename is present (but if
```

"My File" exists as a folder, it is assumed to be *RootDir*).

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" drop-down list. To include more than one file extension in the filter, separate them with semicolons as illustrated in the example above.

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[ErrorLevel](#) is set to 1 if the user dismissed the dialog without selecting a file (such as by pressing the Cancel button). It is also set to 1 if the system refused to show the dialog (rare). Otherwise, it is set to 0.

Remarks

If the user didn't select anything (e.g. pressed CANCEL), *OutputVar* is made blank.

If multi-select is not in effect, *OutputVar* is set to the full path and name of the

single file chosen by the user.

If the `M` option (multi-select) is in effect, `OutputVar` is set to a list of items, each of which except the last is followed by a linefeed (`\n`) character. The first item in the list is the path that contains all the selected files (this path will end in a backslash only if it is a root folder such as `C:\`). The other items are the selected filenames (without path). For example:

```
C:\My Documents\New Folder [this is the path in
which all the files below reside]
test1.txt [these are the naked filenames: no
path info]
test2.txt
... etc.
```

(The example at the bottom of this page demonstrates how to extract the files one by one.)

When multi-select is in effect, the sum of the lengths of the selected filenames is limited to 64 KB. Although this is typically enough to hold several thousand files, `OutputVar` will be made blank if the limit is exceeded.

A GUI window may display a modal file-selection dialog by means of [Gui+OwnDialogs](#). A modal dialog prevents the user from interacting with the GUI window until the dialog is dismissed.

Known limitation: A [timer](#) that launches during the display of a `FileSelectFile` dialog will postpone the effect of the user's clicks inside the dialog until after the timer finishes. To work around this, avoid using timers whose subroutines take a long time to finish, or disable all timers during the dialog:

```
Thread, NoTimers
FileSelectFile, OutputVar
Thread, NoTimers, false
```

[v1.0.25.06+]: The multi-select option "4" is obsolete. However, for compatibility with older scripts, it still functions as it did before. Specifically, if

the user selects only one file, *OutputVar* will contain its full path and name followed by a linefeed (`\n`) character. If the user selects more than one file, the format is the same as that of the M option described above, except that the last item also ends in a linefeed (`\n`).

Related

[FileSelectFolder](#), [MsgBox](#), [InputBox](#), [ToolTip](#), [GUI](#), [CLSID List](#), [parsing loop](#), [SplitPath](#)

Also, the operating system offers standard dialog boxes that prompt the user to pick a font, color, or icon. These dialogs can be displayed via [DllCall\(\)](#) as demonstrated at www.autohotkey.com/forum/topic17230.html.

Examples

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

; CLSID Example (requires XP/2003 or earlier):
FileSelectFile, OutputVar,, ::{645ff040-5081-101b-
9f08-00aa002f954e} ; Recycle Bin.

; Multi-Select Example:
FileSelectFile, files, M3 ; M3 = Multiselect
existing files.
if files =
{
    MsgBox, The user pressed cancel.
    return
}
```

```
Loop, parse, files, `n
{
    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 [, StartingFolder,  
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). If the user selects a root directory (such as C:\), *OutputVar* will contain a trailing backslash. If this is undesirable, remove it as follows:

```
FileSelectFolder, Folder  
Folder := RegExReplace(Folder, "\\$") ;  
Removes the trailing backslash, if present.
```

StartingFolder

If blank or omitted, the dialog's initial selection will be the user's My Documents folder (or possibly My Computer). A [CLSID folder](#) such as `::{20d04fe0-3aea-1069-a2d8-08002b30309d}` (i.e. My Computer) may be specified start navigation at a specific special folder.

Otherwise, the most common usage of this parameter is an asterisk followed immediately by the absolute path of the drive or folder to be initially selected. For example, `*C:\` would initially select the C drive. Similarly, `*C:\My Folder` would initially select that particular folder.

The asterisk indicates that the user is permitted to navigate upward (closer to the root) from the starting folder. Without the asterisk, the user would

be forced to select a folder inside *StartingFolder* (or *StartingFolder* itself). One benefit of omitting the asterisk is that *StartingFolder* is initially shown in a tree-expanded state, which may save the user from having to click the first plus sign.

If the asterisk is present, upward navigation may optionally be restricted to a folder other than Desktop. This is done by preceding the asterisk with the absolute path of the uppermost folder followed by exactly one space or tab. In the following example, the user would not be allowed to navigate any higher than C:\My Folder (but the initial selection would be C:\My Folder\Projects):

```
C:\My Folder *C:\My Folder\Projects
```

Options

One of the following numbers:

0: The options below are all disabled (except on Windows 2000, where the "make new folder" button might appear anyway).

1 (default): A button is provided that allows the user to create new folders.

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.

Add 4 to the above number to omit the BIF_NEWDIALOGSTYLE property. Adding 4 ensures that FileSelectFolder will work properly even in a Preinstallation Environment like WinPE or BartPE. However, this prevents the appearance of a "make new folder" button, at least on Windows XP. ["4" requires v1.0.48+]

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.

This parameter can be an [expression](#).

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).

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[ErrorLevel](#) is set to 1 if the user dismissed the dialog without selecting a folder (such as by pressing the Cancel button). It is also set to 1 if the system refused to show the dialog (rare). Otherwise, it is set to 0.

Remarks

A GUI window may display a modal folder-selection dialog by means of [Gui +OwnDialogs](#). A modal dialog prevents the user from interacting with the GUI window until the dialog is dismissed.

Known limitation: A [timer](#) that launches during the display of a FileSelectFolder dialog will postpone the effect of the user's clicks inside the dialog until after the timer finishes. To work around this, avoid using timers whose subroutines take a long time to finish, or disable all timers during the dialog:

```
Thread, NoTimers  
FileSelectFolder, OutputVar,, 3  
Thread, NoTimers, false
```

Related

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

Also, the operating system offers standard dialog boxes that prompt the user to pick a font, color, or icon. These dialogs can be displayed via [DllCall\(\)](#) as demonstrated at www.autohotkey.com/forum/topic17230.html.

Example

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

; CLSID Example:
FileSelectFolder, OutputVar, ::{20d04fe0-3aea-
1069-a2d8-08002b30309d} ; My Computer.
```

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`. *FilePattern* 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.

This parameter can be an [expression](#).

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*. However, files and folders with a complete path name longer than 259 characters are skipped over as though they do not exist. Such files are rare because normally, the operating system does not allow their creation.

This parameter can be an [expression](#).

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[ErrorLevel](#) is set to the number of files that failed to be changed or 0 otherwise.

If files were found, [A_LastError](#) is set to 0 (zero) or the result of the operating system's GetLastError() function immediately after the last failure. Otherwise [A_LastError](#) contains an error code that might indicate why no files were found.

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 valid only when used without any other attributes)

O = OFFLINE

T = TEMPORARY

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

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 blank or 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.

This parameter is an [expression](#). Consequently, if multiple variables need to be concatenated to form a single timestamp, the [dot operator](#) should be used instead of percent signs. For example: `FileSetTime, Year . Month . Day, C:\My File.txt`.

FilePattern

The name of a single file or folder, or a wildcard pattern such as `C:\Temp*.tmp`. *FilePattern* 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 parameter is blank or 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.
- 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.

This parameter can be an [expression](#).

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*. However, files and folders with a complete path name longer than 259 characters are skipped over as though they do not exist. Such files are rare because normally, the operating system does not allow their creation.

This parameter can be an [expression](#).

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[ErrorLevel](#) is set to the number of files that failed to be changed or 0 otherwise. If the specified timestamp is invalid, or *FilePattern* resolves to a blank value, ErrorLevel is set to 1.

If files were found, [A_LastError](#) is set to 0 (zero) or the result of the operating system's GetLastError() function immediately after the last failure. Otherwise [A_LastError](#) contains an error code that might indicate why no files were found.

Remarks

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). For example, 09 is 9am and 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. Similarly, [A_NowUTC](#) contains the current Coordinated Universal Time.

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; for example, 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

[FileGetTime](#), [FileGetAttrib](#), [FileSetAttrib](#), [FileGetSize](#), [FileGetVersion](#), [FormatTime](#), [File-loop](#), [EnvAdd](#), [EnvSub](#)

Example

```
; Set the modification time to the current time
for all matching files:
FileSetTime, , C:\temp\*.txt

; Set the modification date (time will be
midnight):
FileSetTime, 20040122, C:\My Documents\test.doc

; Set the creation date. The time will be set to
4:55pm:
FileSetTime, 200401221655, C:\My
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 parameter:
FileSetTime, 20040122165500, C:\Temp\*.*, M, 1
```

If[Not]Exist

Checks for the existence of a file or folder.

Deprecated: These commands are not recommended for use in new scripts. Use the [FileExist](#) function instead.

```
IfExist, FilePattern  
IfNotExist, FilePattern
```

Parameters

FilePattern

The path, filename, or file pattern to check. *FilePattern* is assumed to be in `%A_WorkingDir%` if an absolute path isn't specified.

Related

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

Examples

```
IfExist, D:\  
    MsgBox, The drive exists.  
IfExist, D:\Docs\*.txt  
    MsgBox, At least one .txt file exists.  
IfNotExist, C:\Temp\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 .ini file. **If omitted, the entire *Section* will be deleted.**

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

`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, C:\Temp\myfile.ini, section2, key
```

IniRead

Reads a value, section or list of section names from a standard format .ini file.

```
IniRead, OutputVar, Filename, Section, Key [,  
Default]  
IniRead, OutputVarSection, Filename, Section  
IniRead, OutputVarSectionNames, Filename
```

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 *Default* parameter (described below).

OutputVarSection

[AHK_L 57+]: Omit the *Key* parameter to read an entire section. Comments and empty lines are omitted. Only the first 65,533 characters of the section are retrieved.

OutputVarSectionNames

[AHK_L 57+]: Omit the *Key* and *Section* parameters to retrieve a linefeed (`\n`) delimited list of section names.

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 .ini file.

Default

The value to store in *OutputVar* if the requested key is not found. If omitted, it defaults to the word ERROR. To store a blank value (empty string), specify `%A_Space%`.

[AHK_L 57+]: This parameter is not used if *Key* is omitted.

ErrorLevel

ErrorLevel is **not** set by this command. If there was a problem, *OutputVar* will be set to the *Default* value as described above.

Remarks

The operating system automatically omits leading and trailing spaces/tabs from the retrieved string. To prevent this, enclose the string in single or double quote marks. The outermost set of single or double quote marks is also omitted, but any spaces inside the quote marks are preserved.

Values longer than 65,535 characters are likely to yield inconsistent results.

A standard ini file looks like:

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

Unicode: IniRead and IniWrite rely on the external functions [GetPrivateProfileString](#) and [WritePrivateProfileString](#) to read and write values. These functions support Unicode only in UTF-16 files; all other files are assumed to use the system's default ANSI code page.

Related

[IniDelete](#), [IniWrite](#), [RegRead](#), [file-reading loop](#), [FileRead](#)

Example

```
IniRead, OutputVar, C:\Temp\myfile.ini, section2,  
key  
MsgBox, The value is %OutputVar%.
```

IniWrite

Writes a value or section to a standard format .ini file.

```
IniWrite, Value, Filename, Section, Key
IniWrite, Pairs, Filename, Section
```

Parameters

Value

The string or number that will be written to the right of *Key*'s equal sign (=).

If the text is long, it can be broken up into several shorter lines by means of a [continuation section](#), which might improve readability and maintainability.

Pairs

[AHK_L 57+]: The complete content of a section to write to the .ini file, excluding the [SectionName] header. *Key* must be omitted. *Pairs* must not contain any blank lines. If the section already exists, everything up to the last key=value pair is overwritten. *Pairs* can contain lines without an equal sign (=), but this may produce inconsistent results. Comments can be written to the file but are stripped out when they are read back by [IniRead](#).

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 .ini file.

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

Remarks

Values longer than 65,535 characters can be written to the file, but may produce inconsistent results as they usually cannot be read correctly by [IniRead](#) or other applications.

A standard ini file looks like:

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

New files are created in either the system's default ANSI code page or UTF-16, depending on [the version of AutoHotkey](#). UTF-16 files may appear to begin with a blank line, as the first line contains the UTF-16 byte order mark. See below for a workaround.

Unicode: IniRead and IniWrite rely on the external functions [GetPrivateProfileString](#) and [WritePrivateProfileString](#) to read and write values. These functions support Unicode only in UTF-16 files; all other files are assumed to use the system's default ANSI code page. In [Unicode scripts](#), IniWrite uses UTF-16 for each new file. If this is undesired, ensure the file exists before calling IniWrite. For example:

```
FileAppend,, NonUnicode.ini, CP0 ; The last  
parameter is optional in most cases.
```

Related

IniDelete, IniRead, RegWrite

Example

```
IniWrite, this is a new value, C:\Temp\myfile.ini,  
section2, key
```

Loop (files & folders)

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

New Syntax [v1.1.21+]

`Loop, Files, FilePattern [, Mode]`

Parameters

Files

The literal word `Files` (case-insensitive). This cannot be a variable or expression.

FilePattern

The name of a single file or folder, or a wildcard pattern such as `C:\Temp*.tmp`. *FilePattern* is assumed to be in [A_WorkingDir](#) if an absolute path isn't specified.

Both asterisks and question marks are supported as wildcards. A match occurs when the pattern appears in either the file's long/normal name or its [8.3 short name](#).

If this parameter is a single file or folder (i.e. no wildcards) and *Mode* includes `R`, more than one match will be found if the specified file name appears in more than one of the folders being searched.

Mode

Zero or more of the following letters:

`D` : Include directories (folders).

`F` : Include files. If both `F` and `D` are omitted, files are included but not folders.

`R` : Recurse into subdirectories (subfolders). If `R` is omitted, files and folders in subfolders are not included.

Old Syntax

Deprecated: This syntax is not recommended for use in new scripts. Use the [new syntax](#) instead.

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

Parameters

FilePattern

The name of a single file or folder, or a wildcard pattern such as `C:\Temp*.tmp`. *FilePattern* is assumed to be in `%A_WorkingDir%` if an absolute path isn't specified.

Both asterisks and question marks are supported as wildcards. A match occurs when the pattern appears in either the file's long/normal name or its [8.3 short name](#).

If this parameter is a single file or folder (i.e. no wildcards) and *Recurse* 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?

One of the following digits, or blank to use the default:

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?

One of the following digits, or blank to use the default:

0 (default) Subfolders are not recursed into.

1 Subfolders are recursed into so that files and folders contained therein are retrieved if they match *FilePattern*. All subfolders will be recursed into, not just those whose names match *FilePattern*.

Special Variables Available Inside a File-Loop

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_LoopFileExt	The file's extension (e.g. TXT, DOC, or EXE). The period (.) is not included.
A_LoopFileFullPath A_LoopFilePath	<p>The path and name of the file/folder currently retrieved. If <i>FilePattern</i> contains a relative path rather than an absolute path, the path here will also be relative. In addition, any short (8.3) folder names in <i>FilePattern</i> will still be short (see next item to get the long version).</p> <p>A_LoopFilePath is available in [v1.1.28+] as an alias of A_LoopFileFullPath, which is a misnomer.</p>
A_LoopFileLongPath	This is different than A_LoopFileFullPath in the following ways: 1) It always contains the absolute/complete path of the file even if <i>FilePattern</i> contains a relative path; 2) Any short (8.3) folder names in <i>FilePattern</i> itself are converted to their long names; 3) Characters in <i>FilePattern</i> are converted to uppercase or lowercase to match the case stored in the file system. This is useful for converting file names -- such as those passed into a script as command line parameters -- to their exact path names as shown by Explorer.

<p>A_LoopFileShortPath</p>	<p>The 8.3 short path and name of the file/folder currently retrieved. For example: C:\MYDOCU~1\ADDRES~1.txt. If <i>FilePattern</i> contains a relative path rather than an absolute path, the path here will also be relative.</p> <p>To retrieve the complete 8.3 path and name for a single file or folder, specify its name for <i>FilePattern</i> as in this example:</p> <pre style="background-color: #f0f0f0; padding: 10px;"> Loop, C:\My Documents\Address List.txt ShortPathName = %A_LoopFileShortPath%</pre> <p>Note: This variable will be blank if the file does not have a short name, which can happen on systems where <code>NtfsDisable8dot3NameCreation</code> has been set in the registry. It will also be blank if <i>FilePattern</i> contains a relative path and the body of the loop uses <code>SetWorkingDir</code> to switch away from the working directory in effect for the loop itself.</p>
<p>A_LoopFileShortName</p>	<p>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 an NTFS file system), <i>A_LoopFileName</i> will be retrieved instead.</p>
<p>A_LoopFileDir</p>	<p>The path of the directory in which <i>A_LoopFileName</i> resides. If <i>FilePattern</i> contains a relative path rather than an absolute path, the path here will also be relative. A root directory will not contain a trailing backslash.</p>

	For example: C:
A_LoopFileTimeModified	The time the file was last modified. Format YYYYMMDDHH24MISS .
A_LoopFileTimeCreated	The time the file was created. Format YYYYMMDDHH24MISS .
A_LoopFileTimeAccessed	The time the file was last accessed. Format YYYYMMDDHH24MISS .
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 also supported.
A_LoopFileSizeKB	The size in Kbytes of the file currently retrieved, rounded down to the nearest integer.
A_LoopFileSizeMB	The size in Mbytes of the file currently retrieved, rounded down to the nearest integer.

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. By contrast, OS features such as the DIR command omit hidden files by default. To avoid processing hidden, system, and/or read-only files, use something like the following inside the loop:

```
if A_LoopFileAttrib contains H,R,S ; Skip any
file that is either H (Hidden), R (Read-only),
or S (System). Note: No spaces in "H,R,S".
    continue ; Skip this file and move on to
the next one.
```

To retrieve files' relative paths instead of absolute paths during a recursive search, use [SetWorkingDir](#) to change to the base folder prior to the loop, and then omit the path from the Loop (e.g. `Loop, *.* , 0, 1`). That will cause [A_LoopFileFullPath](#) to contain the file's path relative to the base folder.

A file-loop can disrupt itself if it creates or renames files or folders within its own purview. For example, if it renames files via [FileMove](#) or other means, each such file might be found twice: once as its old name and again as its new name. To work around this, rename the files only after creating a list of them. For example:

```
FileList =
Loop, Files, *.jpg
    FileList = %FileList%%A_LoopFileName%`n
Loop, Parse, FileList, `n
    FileMove, %A_LoopField%,
renamed_%A_LoopField%
```

Files in an NTFS file system are probably always retrieved in alphabetical order.

Files in other file systems are retrieved in no particular order. To ensure a particular ordering, use the [Sort](#) command as shown in the Examples section below.

Files and folders with a complete path name longer than 259 characters are skipped over as though they do not exist. Such files are rare because normally, the operating system does not allow their creation.

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](#), [SplitPath](#), [FileSetAttrib](#), [FileSetTime](#)

Examples

```
; Example #1:
Loop Files, %A_ProgramFiles%\*.txt, R ; Recurse
into subfolders.
{
    MsgBox, 4, , Filename =
%A_LoopFileFullPath%\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 ; Ask the user to
pick a folder.
Loop, Files, %WhichFolder%\*.*, R
    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, Files, %A_MyDocuments%\Photos\*.*, FD ;
Include Files and Directories
    FileList =
%FileList%A_LoopFileTimeModified%\t%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, Files, %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_LoopFileFullPath%,
%CopyDest%\%A_LoopFileName%, 1 ; Copy with
overwrite=yes
        if ErrorLevel
            MsgBox, Could not copy
%A_LoopFileFullPath% to
"%CopyDest%\%A_LoopFileName%".
    }
}
Return

```

```

; Example #6: Convert filenames passed in via
command-line parameters to long names,
; complete path, and correct uppercase/lowercase
characters as stored in the file system.
Loop %0% ; For each file dropped onto the script
(or passed as a parameter).
{

```

```
GivenPath := %A_Index% ; Retrieve the next
command line parameter.
Loop %GivenPath%, 1
    LongPath = %A_LoopFileLongPath%
    MsgBox The case-corrected long path name of
file`n%GivenPath%`nis:`n%LongPath%
}
```

Loop (read file contents)

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

```
Loop, Read, InputFile [, OutputFile]
```

Parameters

Read

This parameter must be the word READ.

InputFile

The name of the text file whose contents will be read by the loop, which is assumed to be in [%A_WorkingDir%](#) if an absolute path isn't specified. Windows and Unix formats are supported; that is, the file's lines may end in either carriage return and linefeed (`\r\n`) or just linefeed (`\n`).

OutputFile

(Optional) The name of the file to be kept open for the duration of the loop, which is assumed to be in [%A_WorkingDir%](#) if an absolute path isn't specified.

Within the loop's body, use the [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 [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 linefeed (`\n`) after the text, if desired.

The file is not opened if nothing is ever written to it. This happens if the Loop performs zero iterations or if it never uses the [FileAppend](#) command.

End of line (EOL) translation: To disable EOL translation, prepend an asterisk to the filename. This causes each linefeed character (`\n`) to be

written as a single linefeed (LF) rather than the Windows standard of CR+LF. For example: `*C:\My Unix File.txt`. Even without the asterisk, EOL translation is disabled automatically if the Loop's first use of [FileAppend](#) writes any carriage return and linefeed pairs (`\r\n`).

Standard Output (stdout): Specifying an asterisk (*) for *OutputFile* sends any text written by [FileAppend](#) to standard output (stdout). Although such output can be redirected to a file, piped to another EXE, or captured by [fancy text editors](#), it will not appear at the command prompt it was launched from. See [FileAppend](#) for more details.

Escaped Commas: Unlike the last parameter of most other commands, commas in *OutputFile* must be escaped (`\,`).

Remarks

A file-reading loop is useful when you want to operate on 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-reading loop. It contains the contents of the current line excluding the carriage return and linefeed (`\r\n`) that marks the end of the line. If an inner file-reading loop is enclosed by an outer file-reading 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-reading 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 be retrieved as in this example:

```
Loop, read, C:\Database Export.txt
{
    Loop, parse, A_LoopReadLine, %A_Tab%
```

```
    {
        MsgBox, Field number %A_Index% is
%A_LoopField%.
    }
}
```

To load an entire file into a variable, use [FileRead](#) because it performs much better than a loop (especially for large files).

To have multiple files open simultaneously, use [FileOpen](#).

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

To control how the file is decoded when no byte order mark is present, use [FileEncoding](#).

Related

[FileEncoding](#), [FileOpen/File Object](#), [FileRead](#), [FileReadLine](#), [FileAppend](#), [Sort](#), [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 the first line to overwrite rather
than append to any existing file.
```

```
;FileDelete, C:\Docs\Family Addresses.txt
```

```
Loop, read, C:\Docs\Address List.txt,
```

```
C:\Docs\Family Addresses.txt
```

```
{
```

```
    IfInString, A_LoopReadLine, family,
```

```
FileAppend, %A_LoopReadLine%\n
```

```
}
```

```
; Example #2: Retrieve the last line from a text file.
```

```
Loop, read, C:\Log File.txt  
    last_line := A_LoopReadLine ; When loop finishes, this will hold the last line.
```

```
; Example #3: 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:
    ArrayElement := URLStart%A_Index%
    if ArrayElement = ; End of the pseudo-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, URLLcleansed, 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

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

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

`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.

To make a script unconditionally use its own folder as its working directory, make its first line the following:

```
SetWorkingDir %A_ScriptDir%
```

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

Example

```
SetWorkingDir %A_ScriptDir%  
SetWorkingDir, D:\My Folder\Temp
```

SplitPath

Separates a file name or URL 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.

[v1.1.21+]: This parameter can be an `% expression`, but the percent-space prefix must be used.

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 (no backslash). 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 that lacks a drive letter (that is, it has no path or merely a relative path), *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 ?) and other characters illegal in filenames are treated the same as legal characters, with the exception of colon, backslash, and period (dot), which are processed according to their nature in delimiting the drive letter, directory, and extension of the file.

Support for URLs: If *InputVar* contains a colon-double-slash, such as `http://domain.com` or `ftp://domain.com`, *OutDir* is set to the protocol prefix + domain name + directory (e.g. `http://domain.com/images`) and *OutDrive* is set to the protocol prefix + domain name (e.g. `http://domain.com`). All other variables are set according to their definitions above.

Related

[A_LoopFileExt](#), [StringSplit](#), [StringGetPos](#), [StringMid](#), [StringTrimLeft](#), [StringLeft](#), [FileSelectFile](#), [FileSelectFolder](#)

Example

```
FullFileName = C:\My Documents\Address List.txt
```

```
; To fetch only the bare filename from the above:  
SplitPath, FullFileName, name
```

```
; To fetch only its directory:  
SplitPath, FullFileName,, dir
```

```
; To fetch all info:  
SplitPath, FullFileName, name, dir, ext,  
name_no_ext, drive
```

```
; The above will set the variables as follows:  
; name = Address List.txt  
; dir = C:\My Documents  
; ext = txt  
; name_no_ext = Address List  
; drive = C:
```

#Include[Again]

Causes the script to behave as though the specified file's contents are present at this exact position.

```
#Include FileOrDirName
#Include <LibName>
#IncludeAgain FileOrDirName
```

Parameters

FileOrDirName

The path of a file or directory as explained below. This **must not** contain double quotes, wildcards, or references to non-built-in variables. [Escape sequences](#) other than semicolon (;) must not be used, nor are they needed because characters such as percent signs are treated literally.

Percent signs which are not part of a valid variable reference are interpreted literally. All built-in variables are valid, except for [ErrorLevel](#), [A_Args](#) and the [numbered variables](#). Prior to [v1.1.28], only [%A_ScriptDir%](#), [%A_AppData%](#), [%A_AppDataCommon%](#) and [in v1.1.11+] [%A_LineFile%](#) were supported.

File: The name of the file to be included, which is assumed to be in the startup/working directory if an absolute path is not specified (except for [ahk2exe](#), which assumes the file is in the script's own directory). Note: [SetWorkingDir](#) has no effect on [#Include](#) because [#Include](#) is processed before the script begins executing.

Directory: Specify a directory instead of a file to change the working directory used by all subsequent occurrences of [#Include](#) and [FileInstall](#). Note: Changing the working directory in this way does not affect the script's initial working directory when it starts running ([A_WorkingDir](#)). To change that, use [SetWorkingDir](#) at the top of the script.

LibName

[AHK_L 57+]: A library file or function name. For example, `#include <lib>` and `#include <lib_func>` would both include lib.ahk from one of the [function library folders](#). *LibName* cannot contain variable references.

Remarks

A script behaves as though the included file's contents are physically present at the exact position of the #Include directive (as though a copy-and-paste were done from the included file). Consequently, it generally cannot merge two isolated scripts together into one functioning script (to achieve that, see www.autohotkey.com/forum/topic18545.html).

#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.

The *FileName* parameter may optionally be preceded by `*i` and a single space, which causes the program to ignore any failure to read the included file. For example: `#Include *i SpecialOptions.ahk`. This option should be used only when the included file's contents are not essential to the main script's operation.

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 change the line numbering of the main script file by only one line, namely that of the #Include line itself (except for [compiled scripts](#), which merge their included files into one big script at the time of compilation).

#Include is often used to load [functions](#) defined in an external file. Unlike subroutine labels, [functions](#) can be included at the very top of the script without affecting the [auto-execute section](#).

Like other # directives, #Include cannot be executed conditionally. In other words, this example would not work:

```
if x = 1
    #Include SomeFile.ahk ; This line takes
effect regardless of the value of x.
    y = 2 ; And this line would belong to the
IF above, since # directives cannot belong to
IFs.
```

Files can be automatically included (i.e. without having to use #Include) by calling a [library function](#) by name.

[v1.1.11+]: Use `%A_LineFile%\..` to refer to the directory which contains the current file, even if it is not the main script file. For example, `#Include %A_LineFile%\..\other.ahk`. [v1.1.28+]: `%A_AhkPath%\..` can be used to refer to the directory containing AutoHotkey.exe.

Related

[Libraries of Functions](#), [Functions](#), [FileInstall](#)

Example

```
#Include C:\My Documents\Scripts\Utility
Subroutines.ahk
#Include %A_ScriptDir% ; Changes the working
directory for subsequent #Includes and
FileInstalls.
#Include C:\My Scripts ; Same as above but for an
explicitly named directory.
```

{...} (block)

A pair of braces denotes a block. Blocks are typically used with [functions](#), [Else](#), [Loop](#), [While-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 [functions](#), [Else](#), [Loop](#), or an IF-type command such as [IfEqual](#) or [IfWinExist](#).

If an [IF](#), [ELSE](#), [Loop](#), [While-loop](#), or [For-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.

A block may be empty (contain zero commands), which may be useful in cases where you want to comment out the contents of the block without removing the block itself.

One True Brace (OTB, K&R style): The OTB style may optionally be used in the following places: [expression if-statements](#), the [else](#) keyword, [while-loops](#), [For-loops](#), [normal loops](#), [function definitions](#), [Try](#), [Catch](#), and [Finally](#). This style puts the block's opening brace on the same line as the block's controlling statement rather than underneath on a line by itself. For example:

```
if (x < y) {
    ...
} else {
    ...
}
While x < y {
    ...
}
For k,v in obj {
    ...
}
Loop %RepeatCount% {
    ...
}
MyFunction(x, y) {
    ...
}
Try {
    ...
} Catch e {
    ...
} Finally {
    ....
}
```

Similarly, a command or other action may exist to the right of a brace (except the open-brace of the One True Brace style). For example:

```
if x = 1
{ MsgBox This line appears to the right of an
```

opening brace. It executes whenever the IF-statement is true.

```
    MsgBox This is the next line.  
} MsgBox This line appears to the right of a  
closing brace. It executes unconditionally.
```

Related

[Functions](#), [While-loop](#), [Loop](#), [Else](#), [If](#), [If\(Expression\)](#)

Example

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

Break

Exits (terminates) a [loop](#). Valid inside any kind of [loop](#).

```
Break [, LoopLabel]
```

[AHK_L 59+]: If specified, *LoopLabel* identifies which loop this statement should apply to; either by [label name](#) or numeric nesting level. If omitted or 1, this statement applies to the innermost loop in which it is enclosed. *LoopLabel* must be a constant value - variables and expressions are not supported. If a [label](#) is specified, it must point directly at a loop command.

The use of [Break](#) and [Continue](#) are encouraged over [goto](#) since they usually make scripts more readable and maintainable.

Related

[Continue](#), [Loop](#), [While-loop](#), [For-loop](#), [Blocks](#), [Labels](#)

Example

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

```
; Break the outer loop from within a nested loop.
outer:
Loop 3
```

```
{
  x := A_Index
  Loop 3
  {
    if (x*A_Index = 6)
      break outer ; Equivalent to break 2
    or goto break_outer.
    MsgBox %x%,%A_Index%
  }
}
break_outer: ; For goto.
```

Catch [v1.1.04+]

Specifies the code to execute if an exception is raised during execution of a [try](#) statement.

```
Catch [, OutputVar]  
  Statement
```

```
Catch [, OutputVar]  
{  
  Statements  
}
```

Parameters

OutputVar

(Optional) The name of the variable in which to store the value of the exception.

Statement(s)

The commands or expressions to execute if an exception is raised.

Remarks

Every use of *catch* must belong to (be associated with) a [try](#) statement above it. A *catch* always belongs to the nearest unclaimed *try* statement above it unless a [block](#) is used to change that behavior.

The [One True Brace \(OTB\)](#) style may optionally be used. For example:

```
try {  
  ...  
} catch e {
```

```
}    ...
```

Runtime Errors

A *try-catch* statement can also be used to handle runtime errors. There are two kinds of runtime errors: those which normally set [ErrorLevel](#), and those which normally cause the current thread to exit after displaying an error message. Loadtime errors cannot be handled, since they occur before the *try* statement is executed. For backward-compatibility (and in some cases convenience), runtime errors only throw exceptions while a *try* block is executing. Most commands support the use of *try-catch*; however, [ErrorLevel](#) is still set to 0 if the command succeeds.

The value that is stored in *OutputVar* (if present) is an [exception object](#).

Related

[Try](#), [Throw](#), [Finally](#), [Blocks](#)

Examples

See [Try](#).

Continue

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

```
Continue [, LoopLabel]
```

[AHK_L 59+]: If specified, *LoopLabel* identifies which loop this statement should apply to; either by [label name](#) or numeric nesting level. If omitted or 1, this statement applies to the innermost loop in which it is enclosed. *LoopLabel* must be a constant value - variables and expressions are not supported. If a [label](#) is specified, it must point directly at a loop command.

Continue behaves the same as reaching the loop's closing brace:

1. It increases [A_Index](#) by 1.
2. It skips the rest of the loop's body.
3. The loop's condition (if it has one) is checked to see if it is satisfied. If so, a new iteration begins; otherwise the loop ends.

The use of [Break](#) and Continue are encouraged over [goto](#) since they usually make scripts more readable and maintainable.

Related

[Break](#), [Loop](#), [Until](#), [While-loop](#), [For-loop](#), [Blocks](#), [Labels](#)

Example

```
; This example displays 5 MsgBoxes, one for each  
number between 6 and 10.  
; Note that in the first 5 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%
}
```

```
; Continue the outer loop from within a nested
loop.
outer:
Loop 3
{
    x := A_Index
    Loop 3
    {
        if (x*A_Index = 4)
            continue outer ; Equivalent to
continue 2 or goto continue_outer.
        MsgBox %x%,%A_Index%
    }
    continue_outer: ; For goto.
    ErrorLevel:=ErrorLevel ; Prior to revision 57,
labels could not point to the end of a block.
}
```

Critical

Prevents the [current thread](#) from being interrupted by other threads, or enables it to be interrupted.

```
Critical [, Off]
```

```
Critical 50 ; See bottom of remarks.
```

If the first parameter is omitted (or the word On), the [current thread](#) is made critical, meaning that it cannot be interrupted by another thread. If the first parameter is the word Off or [in v1.0.48+] 0, the current thread immediately becomes interruptible, regardless of the settings of [Thread Interrupt](#).

Behavior of Critical Threads

Unlike [high-priority](#) threads, events that occur during a critical thread are not discarded. For example, if the user presses a [hotkey](#) while the current thread is critical, the hotkey is buffered indefinitely until the current thread finishes or becomes noncritical, at which time the hotkey is launched as a new thread.

A critical thread will be interrupted in emergencies. Emergencies consist of: 1) the [OnExit](#) subroutine; 2) any [OnMessage\(\)](#) function that monitors a message number less than 0x312 (or a [callback](#) triggered by such a message); and 3) any [callback](#) indirectly triggered by the critical thread itself (e.g. via [SendMessage](#) or [DllCall](#)). To avoid these interruptions, temporarily disable such functions.

A critical thread becomes interruptible when a [MsgBox](#) or other dialog is displayed. However, unlike [Thread Interrupt](#), the thread becomes critical again after the user dismisses the dialog.

Critical Off

When buffered events are waiting to start new threads, using `Critical Off` will not result in an immediate interruption of the current thread. Instead, an average of 5 milliseconds will pass before an interruption occurs. This makes it

more than 99.999% likely that at least one line after `Critical Off` will execute before an interruption. You can force interruptions to occur immediately by means of a delay such as a `Sleep -1` or a `WinWait` for a window that does not yet exist.

`Critical Off` cancels the current thread's period of uninterruptibility even if the thread was not Critical, thereby letting events such as `GuiSize` be processed sooner or more predictably.

Thread Settings

See `A_IsCritical` for how to save and restore the current setting of Critical. However, since Critical is a thread-specific setting, when a critical thread ends, the underlying/resumed thread (if any) will be automatically noncritical. Consequently there is no need to do `Critical Off` right before ending a thread.

If Critical is not used in the auto-execute section (top part of the script), all threads start off as noncritical (though the settings of `Thread Interrupt` will still apply). By contrast, if the auto-execute section turns on Critical but never turns it off, every newly launched thread (such as a `hotkey`, `custom menu item`, or `timed subroutine`) starts off critical.

The command `Thread NoTimers` is similar to Critical except that it only prevents interruptions from `timers`.

[v1.0.47+]: Turning on Critical also puts `SetBatchLines -1` into effect for the `current thread`.

Message Check Interval

[v1.0.47+]: Specifying a positive number as the first parameter (e.g. `Critical 30`) turns on Critical but also changes the number of milliseconds between checks of the internal message queue. If unspecified, messages are checked every 16 milliseconds while Critical is On, and every 5 ms while Critical is Off. Increasing the interval postpones the arrival of messages/events, which gives the `current thread` more time to finish. This reduces the possibility that certain `OnMessage()` and `GUI events` will be lost due to "thread already running". However, commands that wait such as `Sleep` and `WinWait` will check messages

regardless of this setting (a workaround is `DllCall("Sleep", UInt, 500)`).

Note: Increasing the message-check interval too much may reduce the responsiveness of various events such as [GUI](#) window repainting.

Related

[Thread \(command\)](#), [Threads](#), [#MaxThreadsPerHotkey](#), [#MaxThreadsBuffer](#), [OnMessage\(\)](#), [RegisterCallback\(\)](#), [Hotkey](#), [Menu](#), [SetTimer](#)

Example

```
#space:: ; Win+Space hotkey.  
Critical  
ToolTip No new threads will launch until after  
this ToolTip disappears.  
Sleep 3000  
ToolTip ; Turn off the tip.  
return ; Returning from a hotkey subroutine ends  
the thread. Any underlying thread to be resumed is  
noncritical by definition.
```

Else

Specifies the command(s) to perform if an IF-statement evaluates to FALSE. When more than one command is present, enclose them in a [block](#) (braces).

Else

Remarks

Every use of ELSE must belong to (be associated with) an IF-statement above it. An ELSE always belongs to the nearest unclaimed IF-statement 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 at the bottom).

When an IF or an ELSE owns more than one line, those lines must be enclosed in braces. However, if only one line belongs to an IF or ELSE, the braces are optional. For example:

```
if count > 0 ; No braces are required around  
the next line because it's only a single line.  
    MsgBox Press OK to begin the process.  
else ; Braces must be used around the section  
below because it consists of more than one line.  
{  
    WinClose Untitled - Notepad  
    MsgBox There are no items present.  
}
```

The [One True Brace \(OTB\) style](#) may optionally be used around an "else". For example:

```
if IsDone {  
    ...  
} else if (x < y) {  
    ...  
} else {  
    ...  
}
```

Related

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

Examples

```
IfWinExist, Untitled - Notepad  
{  
    WinActivate  
    Send This is a test.{Enter}  
}  
else  
{  
    WinActivate, Some Other Window  
    MouseClick, left, 100, 200  
}  
  
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](#) and contains no hotkeys) the entire script.

```
Exit [, ExitCode]
```

Parameters

ExitCode

An integer between -2147483648 and 2147483647 (can be an [expression](#)) 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.

Remarks

If the script has no hotkeys, isn't [persistent](#), and hasn't requested `NumLock` `ScrollLock` `CapsLock` `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](#), [Functions](#), [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 between -2147483648 and 2147483647 (or [in v1.0.48.01+] an [expression](#)) 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.

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    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.
```

Finally [v1.1.14+]

Ensures that one or more statements (commands or expressions) are always executed after the execution of a [try](#) statement.

Finally Statement

```
Finally
{
    Statements
}
```

Remarks

Every use of *finally* must belong to (be associated with) a [try](#) (or [catch](#)) statement above it. A *finally* always belongs to the nearest unclaimed *try* statement above it unless a [block](#) is used to change that behavior.

Try statements behave differently depending on whether *catch* or *finally* is present. For more information, see [Try](#).

Goto, *break*, *continue* and *return* cannot be used to exit a *finally* block, as that would require suppressing any control flow instructions within the *try* block. For example, if *try* uses `return 42`, the value 42 is returned after the *finally* block executes. Attempts to jump out of a *finally* block using one of these commands are detected as errors at load time where possible, or at run time otherwise.

Prior to [v1.1.19.02], a bug existed which prevented control flow statements within *try* from working when *finally* was present. *Return* was erroneously permitted within *finally*, but was ignored if an exception had been thrown.

The [One True Brace \(OTB\) style](#) may optionally be used with the *finally* command. For example:

```
try {
    ...
} finally {
    ...
}

try {
    ...
} catch e {
    ...
} finally {
    ...
}
```

Related

[Try, Catch, Throw, Blocks](#)

Examples

```
try
{
    Tooltip, Working...
    Example1()
}
catch e
{
    ; For more detail about the object that e
contains, see Catch.
    MsgBox, 16,, % "Exception thrown!`n`nwhat: "
e.what "`nfile: " e.file
        . "`nline: " e.line "`nmessage: "
e.message "`nextra: " e.extra
}
finally
{
```

```
        ToolTip ; hide the ToolTip
    }

    MsgBox, Done!

; This function has a Finally block that acts as
cleanup code
Example1()
{
    try
        Example2()
    finally
        MsgBox, This is always executed regardless
of exceptions
}

; This function fails when the minutes are odd
Example2()
{
    if Mod(A_Min, 2)
        throw Exception("Test exception")
    MsgBox, Example2 did not fail
}
```

For-loop [AHK_L 59+]

Repeats a series of commands once for each key-value pair in an object.

```
For Key [, Value] in Expression
```

Parameters

Key

Name of the variable in which to store the key at the beginning of each iteration.

Value

Name of the variable in which to store the value associated with the current key.

Expression

An [expression](#) which results in an object, or a variable which contains an object.

Remarks

Expression is evaluated only once, before the loop begins. If its result is not an object, execution jumps immediately to the line following the loop's body; otherwise, the object's `_NewEnum()` method is called to retrieve an *enumerator* object. At the beginning of each iteration, the enumerator's `Next()` method is used to retrieve the next key-value pair. If `Next()` returns zero or an empty string, the loop terminates.

Although not exactly equivalent to a for-loop, the following demonstrates this process:

```
_enum := (Expression)._NewEnum()  
if IsObject(_enum)
```

```
while _enum.Next(Key, Value)
{
    ...
}
```

Existing key-value pairs may be modified during the loop, but inserting or removing keys may cause some items to be skipped or enumerated multiple times. One workaround is to build a list of keys to remove, then use a second loop to remove the keys after the first loop completes. Note that `Object.Remove(first, last)` can be used to remove a range of keys without looping.

A for-loop 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 statement does not require a block (an "if" and its "else" count as a single statement for this purpose). The One True Brace (OTB) style may optionally be used, which allows the open-brace to appear on the same line rather than underneath. For example: `for x, y in z {`.

As with all loops, [Break](#), [Continue](#) and [A_Index](#) may be used.

COM Objects

Since *Key* and *Value* are passed directly to the enumerator's `Next()` method, the values they are assigned depends on what type of object is being enumerated. For COM objects, *Key* contains the value returned by `IEnumVARIANT::Next()` and *Value* contains a number which represents its [variant type](#). For example, when used with a [Scripting.Dictionary](#) object, each *Key* contains a key from the dictionary and *Value* is typically 8 for strings and 3 for integers. See [ComObjType](#) for a list of type codes.

[v1.0.96.00+]: When enumerating a [SafeArray](#), *Key* contains the current element and *Value* contains its variant type.

Related

[Enumerator object](#), [Object.NewEnum\(\)](#), [While-loop](#), [Loop](#), [Until](#), [Break](#), [Continue](#), [Blocks](#)

Examples

```
; List the key-value pairs of an object:
colours := Object("red", 0xFF0000, "blue",
0x0000FF, "green", 0x00FF00)
; The above expression could be used directly in
place of "colours" below:
for k, v in colours
    s .= k "=" v "`n"
MsgBox % s
```

```
; List all open Explorer and Internet Explorer
windows:
for window in
ComObjCreate("Shell.Application").Windows
    windows .= window.LocationName " :: "
window.LocationURL "`n"
MsgBox % windows
```

```
/*
Class: CEnumerator

Generic enumerator object that can be used for
iterating over numeric keys.
The array must not be modified during iteration,
otherwise the iterated range will be invalid.
It's possible to define a custom MaxIndex()
functions for array boundaries.
If there are missing array members between 1 and
max index, they will be iterated but will have a
value of "".
This means that real sparse arrays are not
supported by this enumerator by design.
To make an object use this iterator, insert this
function in the class definition:
```

```
_NewEnum()  
{  
    return new CEnumerator(this)  
}
```

Source:

<http://www.autohotkey.com/board/topic/2667-suggestions-on-documentation-improvements/?p=531509>
*/

```
; Iterate over the enumerator
```

```
For k, v in Test  
    MsgBox %k%=%v%
```

```
; Test class for demonstrating usage
```

```
class Test  
{  
    static Data := ["abc", "def", "ghi"]  
  
    _NewEnum()  
    {  
        return new CEnumerator(this.Data)  
    }  
}
```

```
class CEnumerator
```

```
{  
    __New(Object)  
    {  
        this.Object := Object  
        this.first := true  
        ; Cache for speed. Useful if custom  
MaxIndex() functions have poor performance.  
        ; In return, that means that no key-value  
pairs may be inserted during iteration or the
```

range will become invalid.

```
    this.ObjMaxIndex := Object.MaxIndex()
}

Next(ByRef key, ByRef value)
{
    if (this.first)
    {
        this.Remove("first")
        key := 1
    }
    else
        key ++
    if (key <= this.ObjMaxIndex)
        value := this.Object[key]
    else
        key := ""
    return key != ""
}
}
```

Gosub

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

```
Gosub, Label
```

Parameters

Label

The name of the [label](#), [hotkey label](#), or [hotstring label](#) to which to jump, which causes the commands beneath *Label* to be executed until a [Return](#) or [Exit](#) is encountered. ["Return"](#) causes the script to jump back to the first command beneath the `Gosub` and resume execution there. ["Exit"](#) terminates the [current thread](#).

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 is slightly reduced because the target label must be "looked up" each time rather than only once when the script is first loaded.

When using a dynamic label such as `%MyLabel%`, an error dialog will be displayed if the label does not exist. To avoid this, call [IsLabel\(\)](#) beforehand. For example:

```
if IsLabel(VarContainingLabelName)
    Gosub %VarContainingLabelName%
```

Although `Gosub` is useful for simple, general purpose subroutines, consider using [functions](#) for more complex purposes.

Related

Return, Functions, IsLabel(), 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](#) to which to jump.

Remarks

When using a dynamic label such as %MyLabel%, an error dialog will be displayed if the label does not exist. To avoid this, call [IsLabel\(\)](#) beforehand. For example:

```
if IsLabel(VarContainingLabelName)
    Goto %VarContainingLabelName%
```

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

[Gosub](#), [Return](#), [IsLabel\(\)](#), [Else](#), [Blocks](#), [Break](#), [Continue](#)

Example

```
Goto, MyLabel
...
MyLabel:
```

Sleep, 100

...

if (expression)

Specifies the command(s) to perform if an [expression](#) evaluates to TRUE.

```
if (expression)
```

Remarks

An if-statement that contains an expression is differentiated from a [traditional-if](#) such as `If FoundColor <> Blue` by making the character after the word "if" an open-parenthesis. Although this is usually accomplished by enclosing the entire expression in parentheses, it can also be done with something like `if (x > 0)` and `(y > 0)`. In addition, the open-parenthesis may be omitted entirely if the first item after the word "if" is a [function call](#) or an operator such as "not" or "!".

If the if-statement's expression evaluates to true (which is any result other than an empty string or the number 0), the line or [block](#) underneath it is executed. Otherwise, if there is a corresponding ELSE, execution jumps to the line or block underneath it.

When an IF or an [ELSE](#) owns more than one line, those lines must be enclosed in [braces](#). However, if only one line belongs to an IF or ELSE, the braces are optional. See the examples at the bottom of this page.

The space after `if` is optional if the expression starts with an open-parenthesis, as in `if(expression)`.

The One True Brace (OTB) style may optionally be used with if-statements that are expressions (but not [traditional if-statements](#)). For example:

```
if (x < y) {  
    ...  
}  
if WinExist("Untitled - Notepad") {
```

```
    WinActivate
}
if IsDone {
    ...
} else {
    ...
}
```

Unlike an "else" statement -- which supports any type of statement immediately to its right -- an if-statement supports only a "{" to its right.

On a related note, the command `if var [not] between LowerBound and UpperBound` checks whether a variable is between two values, and `if var [not] in value1,value2` can be used to check whether a variable's contents exist within a list of values.

Related

[Expressions](#), [Assign expression \(:=\)](#), [if var in/contains MatchList](#), [if var between](#), [IfInString](#), [Blocks](#), [Else](#), [While-loop](#)

Example

```
if (A_Index > 100)
    return

if (A_TickCount - StartTime > 2*MaxTime + 100)
{
    MsgBox Too much time has passed.
    ExitApp
}

if (Color = "Blue" or Color = "White")
{
    MsgBox The color is one of the allowed values.
    ExitApp
}
```

```
else if (Color = "Silver")
{
    MsgBox Silver is not an allowed color.
    return
}
else
{
    MsgBox This color is not recognized.
    ExitApp
}
```

If / IfEqual / IfNotEqual / IfLess / IfLessOrEqual / IfGreater / IfGreaterOrEqual

Specifies the command(s) to perform if the comparison of a [variable](#) to a value evaluates to TRUE. When more than one command is present, enclose them in a [block](#) (braces).

Deprecated: IfEqual, IfNotEqual, IfLess, IfLessOrEqual, IfGreater and IfGreaterOrEqual are not recommended for use in new scripts. See [Scripting Language: If Statement](#) for details.

```
IfEqual, Var, Value (same: if Var = Value)
IfNotEqual, Var, Value (same: if Var <> Value) (!=
can be used in place of <>)
IfLess, Var, Value (same: if Var < Value)
IfLessOrEqual, Var, Value (same: if Var <= Value)
IfGreater, Var, Value (same: if Var > Value)
IfGreaterOrEqual, Var, Value (same: if Var >= Value)
If Var ; If Var's contents are blank or 0, it is
considered false. Otherwise, it is true.
```

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 *Var* 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*).

When an IF or an **ELSE** owns more than one line, those lines must be enclosed in braces. For example:

```
if count <= 0
{
    WinClose Untitled - Notepad
    MsgBox There are no items present.
}
```

However, if only one line belongs to the IF or ELSE, the braces are optional.

Another command can only appear on the same line as the IF statement if you use the command-name style. In other words, these are valid:

```
IfEqual, x, 1, Sleep, 1
IfGreater, x, 1, EnvAdd, x, 2
```

But these are not valid:

```
if x = 1 Sleep 1
IfGreater, x, 1, x += 2
```

The **One True Brace (OTB) style** may **not** be used with these types of if-statements. It can only be used with expression if-statements.

On a related note, the command **if var [not] between LowerBound and UpperBound** checks whether a variable is between two values, and **if var [not] in value1,value2** can be used to check whether a variable's contents exist within a list of values.

Related

[IF \(expression\)](#), [StringCaseSense](#), [Assign expression \(:=\)](#), [if var in/contains](#), [MatchList](#), [if var between](#), [IfInString](#), [Blocks](#), [Else](#)

Example

```
if counter >= 1
    Sleep, 10

if counter >=1    ; If an IF has more than one
line, enclose those lines in braces:
{
    WinClose, Untitled - Notepad
    Sleep 10
}

if MyVar = %MyVar2%
    MsgBox The contents of MyVar and MyVar2 are
identical.
else if MyVar =
{
    MsgBox, 4,, MyVar is empty/blank. Continue?
    IfMsgBox, No
        Return
}
else if MyVar <> ,
    MsgBox The value in MyVar is not a comma.
else
    MsgBox The value in MyVar is a comma.

if Done
    MsgBox The variable Done is neither empty nor
zero.
```

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, *Var* must be greater than or equal to this string, number, or variable reference.

UpperBound

To be within the specified range, *Var* 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.

Caution: The operators "between", "is", "in", and "contains" are not supported in [expressions](#).

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 %VarLow% and %VarHigh%
    MsgBox %var% is between %VarLow% and
    %VarHigh%.

if var between blue and red
    MsgBox %var% is alphabetically between the
    words 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 <i>Var</i> is non-empty and contains a purely numeric string (decimal or hexadecimal) without a decimal point. Leading and trailing spaces and tabs are allowed. The string may start with a plus or minus sign.
float	True if <i>Var</i> is non-empty and contains a floating point number; that is, a purely numeric string containing a decimal point. Leading and trailing spaces and tabs are allowed. The string may start with a plus sign, minus sign, or decimal point.
number	True if <i>Var</i> contains an integer or floating point number (each of which is described above).
digit	True if <i>Var</i> is empty or contains only digits, which consist of the characters 0 through 9. Other characters such as the following are not allowed: spaces, tabs, plus signs, minus signs, decimal points, hexadecimal digits, and the 0x prefix.

xdigit	Hexadecimal digit: Same as <i>digit</i> except the characters A through F (uppercase or lowercase) are also allowed. [v1.0.44.09+]: A prefix of 0x is tolerated if present.
alpha	True if <i>Var</i> is empty or contains only alphabetic characters. False if there are any digits, spaces, tabs, punctuation, or other non-alphabetic characters anywhere in the string. For example, if <i>Var</i> contains a space followed by a letter, it is <i>not</i> considered to be <i>alpha</i> .
upper	True if <i>Var</i> is empty or contains only uppercase characters. False if there are any digits, spaces, tabs, punctuation, or other non-uppercase characters anywhere in the string.
lower	True if <i>Var</i> is empty or contains only lowercase characters. False if there are any digits, spaces, tabs, punctuation, or other non-lowercase characters anywhere in the string.
alnum	Same as <i>alpha</i> except that characters 0 through 9 are also allowed.
space	True if <i>Var</i> 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	<p>True if <i>Var</i> 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.</p> <p>Years less than 1601 are not considered valid because the operating system generally does not support them. The maximum year considered valid is 9999.</p> <p>The word DATE may be used as a substitute for the word TIME, with the same result.</p>

Note: The operators "between", "is", "in", and "contains" are **not**

supported in expressions.

[AHK_L 42+]: The system locale is ignored unless [StringCaseSense Locale](#) has been used.

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.
```

If var [not] in/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 *Var* contains one of the list items as a substring.

MatchList

A comma-separated list of strings, each of which will be compared to the contents of *Var* 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 *MatchList* is set to `ABC , XYZ` then *Var* must contain either ABC with a trailing space or XYZ with a leading space 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 string1:

```
If Var In string1,,string2 . Similarly, the following list
contains only a single item with a literal comma inside it: If Var In
single,,item . To include a blank item in the list, make the first
character a comma as in this example: If Var In
,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).

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.

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.

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.

If *MatchList* is long, it can be broken up into several shorter lines by means of a [continuation section](#), which might improve readability and maintainability.

Caution: The operators "between", "is", "in", and "contains" are not supported in [expressions](#).

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 contains 1,3 ; Note that it compares the  
values as strings, not numbers.
```

```
    MsgBox Var contains the digit 1 or 3 (Var  
could be 1, 3, 10, 21, 23, etc.)
```

```
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.
```

If[Not]InString

Checks if a [variable](#) contains the specified string.

Deprecated: These commands are not recommended for use in new scripts. Use the [InStr](#) function instead.

```
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. Matching is not case sensitive unless [StringCaseSense](#) has been turned on.

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
```

However, items other than named commands are not supported on the same line.

For example:

```
IfInString, MyVar, abc, found := true ;  
Invalid.
```

Related

[InStr\(\)](#), [RegExMatch\(\)](#), [StringGetPos](#), [StringCaseSense](#), [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
```

IfMsgBox

Checks which button was pushed by the user during the most recent [MsgBox](#) command.

IfMsgBox, `ButtonName`

Parameters

ButtonName

One of the following strings to represent which button the user pressed in the most recent [MsgBox](#) command:

Yes

No

OK

Cancel

Abort

Ignore

Retry

Continue [v1.0.44.08+]

TryAgain [v1.0.44.08+]

Timeout (that is, the word "timeout" is present if the [MsgBox](#) [timed out](#))

Related

[MsgBox](#)

Example

```
MsgBox, 4, , Would you like to continue?, 5 ; 5-  
second timeout.  
IfMsgBox, No  
    Return ; User pressed the "No" button.
```

IfMsgBox, Timeout

```
Return ; i.e. Assume "No" if it timed out.  
; Otherwise, continue:
```

```
...
```

WinActive() / IfWin[Not]Active

Checks if the specified window exists and is currently active (foremost). If it is, *WinActive()* returns its Unique ID (HWND).

WinActive()

```
UniqueID := WinActive(WinTitle [, WinText,  
ExcludeTitle, ExcludeText])
```

Parameters

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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.

Return Value

This function returns the [Unique ID \(HWND\)](#) (as hexadecimal integer) of the active window if it matches the specified criteria. If it does not, the function returns 0. Since all non-zero numbers are seen as "true", the statement `if WinActive(WinTitle)` is true whenever *WinTitle* is active.

Examples

```
if WinActive("ahk_class Notepad") or  
WinActive("ahk_class" . ClassName) ; "ahk_class"
```

need not have a space after it.

WinClose ; Uses the last found window.

IfWin[Not]Active

Deprecated: This command is not recommended for use in new scripts. Use the [WinActive](#) function instead.

```
IfWinActive [, WinTitle, WinText, ExcludeTitle, ExcludeText]
IfWinNotActive [, WinTitle, WinText, ExcludeTitle, ExcludeText]
```

Parameters

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). Hidden text elements are detected if [DetectHiddenText](#) is ON.

ExcludeTitle

Windows whose titles include this value will not be considered.

Note: Due to backward compatibility, *IfWin[Not]Active* interpret this parameter as a command if it exactly matches the name of a command. To work around this, use the [WinActive](#) function instead.

ExcludeText

Windows whose text include this value will not be considered.

Remarks

[SetWinDelay](#) does not apply to this command.

Examples

```
IfWinActive, Untitled - Notepad
{
    WinMaximize ; Maximizes the Notepad window
    found by IfWinActive above.
    Send, Some text.{Enter}
    return
}
```

Remarks

If all parameters are omitted, the [Last Found Window](#) will be used.

If the function or command 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 *WinActive* or *IfWinActive* evaluates to true or *IfWinNotActive* evaluates to false, the [Last Found Window](#) will be updated.

An easy way to retrieve the unique ID of the active window is with `ActiveHwnd := WinExist("A")`.

Window titles and text are case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[IfWinExist](#), [SetTitleMatchMode](#), [DetectHiddenWindows](#), [Last Found Window](#), [WinActivate](#), [WinWaitActive](#), [WinWait](#), [WinWaitClose](#), [#IfWinActive/Exist](#)

WinExist() / IfWin[Not]Exist

Checks if a matching window exists. If it is, *WinExist()* returns the Unique ID (HWND) of the first matching window.

WinExist()

```
UniqueID := WinExist(WinTitle [, WinText,  
ExcludeTitle, ExcludeText])
```

Parameters

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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.

Return Value

This function returns the [Unique ID \(HWND\)](#) (as hexadecimal integer) of the first matching window (0 if none). Since all non-zero numbers are seen as "true", the statement `if WinExist(WinTitle)` is true whenever *WinTitle* exists.

Examples

```
if WinExist("ahk_class Notepad") or  
WinExist("ahk_class" . ClassName)  
    WinActivate ; Uses the last found window.
```

```
MsgBox % "The active window's ID is " .  
WinExist("A")
```

```
If !WinExist("Calculator") ; Equivalent to  
IfWinNotExist, Calculator  
return
```

IfWin[Not]Exist

Deprecated: This command is not recommended for use in new scripts. Use the [WinExist](#) function instead.

```
IfWinExist [, WinTitle, WinText, ExcludeTitle, ExcludeText]
IfWinNotExist [, WinTitle, WinText, ExcludeTitle, ExcludeText]
```

Parameters

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). Hidden text elements are detected if [DetectHiddenText](#) is ON.

ExcludeTitle

Windows whose titles include this value will not be considered.

Note: Due to backward compatibility, *IfWin[Not]Exist* interpret this parameter as a command if it exactly matches the name of a command. To work around this, use the [WinExist](#) function instead.

ExcludeText

Windows whose text include this value will not be considered.

Remarks

[SetWinDelay](#) does not apply to this command.

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
}
```

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 the function or command determines that a qualified window exists, the [Last Found Window](#) will be updated to be that window. In other words, if *WinExist* or *IfWinExist* evaluates to true or *IfWinNotExist* evaluates to false, the [Last Found Window](#) will be updated.

To discover the HWND of a control (for use with [Post/SendMessage](#) or [DllCall](#)), use [ControlGet Hwnd](#) or [MouseGetPos](#).

Window titles and text are case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[IfWinActive](#), [SetTitleMatchMode](#), [DetectHiddenWindows](#), [Last Found Window](#), [Process](#), [WinActivate](#), [WinWaitActive](#), [WinWait](#), [WinWaitClose](#), [#IfWinActive/Exist](#)

Loop (normal)

Perform a series of commands 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, the Loop continues indefinitely until a `break` or `return` is encountered.

If *Count* is a variable reference such as `%ItemCount%`, the loop is skipped entirely whenever the variable is blank or contains a number less than 1.

Due to the need to support [file-pattern loops](#), *Count* cannot be an expression. However, as with all non-expression parameters, an expression can be forcibly used by preceding it with a `%` and a space. For example: `Loop % Count + 1`. In such cases, the expression is evaluated only once, right before the loop begins.

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 statement does not require a block (an "if" and its "else" count as a single statement for this purpose).

A common 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. One can also create a "While" or "Do...While/Until" loop by making the first or

last statement of the loop's *body* an IF statement that conditionally issues the [break](#) command, but the use of [While](#) or [Loop...Until](#) is usually preferred.

The built-in variable **A_Index** contains the number of the current loop iteration. It contains 1 the first time the loop's *body* is executed. For the second time, it contains 2; and so on. If an inner loop is enclosed by an outer loop, the inner loop takes precedence. A_Index works inside all types of loops, including [file-loops](#) and [registry-loops](#); but A_Index contains 0 outside of a loop.

The [One True Brace \(OTB\) style](#) may optionally be used with normal loops (but not specialized loops such as [file-pattern](#) and [parsing](#)). For example:

```
Loop {
    ...
}
Loop %RepeatCount% {
    ...
}
```

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-reading loops](#) can operate on the entire contents of a file, one line at a time. Finally, [parsing loops](#) can operate on the individual fields contained inside a delimited string.

Related

[Until](#), [While-loop](#), [For-loop](#), [Files-and-folders loop](#), [Registry loop](#), [File-reading 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 at a time.

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

Parameters

Parse

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`.

InputVar

The name of the variable whose contents will be analyzed. Do not enclose the name in percent signs unless you want the *contents* of the variable to be used as the name.

[v1.1.21+]: This parameter can be an `% expression`, but the percent-space prefix must be used.

Delimiters

If this parameter is blank or omitted, each character of *InputVar* will be treated as a separate substring.

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"
```

Otherwise, *Delimiters* contains one or more characters (case sensitive), each of which is used to determine where the boundaries between

substrings occur in *InputVar*.

Delimiter characters are not considered to be part of the substrings themselves. In addition, if there is nothing between a pair of delimiters within *InputVar*, 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 *InputVar*.

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:

```
StringReplace, NewHTML, HTMLString, <br>,
¢, All
Loop, parse, NewHTML, ¢ ; Parse the string
based on the cent symbol.
{
...
}
```

OmitChars

An optional list of characters (case sensitive) to exclude from the beginning and end of each substring. For example, if *OmitChars* is `%A_Space%%A_Tab%`, spaces and tabs will be removed from the beginning and end (but not the middle) of every retrieved substring.

If *Delimiters* is blank, *OmitChars* indicates which characters should be excluded from consideration (the loop will not see them).

Unlike the last parameter of most other commands, commas in *OmitChars* must be escaped (`\,`).

Remarks

A string parsing loop is 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 [pseudo-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.

Although there is no built-in variable "A_LoopDelimiter", the example at the very bottom of this page demonstrates how to detect which delimiter was encountered for each field.

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 contents.

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

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

Related

[StringSplit](#), [file-reading loop](#), [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: Read the lines inside a variable,
one by one (similar to a file-reading loop).
; A file can be loaded into a variable via
FileRead:
Loop, parse, FileContents, `n, `r ; Specifying `n
prior to `r allows both Windows and Unix files to
be parsed.
{
    MsgBox, 4, , Line number %A_Index% is
%A_LoopField%.`n`nContinue?
    IfMsgBox, No, break
}
```

```
; Example #3: This is the same as the example
above except that it's for the clipboard.
; It's 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 #4: 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
    }
}

```

; Example #5: Determining which delimiter was encountered.

; Initialize string to search.

Colors = red,green|blue;yellow|cyan,magenta

; Initialize counter to keep track of our position in the string.

Position := 0

```

Loop, Parse, Colors, `,|;

```

```
{
```

; Calculate the position of the delimiter at the end of this field.

```
    Position += StrLen(A_LoopField) + 1
```

; Retrieve the delimiter found by the parsing loop.

```
    Delimiter := SubStr(Colors, Position, 1)
```

```

    MsgBox Field: %A_LoopField%`nDelimiter:
%Delimiter%

```

```
}
```

Loop (registry)

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

New Syntax [v1.1.21+]

```
Loop, Reg, KeyName [, Mode]
```

Parameters

Reg

The literal word `Reg` (case-insensitive). This cannot be a variable or expression.

KeyName

The full name of the registry key such as `HKLM\Software` or `%FullPathOfKey%`.

This must start with `HKEY_LOCAL_MACHINE`, `HKEY_USERS`, `HKEY_CURRENT_USER`, `HKEY_CLASSES_ROOT`, or `HKEY_CURRENT_CONFIG` (or the abbreviations for each of these, such as `HKLM`). To access a [remote registry](#), prepend the computer name and a colon, as in this example:

```
\\workstation01:HKEY_LOCAL_MACHINE
```

Mode

Zero or more of the following letters:

`K`: Include keys.

`V`: Include values. Values are also included if both `K` and `V` are omitted.

`R`: Recurse into subkeys. If `R` is omitted, keys and values within subkeys of `KeyName` are not included.

Old Syntax

Deprecated: This syntax is not recommended for use in new scripts. Use the [new syntax](#) instead.

`Loop, RootKey [, Key, IncludeSubkeys?, Recurse?]`

Parameters

RootKey

Must be either HKEY_LOCAL_MACHINE (or HKLM), HKEY_USERS (or HKU), HKEY_CURRENT_USER (or HKCU), HKEY_CLASSES_ROOT (or HKCR), or HKEY_CURRENT_CONFIG (or HKCC).

To access a [remote registry](#), prepend the computer name and a colon, as in this example: \\workstation01:HKEY_LOCAL_MACHINE

Key

The name of the key (e.g. Software\SomeApplication). If blank or omitted, the contents of *RootKey* will be retrieved.

IncludeSubkeys?

0 (default) Subkeys contained within *Key* are not retrieved (only the values).

1 All values and subkeys are retrieved.

2 Only the subkeys are retrieved (not the values).

Recurse?

0 (default) Subkeys are not recursed into.

1 Subkeys are recursed into, so that all values and subkeys contained therein are retrieved.

Remarks

A registry-loop is useful when you want to operate on a collection registry values or subkeys, one at a time. The values and subkeys are retrieved in reverse order (bottom to top) so that [RegDelete](#) can be used inside the loop without disrupting the loop.

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

A_LoopRegName	Name of the currently retrieved item, which can be either a value name or the name of a subkey. Value names displayed by Windows RegEdit as "(Default)" will be retrieved if a value has been assigned to them, but A_LoopRegName will be blank for them.
A_LoopRegType	The type of the currently retrieved item, which is one of the following words: KEY (i.e. the currently retrieved item is a subkey not a value), REG_SZ, REG_EXPAND_SZ, REG_MULTI_SZ, REG_DWORD, REG_QWORD, REG_BINARY, REG_LINK, REG_RESOURCE_LIST, REG_FULL_RESOURCE_DESCRIPTOR, REG_RESOURCE_REQUIREMENTS_LIST, REG_DWORD_BIG_ENDIAN (probably rare on most Windows hardware). It will be empty if the currently retrieved item is of an unknown type.
A_LoopRegKey	The name of the root key being accessed (HKEY_LOCAL_MACHINE, HKEY_USERS, HKEY_CURRENT_USER, HKEY_CLASSES_ROOT, or HKEY_CURRENT_CONFIG). For remote registry access, this value will not include the

	computer name.
A_LoopRegSubKey	Name of the current subkey. This will be the same as the <i>Key</i> parameter unless the <i>Recurse</i> parameter is being used to recursively explore other subkeys. In that case, it will be the full path of the currently retrieved item, not including the root key. For example: Software\SomeApplication\My SubKey
A_LoopRegTimeModified	The time the current subkey or any of its values was last modified. Format YYYYMMDDHH24MISS . This variable will be empty if the currently retrieved item is not a subkey (i.e. <i>A_LoopRegType</i> is not the word KEY).

When used inside a registry-loop, the following commands can be used in a simplified way to indicate that the currently retrieved item should be operated upon:

RegRead , OutputVar	Reads the current item. If the current item is a key, ErrorLevel will be set to 1 and <i>OutputVar</i> will be made empty.
RegWrite [, Value]	Writes to the current item. If <i>Value</i> is omitted, the item will be made 0 or blank depending on its type. If the current item is a key, ErrorLevel will be set to 1 and there will be no other effect.
RegDelete	Deletes the current item. If the current item is a key, it will be deleted along with any subkeys and values it contains.

When accessing a remote registry (via the *RootKey* or *KeyName* parameter described above), the following notes apply:

- The target machine must be running the Remote Registry service.
- Access to a remote registry may fail if the target computer is not in the same domain as yours or the local or remote username lacks sufficient permissions (however, see below for possible workarounds).

- Depending on your username's domain, workgroup, and/or permissions, you may have to connect to a shared device, such as by mapping a drive, prior to attempting remote registry access. Making such a connection -- using a remote username and password that has permission to access or edit the registry -- may as a side-effect enable remote registry access.
- If you're already connected to the target computer as a different user (for example, a mapped drive via user Guest), you may have to terminate that connection to allow the remote registry feature to reconnect and re-authenticate you as your own currently logged-on username.
- For Windows Server 2003 and Windows XP Professional, MSDN states: "If the [registry server] computer is joined to a workgroup and the *Force network logons using local accounts to authenticate as Guest* policy is enabled, the function fails. Note that this policy is enabled by default if the computer is joined to a workgroup."
- For Windows XP Home Edition, MSDN states that "this function always fails". Home Edition lacks both the registry server and client, though the client can be extracted from one of the OS cab files.

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](#), [RegRead](#), [RegWrite](#), [RegDelete](#), [SetRegView](#)

Examples

```
; Example: Delete Internet Explorer's history of
URLs typed by the user:
Loop, HKEY_CURRENT_USER,
Software\Microsoft\Internet Explorer\TypedURLs
    RegDelete
```

```
; Example: A working test script:
Loop, Reg,
HKEY_CURRENT_USER\Software\Microsoft\Windows, KVR
{
    if A_LoopRegType = key
        value =
    else
    {
        RegRead, value
        if ErrorLevel
            value = *error*
    }
    MsgBox, 4, , %A_LoopRegName% = %value%
    (%A_LoopRegType)`n`nContinue?
    IfMsgBox, NO, break
}
```

```
; Example: A working example to recursively search
the entire
; registry for particular value(s).
SetBatchLines -1 ; Makes searching occur at
maximum speed.
RegSearchTarget = Notepad ; Tell the subroutine
what to search for.
Gosub, RegSearch
return
```

```

RegSearch:
ContinueRegSearch = y
Loop, Reg, HKEY_LOCAL_MACHINE, KVR
{
    Gosub, CheckThisRegItem
    if ContinueRegSearch = n ; It told us to stop.
        return
}
Loop, Reg, HKEY_USERS, KVR
{
    Gosub, CheckThisRegItem
    if ContinueRegSearch = n ; It told us to stop.
        return
}
Loop, Reg, HKEY_CURRENT_CONFIG, KVR
{
    Gosub, CheckThisRegItem
    if ContinueRegSearch = n ; It told us to stop.
        return
}
; Note: I believe HKEY_CURRENT_USER does not need
to be searched if HKEY_USERS
; is being searched. The same might also be true
for HKEY_CLASSES_ROOT if
; HKEY_LOCAL_MACHINE is being searched.
return

```

```

CheckThisRegItem:
if A_LoopRegType = KEY ; Remove these two lines
if you want to check key names too.
    return
RegRead, RegValue
if ErrorLevel
    return
IfInString, RegValue, %RegSearchTarget%
{

```

```
    MsgBox, 4, , The following match was  
found: `n%A_LoopRegKey%\%A_LoopRegSubKey%\%A_LoopRe  
gName%`nValue = %RegValue%`n`nContinue?  
    IfMsgBox, No  
        ContinueRegSearch = n ; Tell our caller  
to stop searching.  
    }  
return
```

OnExit

Specifies a [callback function](#) or [subroutine](#) to run automatically when the script exits.

OnExit() [v1.1.20+]

```
OnExit(Func [, AddRemove])
```

Parameters

Func

A function name or [function object](#) to call when the script is exiting. The function can optionally define parameters as shown below. If an OnExit function returns a non-zero integer, the script does not exit. Otherwise, the script exits after all registered functions are called.

```
ExitFunc(ExitReason, ExitCode)
```

AddRemove

One of the following values:

- 1** (the default): Call the function after any previously registered functions.
- 1**: Call the function before any previously registered functions.
- 0**: Do not call the function.

If a label (subroutine) has been registered, it is always called first.

Remarks

New scripts should use a function instead of a subroutine -- this reduces the risk of accidentally creating a script which can't be exited, and ensures that the exit code passed to Exit or ExitApp is preserved.

Any number of OnExit functions can be registered. If a label (subroutine) is also registered, the functions are called after the subroutine calls [ExitApp](#). An OnExit function usually should not call ExitApp; if it does, the script terminates immediately.

Examples

; After running this script, right click the tray icon and click Exit to test the OnExit callback function.

`#Persistent` ; Prevent the script from exiting automatically.

; Register a function to be called on exit:
`OnExit("ExitFunc")`

; Register an object to be called on exit:
`OnExit(ObjBindMethod(MyObject, "Exiting"))`

`ExitFunc(ExitReason, ExitCode)`

```
{
    if ExitReason not in Logoff,Shutdown
    {
        MsgBox, 4, , Are you sure you want to
exit?
        IfMsgBox, No
            return 1 ; OnExit functions must
return non-zero to prevent exit.
    }
    ; Do not call ExitApp -- that would prevent
other OnExit functions from being called.
}
```

`class MyObject`

```
{
    Exiting()
    {
        MsgBox, MyObject is cleaning up prior to
exiting...
        /*
        this.SayGoodbye()
        this.CloseNetworkConnections()
    }
}
```

```
}  
  }  
    */
```

OnExit

Deprecated: This command is not recommended for use in new scripts. Use the [OnExit](#) function instead.

```
OnExit [, Label]
```

Parameters

Label

If omitted, any previously registered label is unregistered. Otherwise, specify the name of the [label](#) whose contents will be executed (as a new [thread](#)) when the script exits by any means.

Remarks

IMPORTANT: Since the specified subroutine is called instead of terminating the script, that subroutine must use the [ExitApp](#) command if termination is desired.

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 words from the [table below](#).

Examples

```
; After running this script, right click the tray  
icon and click Exit to test the OnExit subroutine.  
#Persistent ; Prevent the script from exiting  
automatically.  
OnExit, ExitSub  
return
```

ExitSub:

```
if A_ExitReason not in Logoff,Shutdown ; Avoid  
spaces around the comma in this line.
```

```
{
```

```
    MsgBox, 4, , Are you sure you want to exit?
```

```
    IfMsgBox, No
```

```
        return
```

```
}
```

```
ExitApp ; A script with an OnExit subroutine will  
not terminate unless the subroutine uses ExitApp.
```

Remarks

The OnExit callback function or 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.

A script can detect and optionally abort a system shutdown or logoff via [OnMessage\(0x11, "WM_QUERYENDSESSION"\)](#) .

The OnExit [thread](#) does not obey [#MaxThreads](#) (it will always launch when needed). In addition, while it is running, it cannot be interrupted by any [thread](#), including [hotkeys](#), [custom menu items](#), and [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 callback function or 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 callback function or subroutine from making a script impossible to terminate.

If the OnExit [thread](#) was launched due to an [Exit](#) or [ExitApp](#) command that specified an exit code, in [v1.1.19] and earlier that code is ignored and no longer available. In [v1.1.20+] the initial exit code is used unless overridden by calling [ExitApp](#) with a new exit code.

Whenever an exit attempt is made, each OnExit callback function or subroutine starts off fresh with the default values for settings such as [SendMode](#). These defaults can be changed in the [auto-execute section](#).

Exit Reasons

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 close (hide) the window without terminating the script, use WinHide .
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 program or document.
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

[OnMessage\(\)](#), [RegisterCallback\(\)](#), [OnClipboardChange](#), [ExitApp](#), [Shutdown](#), [#Persistent](#), [Threads](#), [Gosub](#), [Return](#), [Menu](#)

Pause

Pauses the script's [current thread](#).

```
#p::Pause ; Pressing Win+P once will pause the
script. Pressing it again will unpause.
Pause [, On|Off|Toggle, OperateOnUnderlyingThread?]
```

Parameters

On|Off|Toggle

If blank or omitted, it defaults to Toggle. Otherwise, specify one of the following words:

Toggle: Pauses the [current thread](#) unless the thread beneath it is paused, in which case the underlying thread is unpaused.

On: Pauses the current thread.

Off: If the thread beneath the current thread is paused, it will be in an unpaused state when resumed. Otherwise, the command has no effect.

OperateOnUnderlyingThread?

This parameter is ignored for `Pause Off` because that always operates on the underlying thread. For the others, it is ignored unless Pause is being turned on (including via Toggle).

Specify one of the following numbers:

0 (or omitted): The command pauses the current thread; that is, the one now running the Pause command.

1: The command marks the thread beneath the current thread as paused so that when it resumes, it will finish the command it was running (if any) and then enter a paused state. If there is no thread beneath the current

thread, the script itself is paused, which prevents [timers](#) from running (this effect is the same as having used the menu item "Pause Script" while the script has no threads).

Note: [A_IsPaused](#) contains the pause state of the underlying thread.

Remarks

By default, the script can also be paused via its [tray icon](#) or [main window](#).

Unlike [Suspend](#) -- which disables [hotkeys](#) and [hotstrings](#) -- turning on pause will freeze the [current thread](#). As a side-effect, any interrupted threads beneath it will lie dormant.

Whenever any thread is paused, [timers](#) will not run. By contrast, explicitly launched threads such as [hotkeys](#) and [menu items](#) can still be launched; but when their [threads](#) finish, the underlying 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 whenever the script's [current thread](#) is in a paused state. This color change can be avoided by freezing the icon, which is achieved by specifying 1 for the last parameter of the `Menu` command. For example:

```
Menu, Tray, Icon, C:\My Icon.ico, , 1
```

To disable [timers](#) without pausing the script, use [Thread, NoTimers](#).

A script is always halted (though not officially paused) while it is displaying any kind of [menu](#) (tray menu, menu bar, GUI context menu, etc.)

Related

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

Examples

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

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

```
; Send a Pause command to another script.
```

```
DetectHiddenWindows, On
```

```
WM_COMMAND := 0x111
```

```
ID_FILE_PAUSE := 65403
```

```
PostMessage, WM_COMMAND, ID_FILE_PAUSE,,,
```

```
C:\YourScript.ahk ahk_class AutoHotkey
```

Reload

Replaces the currently running instance of the script with a new one.

Reload

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.

By default, the script can also be reloaded via its [tray icon](#) or [main window](#).

Any [command-line parameters](#) passed to the original script are not passed to the new instance. To pass such parameters, do not use Reload. Instead, use [Run](#) in conjunction with [A_AhkPath](#) and [A_ScriptFullPath](#) (and [A_IsCompiled](#) if the script is ever used in compiled form). Also, include the string `/restart` as the first parameter (i.e. after the name of the executable), which tells the program to use the same behavior as Reload. See also: [command line switches and syntax](#).

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.

If the script cannot be reloaded -- perhaps because it has a syntax error -- the original instance of the script will continue running. Therefore, the reload command should be followed by whatever actions you want taken in the event of a failure (such as a [return](#) to exit the current subroutine). To have the original instance detect the failure, follow this example:

```
Reload
Sleep 1000 ; If successful, the reload will
close this instance during the Sleep, so the
line below will never be reached.
MsgBox, 4,, The script could not be reloaded.
Would you like to open it for editing?
```

```
IfMsgBox, Yes, Edit  
return
```

Related

[Edit](#)

Example

```
^!r::Reload ; Assign Ctrl-Alt-R as a hotkey to  
restart the script.
```

Return

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

```
Return [, Expression]
```

Parameters

Expression

This parameter should be omitted except when `return` is used inside a [function](#).

Since this parameter is an [expression](#), all of the following are valid examples:

```
return 3
return "literal string"
return MyVar
return i + 1
return true ; Returns the number 1 to mean
"true".
return ItemCount < MaxItems ; Returns a
true or false value.
return FindColor(TargetColor)
```

Known limitation: For backward compatibility and ease-of-use, the following two examples are functionally identical:

```
return MyVar
return %MyVar%
```

In other words, a single variable enclosed in percent signs is treated as a

non-expression. To work around this, make it unambiguously an expression by enclosing it in parentheses; for example: `return (%MyVar%)`.

Remarks

If there is no caller to which to return, *Return* will do an [Exit](#) instead.

There are various ways to return multiple values from function to caller described within [Returning Values to Caller](#).

Related

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

Example

```
#Z::  
MsgBox The Win-Z hotkey was pressed.  
Gosub MySubroutine  
return  
  
MySubroutine:  
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. Also, this mode is mutually exclusive of the 20ms mode in the previous paragraph; that is, only one of them can be in effect at a time.

Remarks

Use `SetBatchLines -1` to never sleep (i.e. have the script run at maximum speed).

The default setting is 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](#), [SendMode](#), [SetKeyDelay](#), [SetMouseDelay](#), and [SetDefaultMouseSpeed](#).

Every newly launched [thread](#) (such as a [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](#), [SendMode](#), [SetKeyDelay](#), [SetMouseDelay](#), [SetDefaultMouseSpeed](#), [Critical](#)

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|Delete, Priority]
```

Parameters

Label

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. 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.

[v1.1.01+]: If *Label* is omitted, [A_ThisLabel](#) will be used. For example, `SetTimer , , Off` can be used inside a timer subroutine to turn off the timer, while `SetTimer , , 1000` would either update the current timer's *Period* or set a new timer using the label which is currently running. [v1.1.24+]: If [A_ThisLabel](#) is empty but the current thread was launched by a timer, that timer is used. This is useful for timers which launch functions or function objects.

[v1.1.20+]: If not a valid label name, this parameter can be the name of a function, or a single variable reference containing a [function object](#). For example, `SetTimer %FuncObj%, 1000` or `SetTimer %FuncObj, 1000`. Other expressions which return objects are currently unsupported. See [the class example](#) for more details.

Note: [v1.1.24+]: Passing an empty variable or an expression which results in an empty value is considered an error. This parameter must be either given a non-empty value or completely omitted.

Period|On|Off|Delete

On: Re-enables a previously disabled timer at its former *period*. If the timer doesn't exist, it is created (with a default period of 250). The timer is also **reset**. If the timer exists but was previously set to **run-only-once mode**, it will again run only once.

Off: Disables an existing timer.

Delete [v1.1.20+]: Disables and deletes an existing timer. If the timer is associated with a **function object**, the object is released. Turning off a timer does not release the object.

Period: Creates or updates a timer using the absolute value of this parameter as the **approximate** number of milliseconds that must pass before the timer is executed. The timer will be automatically enabled and **reset**. It can be set to repeat automatically or run only once:

- If *Period* is positive, the timer will automatically repeat until it is explicitly disabled by the script.
- [v1.0.46.16+]: If *Period* is negative, the timer will run only once. For example, specifying -100 would run the timer 100 ms from now then disable the timer as though `SetTimer, Label, Off` had been used.

[v1.1.24+]: If *Label* is an object created by the script (not an actual function or label), the timer is automatically deleted after the timer function returns, unless the timer was re-enabled. This allows the object to be freed if the script is no longer referencing it, but it also means the timer's *Period* and *Priority* are not retained.

Period must be an integer, unless a variable or expression is used, in which case any fractional part is ignored. Its absolute value must be no larger than 4294967295 ms (49.7 days).

Default: If this parameter is 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 *period* unless a *Priority* is specified.

Priority

This optional parameter is an integer between -2147483648 and 2147483647 (or an [expression](#)) 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 created, re-enabled, or updated with a new *period*, its subroutine will not run right away; its time *period* must expire first. If you wish the timer's first execution to be immediate, use [Gosub](#) to execute the timer's subroutine (however, this will not start a new thread like the timer itself does; so settings such as [SendMode](#) will not start off at their defaults).

Reset: If SetTimer is used on an existing timer and parameter #2 is a number or the word ON (or it is omitted), the timer is reset; in other words, the entirety of its period must elapse before its subroutine will run again.

Timer precision: Due to the granularity of the OS's time-keeping system, *Period* is typically rounded up to the nearest multiple of 10 or 15.6 milliseconds (depending on the type of hardware and drivers installed). For example, a *Period* between 1 and 10 (inclusive) is usually equivalent to 10 or 15.6 on Windows 2000/XP. A shorter delay may be achieved via Loop+Sleep as demonstrated at [DllCall+timeBeginPeriod+Sleep](#).

Reliability: 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](#) (this can be avoided via [Critical](#)). 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.
4. The script is uninterruptible as a result of [Critical](#) or [Thread Interrupt/Priority](#). During such times, timers will not run. Later, when the script becomes interruptible again, any overdue timer will run once as soon as possible and then resume its normal schedule.

Although timers will operate when the script is [suspended](#), they will not run if the [current thread](#) has "[Thread NoTimers](#)" in effect or whenever any thread is [paused](#). In addition, they do not operate when the user is navigating through one of the script's menus (such as the tray icon menu or a menu bar).

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.

Other remarks: 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 examples at the bottom of this page).

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

If [hotkey](#) response time is crucial (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 script running -- such as one that contains only timers -- use [#Persistent](#).

Related

[Gosub](#), [Return](#), [Threads](#), [Thread \(command\)](#), [Critical](#), [IsLabel\(\)](#), [Menu](#), [#Persistent](#)

Examples

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

```
#Persistent  
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:
```

```
#Persistent
```

```
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:
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

```

```

; Example #4: Using a method as the timer
subroutine.

```

```

counter := new SecondCounter
counter.Start()
Sleep 5000
counter.Stop()
Sleep 2000

```

```

; An example class for counting the seconds...
class SecondCounter {
    __New() {
        this.interval := 1000
    }
}

```

```

        this.count := 0
        ; Tick() has an implicit parameter "this"
which is a reference to
        ; the object, so we need to create a
function which encapsulates
        ; "this" and the method to call:
        this.timer := ObjBindMethod(this, "Tick")
    }
    Start() {
        ; Known limitation: SetTimer requires a
plain variable reference.
        timer := this.timer
        SetTimer % timer, % this.interval
        ToolTip % "Counter started"
    }
    Stop() {
        ; To turn off the timer, we must pass the
same object as before:
        timer := this.timer
        SetTimer % timer, Off
        ToolTip % "Counter stopped at " this.count
    }
    ; In this example, the timer calls this
method:
    Tick() {
        ToolTip % ++this.count
    }
}

```

Tips relating to the above example:

- We can also use `this.timer := this.Tick.Bind(this)`. When `this.timer` is called, it will effectively invoke `this.Tick.Call(this)` (except that `this.Tick` is not re-evaluated). By contrast, `ObjBindMethod` produces an object which invokes `this.Tick()`.
- If we rename `Tick` to `Call`, we can just use `this` directly instead of

`this.timer`. This also removes the need for the temporary variable. However, `ObjBindMethod` is useful when the object has multiple methods which should be called by different event sources, such as hotkeys, menu items, GUI controls, etc.

- If the timer is being modified or deleted from within a function/method called by the timer, it may be easier to [omit the *Label* parameter](#). In some cases this avoids the need to retain the original object which was passed to `SetTimer`, which eliminates one temporary variable (like `timer` in the example above).

Sleep

Waits the specified amount of time before continuing.

Sleep, DelayInMilliseconds

Parameters

Delay

The amount of time to pause (in milliseconds) between 0 and 2147483647 (24 days), which can be an [expression](#).

Remarks

Due to the granularity of the OS's time-keeping system, *Delay* is typically rounded up to the nearest multiple of 10 or 15.6 milliseconds (depending on the type of hardware and drivers installed). For example, a delay between 1 and 10 (inclusive) is equivalent to 10 or 15.6 on most Windows 2000/XP systems. To achieve a shorter delay, see [Examples](#).

The actual delay time might wind up being longer than what was requested if the CPU is under load. This is because the OS gives each needy process a slice of CPU time (typically 20 milliseconds) before giving another timeslice to the script.

A delay of 0 yields the remainder of the script's current timeslice to any other processes that need it (as long as they are not significantly lower in [priority](#) than the script). Thus, a delay of 0 produces an actual delay between 0 and 20ms (or more), depending on the number of needy processes (if there are no needy processes, there will be no delay at all). However, a *Delay* of 0 should always wind up being shorter than any longer *Delay* would have been.

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

Sleep -1: A delay of -1 does not sleep but instead makes the script immediately

check its message queue. This can be used to force any pending [interruptions](#) to occur at a specific place rather than somewhere more random. See [Critical](#) for more details.

Related

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

Examples

```
Sleep, 1000 ; 1 second
```

```
; The following is a working example that  
demonstrates how to sleep for less time than the  
; normal 10 or 15.6 milliseconds.  
; NOTE: While a script like this is running, the  
entire operating system and all applications are  
; affected by timeBeginPeriod below.
```

```
SetBatchLines -1 ; Ensures maximum effectiveness  
of this method.
```

```
SleepDuration = 1 ; This can sometimes be finely  
adjusted (e.g. 2 is different than 3) depending on  
the value below.
```

```
TimePeriod = 3 ; Try 7 or 3. See comment below.  
; On a PC whose sleep duration normally rounds up  
to 15.6 ms, try TimePeriod=7 to allow  
; somewhat shorter sleeps, and try TimePeriod=3 or  
less to allow the shortest possible sleeps.
```

```
DllCall("winmm\timeBeginPeriod", UInt, TimePeriod)  
; Affects all applications, not just this script's  
DllCall("Sleep"...), but does not affect SetTimer.
```

```
Iterations = 50
StartTime := A_TickCount

Loop %Iterations%
    DllCall("Sleep", UInt, SleepDuration) ; Must
use DllCall instead of the Sleep command.

DllCall("Winmm\timeEndPeriod", UInt, TimePeriod)
; Should be called to restore system to normal.
MsgBox % "Sleep duration = " . (A_TickCount -
StartTime) / Iterations
```

Suspend

Disables or enables all or selected [hotkeys](#) and [hotstrings](#).

```
Suspend [ , Mode ]
```

Parameters

Mode

On: Suspends all [hotkeys](#) and [hotstrings](#) except those explained the Remarks section.

Off: Re-enables the hotkeys and hotstrings that were disable above.

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

By default, the script can also be suspended via its [tray icon](#) or [main window](#).

Any hotkey/hotstring subroutine whose very first line is Suspend (except `Suspend On`) 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.

The [keyboard](#) and/or [mouse](#) hooks will be installed or removed if justified by the changes made by this command.

To disable selected hotkeys or hotstrings automatically based on the type of window that is present, use [#IfWinActive/Exist](#).

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. This can be avoided by freezing the icon, which is done by specifying 1 for the last parameter of the `Menu` command. For example:

```
Menu, Tray, Icon, C:\My Icon.ico, , 1
```

The built-in variable `A_IsSuspended` contains 1 if the script is suspended and 0 otherwise.

Related

[#IfWinActive/Exist](#), [Pause](#), [Menu](#), [ExitApp](#)

Example

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

```
; Send a Suspend command to another script.  
DetectHiddenWindows, On  
WM_COMMAND := 0x111  
ID_FILE_SUSPEND := 65404  
PostMessage, WM_COMMAND, ID_FILE_SUSPEND,,,  
C:\YourScript.ahk ahk_class AutoHotkey
```

Thread

Sets the priority or interruptibility of [threads](#). It can also temporarily disable all [timers](#).

```
Thread, NoTimers [, false]  
Thread, Priority, n  
Thread, Interrupt [, Duration, LineCount]
```

Thread, NoTimers [, false]: Prevents interruptions from any [timers](#) until the [current thread](#) either ends, executes `Thread, NoTimers, false`, or is interrupted by another thread that allows timers (in which case timers can interrupt the interrupting thread until it finishes).

If `Thread NoTimers` is not used in the auto-execute section (top part of the script), all threads start off as interruptible by timers (though the settings of `Thread Interrupt` [below] will still apply). By contrast, if the auto-execute section turns on *NoTimers* but never turns it off, every newly launched [thread](#) (such as a [hotkey](#), [custom menu item](#), or [timer](#)) starts off immune to interruptions by timers.

Regardless of the default setting, timers will always operate when the script has no threads (unless [Pause](#) has been turned on).

`Thread, NoTimers` is equivalent to `Thread, NoTimers, true`. In addition, since the true/false parameter is an [expression](#), true resolves to 1, and false to 0.

Thread, Priority, n: Specify for **n** an integer between -2147483648 and 2147483647 (or an [expression](#)) to indicate the current thread's new priority. This has no effect on other threads. See [Threads](#) for details.

Due to its ability to buffer events, the command [Critical](#) is generally superior to `Thread Priority`.

On a related note, the OS's priority level for the entire script can be changed as in this example: `Process, Priority,, High`.

Thread, Interrupt [, Duration, LineCount]: This command should be used sparingly because most scripts perform more consistently with settings close to the defaults.

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).

Note: Any *Duration* less than 17 might result in a shorter actual duration or immediate interruption, since the system tick count has a minimum resolution of 10 to 16 milliseconds.

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 maximum for both components is 2147483647.

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.

If [Critical](#) is specified as the first line of the thread's subroutine or function, the thread starts out uninterruptible and the Interrupt setting has no effect. However, this does not apply to bound functions or user-defined [function objects](#).

Remarks

Due to its greater flexibility and its ability to buffer events, the command [Critical](#) is generally more useful than [Thread Interrupt](#) and [Thread Priority](#).

Related

[Critical](#), [Threads](#), [Hotkey](#), [Menu](#), [SetTimer](#), [Process](#)

Example

```
Thread, priority, 1 ; Make priority of current
thread slightly above average.
Thread, interrupt, 0 ; Make each newly launched
thread immediately interruptible:
Thread, interrupt, 50, 2000 ; Make each thread
interruptible after 50ms or 2000 lines, whichever
comes first.
```

Throw [v1.1.04+]

Signals the occurrence of an error. This signal can be caught by a [try-catch](#) statement.

```
Throw [, Expression]
```

Parameters

Expression

A value to store in [catch](#)'s OutputVar.

Since this parameter is an [expression](#), all of the following are valid examples:

```
throw 3
throw "literal string"
throw MyVar
throw i + 1
throw { what: "Custom error", file:
A_LineFile, line: A_LineNumber } ; Throws
an object
```

This parameter is always an expression, so variable references should not be enclosed in [percent signs](#) except to perform a [double-deref](#).

[v1.1.05+]: If omitted, an [exception object](#) is thrown with a default message.

Exception(Message [, What, Extra])

Creates an object with the following properties, also common to exceptions created by [runtime errors](#):

- **Message:** An error message or [ErrorLevel](#) value.
- **What:** The name of the command, function or label which was executing or about to execute when the error occurred.
- **Extra:** Additional information about the error, if available.
- **File:** Set automatically to the full path of the script file which contains the line at which the error occurred.
- **Line:** Set automatically to the line number at which the error occurred.

If *What* is omitted, it defaults to the name of the current function or subroutine. Otherwise it can be a string or a negative offset from the top of the call stack. For example, a value of -1 sets `Exception.What` to the current function or subroutine and `Exception.Line` to the line which called it. However, if the script is [compiled](#) or the offset is invalid, *What* is simply converted to a string.

Message and *Extra* are converted to strings. These are displayed by an error dialog if the exception is thrown and not caught.

```
try
    SomeFunction()
catch e
    MsgBox % "Error in " e.What ", which was
called at line " e.Line

SomeFunction() {
    throw Exception("Fail", -1)
}
```

Related

[Try](#), [Catch](#), [Finally](#)

Examples

See [Try](#).

Try [v1.1.04+]

Guards one or more statements (commands or expressions) against runtime errors and exceptions thrown by the [throw](#) command.

Try Statement

```
Try
{
    Statements
}
```

Remarks

The *try* command is usually followed by a [block](#) - one or more statements (commands or expressions) enclosed in braces. If only a single statement is to be executed, it can be placed on the same line as *try* or on the next line, and the braces can be omitted. To specify code that executes only when *try* catches an error, use the [catch](#) command.

An exception can be thrown by the [throw](#) command or by the program when a runtime error occurs. When an exception is thrown from within a try block or a function called by one, the following occurs:

- If there is a corresponding [catch](#) statement, execution continues there.
- If there is no catch statement but there is a [finally](#) statement, it is executed, but once it finishes the exception is automatically thrown again.
- If there is neither a catch statement nor a finally statement, execution continues at the next line outside the try block.

If an exception is thrown while no try blocks are executing, an error message is shown and the current thread exits.

The [One True Brace \(OTB\) style](#) may optionally be used with the *try* command. For example:

```
try {  
    ...  
} catch e {  
    ...  
}
```

Related

[Catch](#), [Throw](#), [Finally](#), [Blocks](#)

Examples

; Example #1: The basic concept of try/catch/throw.

try ; Attempts to execute code.

```
{  
    HelloWorld()  
    MakeToast()  
}
```

catch e ; Handles the first error/exception raised by the block above.

```
{  
    MsgBox, An exception was  
thrown!`nSpecifically: %e%  
    Exit  
}
```

HelloWorld() ; Always succeeds.

```
{  
    MsgBox, Hello, world!  
}
```

MakeToast() ; Always fails.

```
{
```

```
    ; Jump immediately to the try block's error
handler:
    throw A_ThisFunc " is not implemented, sorry"
}
```

```
; Example #2: Using try/catch instead of
ErrorLevel.
```

```
try
{
    ; The following tries to back up certain types
of files:
    FileCopy, %A_MyDocuments%\*.txt,
D:\Backup\Text documents
    FileCopy, %A_MyDocuments%\*.doc,
D:\Backup\Text documents
    FileCopy, %A_MyDocuments%\*.jpg,
D:\Backup\Photos
}
catch
{
    MsgBox, 16,, There was a problem while backing
the files up!
    ExitApp
}
```

```
; Example #3: Dealing with COM errors.
```

```
try
{
    obj := ComObjCreate("ScriptControl")
    obj.ExecuteStatement("MsgBox ""This is
embedded VBScript""")
    obj.InvalidMethod() ; This line produces a
runtime error.
}
```

```

catch e
{
    ; For more detail about the object that e
contains, see Exception\(\).
    MsgBox, 16,, % "Exception thrown!\n\nwhat: "
e.what "\nfile: " e.file
        . "\nline: " e.line "\nmessage: "
e.message "\nextra: " e.extra
}

```

; Example #4: Nesting try-catch statements.

```

try Example1() ; Any single statement can be on
the same line with a Try command.
catch e
    MsgBox, Example1() threw %e%.

```

```

Example1()
{
    try Example2()
    catch e
    {
        if e = 1
            throw e ; Rethrow the exception so
that the caller can catch it.
        else
            MsgBox, Example2() threw %e%.
    }
}

```

```

Example2()
{
    Random, o, 1, 2
    throw o
}

```

Until [AHK_L 59+]

Applies a condition to the continuation of a Loop or For-loop.

```
Loop {  
    ...  
} Until Expression
```

Parameters

Expression

- Any valid [expression](#).

Remarks

The expression is evaluated once after each iteration, and is evaluated even if [continue](#) was used. If the expression evaluates to false (which is an empty string or the number 0), the loop continues; otherwise, the loop is broken and execution continues at the line following *Until*.

Loop Until is shorthand for the following:

```
Loop {  
    ...  
    if (Expression)  
        break  
}
```

However, Loop Until is often easier to understand and unlike the above, can be used with a single-line action. For example:

```
Loop  
x *= 2
```

```
Until x > y
```

Until can be used with any Loop or For. For example:

```
Loop, Read, %A_ScriptFullPath%  
    lines .= A_LoopReadLine . "`n"  
Until A_Index=5 ; Read the first five lines.  
MsgBox % lines
```

If [A_Index](#) is used in *Expression*, it contains the index of the iteration which has just finished.

Related

[Loop](#), [While-loop](#), [For-loop](#), [Break](#), [Continue](#), [Blocks](#), [Files-and-folders loop](#), [Registry loop](#), [File-reading loop](#), [Parsing loop](#), [If \(expression\)](#)

While-loop [v1.0.48+]

Performs a series of commands repeatedly until the specified [expression](#) evaluates to false.

```
while Expression  
while(Expression)
```

Parameters

Expression

Any valid [expression](#). For example: `while x < y`.

Remarks

The expression is evaluated once before each iteration. If the expression evaluates to true (which is any result other than an empty string or the number 0), the body of the loop is executed; otherwise, execution jumps to the line following the loop's body.

A while-loop 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 statement does not require a block (an "if" and its "else" count as a single statement for this purpose).

The One True Brace (OTB) style may optionally be used, which allows the open-brace to appear on the same line rather than underneath. For example: `while x < y {`.

The built-in variable **A_Index** contains the number of the current loop iteration. It contains 1 the first time the loop's expression and body are executed. For the second time, it contains 2; and so on. If an inner loop is enclosed by an outer loop, the inner loop takes precedence. **A_Index** works inside all types of loops, but contains 0 outside of a loop.

As with all loops, [Break](#) may be used to exit the loop prematurely. Also, [Continue](#) may be used to skip the rest of the current iteration, at which time `A_Index` is increased by 1 and the while-loop's expression is re-evaluated. If it is still true, a new iteration begins; otherwise, the loop ends.

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-reading loops](#) can operate on the entire contents of a file, one line at a time. Finally, [parsing loops](#) can operate on the individual fields contained inside a delimited string.

Related

[Until](#), [Break](#), [Continue](#), [Blocks](#), [Loop](#), [For-loop](#), [Files-and-folders loop](#), [Registry loop](#), [File-reading loop](#), [Parsing loop](#), [If \(expression\)](#)

Examples

```
; As the user drags the left mouse button, a  
ToolTip displays the size of the region inside the  
drag-area.
```

```
CoordMode, Mouse, Screen
```

```
~LButton::  
    MouseGetPos, begin_x, begin_y  
    while GetKeyState("LButton")  
    {  
        MouseGetPos, x, y  
        ToolTip, % begin_x ", " begin_y "`n"  
Abs(begin_x-x) " x " Abs(begin_y-y)  
        Sleep, 10  
    }  
    ToolTip  
return
```

GUI

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

- [New](#): Creates a new window.
- [Add](#): Creates a control such as text, button, or checkbox.
- [Show](#): Displays the window. It can also minimize, maximize, or move the window.
- [Submit](#): Saves the user's input and optionally hides the window.
- [Cancel](#) (or [Hide](#)): Hides the window.
- [Destroy](#): Deletes the window.
- [Font](#): Sets the typeface, size, style, and text color for subsequently created controls.
- [Color](#): Sets the background color for the window and/or its controls.
- [Margin](#): Sets the margin/spacing used whenever no explicit position has been specified for a control.
- [Options and styles for a window](#): Sets various options for the appearance and behavior of the window.
- [Menu](#): Adds or removes a menu bar.
- [Minimize / Maximize / Restore](#): Performs the indicated operation on the window.
- [Flash](#): Blinks the window and its taskbar button.
- [Default](#): Changes the current thread's default GUI window name.
- [Positioning and sizing of controls](#)
- [Storing and responding to user input](#): variables and g-labels
- [Options and styles for controls](#)
- [Window Events](#): [GuiClose](#) | [GuiEscape](#) | [GuiSize](#) | [GuiContextMenu](#) | [GuiDropFiles](#)
- [Creating Multiple GUI windows](#)
- [GUI Events, Threads, and Subroutines](#)
- [Miscellaneous](#): [Keyboard Navigation](#) | [Window Appearance](#) | [General Remarks](#)
- [Examples](#): Contains working scripts that demonstrate GUI windows and controls.

Gui, New [, Options, Title] [v1.1.04+]

Creates a new window and sets it as the [default](#) for the current thread.

```
Gui, New, Options, Title ; Creates a new
unnamed and unnumbered GUI.
Gui, Name: New, Options, Title ; Creates a new
GUI, destroying any existing GUI with that name.
```

Options can contain any of the [options](#) supported by the main GUI command.

If **Title** is omitted, the script's file name is used.

This command comes with the following caveats:

- In most cases the window is created automatically on demand, so it is usually unnecessary to call *Gui New*.
- Although the new window is set as the [default](#) for the *current* thread, non-GUI threads still [default to GUI number 1](#).
- If the GUI has no name and is not the [default GUI](#), it must be identified by its HWND. Use the [+HwndGuiHwnd](#) option to store the HWND of the new window in *GuiHwnd*. `Gui, %GuiHwnd%:Default` can be used to make the other Gui commands operate on it by default.

On the positive side:

- Calling *Gui New* ensures that the script is creating a new GUI, not modifying an existing one. It might also make the script's purpose clearer to other people who read your code (or future you).
- *Gui New* eliminates the need to pick a unique name or number for each GUI. This can be especially useful if the script needs to create more than one GUI, or is intended to be included in other scripts.
- Sometimes it is more intuitive to set the window's title when the GUI is created instead of when it is [shown](#).

If no name is given, the following applies:

- Special *labels* such as *GuiClose* have the default "Gui" prefix unless overridden by *+LabelPrefix* in the options.
- Whenever the GUI launches a new thread, *A_Gui* contains a HWND instead of a name.

Note: Prior to [v1.1.08], this command did not set the default Gui if a name was specified.

Gui, Add, ControlType [, Options, Text]

Adds a control to a GUI window (first creating the GUI window itself, if necessary).

ControlType is one of the following:

- [Text](#), [Edit](#), [UpDown](#), [Picture](#)
- [Button](#), [Checkbox](#), [Radio](#)
- [DropDownList](#), [ComboBox](#)
- [ListBox](#), [ListView](#), [TreeView](#)
- [Link](#), [Hotkey](#), [DateTime](#), [MonthCal](#)
- [Slider](#), [Progress](#)
- [GroupBox](#), [Tab](#), [StatusBar](#)
- [ActiveX](#) (e.g. Internet Explorer Control)
- [Custom](#)

For example:

```
Gui, Add, Text,, Please enter your name:
Gui, Add, Edit, vName
Gui, Show
```

Gui, Show [, Options, Title]

Unless otherwise specified in *Options*, this command makes the window visible, unminimizes it (if necessary), [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).

Omit the X, Y, W, and H options below 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.

Zero or more of the following strings may be present in *Options* (specify each number as decimal, not hexadecimal):

Wn: Specify for **n** the width (in pixels) of the window's client area (the client area excludes the window's borders, title bar, and [menu bar](#)).

Hn: Specify for **n** the height of the window's client area, in pixels.

Xn: Specify for **n** the window's X-position on the screen, in pixels. Position 0 is the leftmost column of pixels visible on the screen.

Yn: Specify for **n** the window's Y-position on the screen, in pixels. Position 0 is the topmost row of pixels visible on the screen.

Center: Centers the window horizontally and vertically on the screen.

xCenter: Centers the window horizontally on the screen. For example: `Gui, Show, xCenter y0`.

yCenter: Centers the window vertically on the screen.

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. For example:

`Gui, Show, AutoSize Center`

One of the following may also be present:

Minimize: Minimizes the window and activates the one beneath it.

Maximize: Maximizes and activates the window.

Restore: Unminimizes or unmaximizes the window, if necessary. The window is also shown and activated, if necessary.

NoActivate: Unminimizes or unmaximizes the window, if necessary. The window is also shown without activating it.

NA: Shows the window without activating it. If the window is minimized, it will stay that way but will probably rise higher in the z-order (which is the order seen in the alt-tab selector). If the window was previously hidden, this will probably cause it to appear on top of the active window even though the active window is not deactivated.

Hide: Hides the window and activates the one beneath it. This is identical in function to [Gui Cancel](#) except that it allows a hidden window to be moved, resized, or given a new title without showing it. For example: `Gui, Show, Hide x55 y66 w300 h200, New Title`.

Gui, Submit [, NoHide]

Saves the contents of each control to its [associated variable](#) (if any) and hides the window unless the NoHide option is present. For controls that produce multiple fields of output, such as a [multi-select ListBox](#), the output uses the window's [current delimiter](#). If the window does not exist -- perhaps due to having been destroyed via [Gui Destroy](#) -- this command has no effect.

Gui, Cancel

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.

Gui, Destroy

Removes the window (if it exists) and all its controls, freeing the corresponding memory and system resources. If the script later recreates the window, all of the window's properties such as color and font will start off at their defaults (as though the window never existed). If `Gui Destroy` is not used, all GUI windows are automatically destroyed when the script exits.

Gui, Font [, Options, FontName]

Sets the font typeface, size, style, and/or color for controls added to the window from this point onward. For example:

```
gui, font, s10, Verdana ; Set 10-point Verdana.
```

Omit the last two 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`, `cFFFFFFAA`, `cDefault`. Note: [Buttons](#) do not obey custom colors. Also, an individual control can be created with a font color other than the current one by including the C option. For example: `Gui, Add, Text, cRed, My Text`.

S: Size (in points). For example: `s12` (specify decimal, not hexadecimal)

W: Weight (boldness), which is a number between 1 and 1000 (400 is normal)

and 700 is bold). For example: `w600` (specify decimal, not hexadecimal)

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. For example, specifying `norm italic` would set the font to normal then to italic.

To specify more than one option, include a space between each. For example: `cBlue s12 bold`.

If a script creates [multiple GUI windows](#), each window remembers its own "current font" for the purpose of creating more controls.

On a related note, the operating system offers standard dialog boxes that prompt the user to pick a font, color, or icon. These dialogs can be displayed via `DllCall()` as demonstrated at www.autohotkey.com/forum/topic17230.html.

Q: [AHK_L 19+]: Text rendering quality. For example: `q3`. Q should be followed by a number from the following table:

0 = DEFAULT_QUALITY	Appearance of the font does not matter.
1 = DRAFT_QUALITY	Appearance of the font is less important than when the PROOF_QUALITY value is used.
2 = PROOF_QUALITY	Character quality of the font is more important than exact matching of the logical-font attributes.
3 = NONANTIALIASED_QUALITY	Font is never antialiased, that is, font smoothing is not done.
4 = ANTIALIASED_QUALITY	Font is antialiased, or smoothed, if the font supports it and the size of the font is not too small or too large.
5 = CLEARTYPE_QUALITY	Windows XP and later: If set, text is rendered (when possible) using ClearType antialiasing method.

For more details of what these values mean, see [MSDN: CreateFont](#).

Since the highest quality setting is usually the default, this feature is more typically used to disable anti-aliasing in specific cases where doing so makes the text clearer.

Gui, Color [, WindowColor, ControlColor]

Sets the background color of the window and/or its controls. *WindowColor* is used as the background for the GUI window itself. *ControlColor* is applied to all present and future controls in the window (though some types of controls do not support a custom color). Although *ControlColor* is initially obeyed by [ListView](#)s and [TreeView](#)s, subsequent changes to *ControlColor* do not affect them. In such cases, use `GuiControl +BackgroundFF9977, MyListView` to explicitly change the 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`, `FFFFFFAA`, `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 color via [Gui, Color](#), buttons will also become transparent. To prevent this, first assign a custom color and then make that color transparent. For example:

```
Gui, Color, EEAA99
Gui +LastFound ; Make the GUI window the last
found window for use by the line below.
WinSet, TransColor, EEAA99
```

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, GuiName:-Caption if
```

it isn't the default window.

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

Gui, Margin [, X, Y]

X and **Y** are the number of pixels of space to leave at the left/right and top/bottom sides of the window when auto-positioning any control that lacks an explicit **X** or **Y coordinate**. Also, the margins are used to determine the vertical and horizontal distance that separates auto-positioned controls from each other. Finally, the margins are taken into account by the first use of **Gui Show** to calculate the window's size (when no explicit size is specified).

Gui, Margin affects only the **default window**, while **Gui, Name:Margin** affects only the **named window**. If this command is not used, 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).

Although the margin may be changed during the course of adding controls, the change will affect only controls added in the future, not ones that already exist. Finally, either **X** or **Y** may be blank to leave the corresponding margin unchanged.

Gui, +/-Option1 +/-Option2 ...

One or more options may be specified immediately after the GUI command. For performance reasons, it is better to set all options in a single line, and to do so 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. For example:

```
Gui +Resize -MaximizeBox ; Change the settings
of the default GUI window.
Gui MyGui:+Resize -MaximizeBox ; Change the
settings of the GUI named MyGui.
```

AlwaysOnTop: Makes the window stay on top of all other windows, which is the same effect as [WinSet AlwaysOnTop](#).

Border: Provides a thin-line border around the window. This is not common.

Caption (present by default): Provides a title bar and a thick window border/edge. When removing the caption from a window that will use [WinSet TransColor](#), remove it only after setting the TransColor.

Delimiter: Specifies that the window should use a field separator other than pipe (|) whenever controls' contents are added via [Gui Add](#), modified via [GuiControl](#), or retrieved via [Gui Submit](#) or [GuiControlGet](#). Specify a single character immediately after the word Delimiter. For example, `Gui +Delimiter`n` would use a linefeed character, which might be especially appropriate with [continuation sections](#). Similarly, `Gui +Delimiter|` would revert to the default delimiter. To use space or tab, specify `Gui +DelimiterSpace` or `Gui +DelimiterTab`. Once the delimiter is changed, it affects all existing and subsequent [threads](#) that operate on this particular window.

Disabled: Disables the window, which prevents the user from interacting with its

controls. This is often used on a window that owns other windows (see [Owner](#)).

DPIScale [v1.1.11+]: Use `Gui -DPIScale` to disable DPI scaling, which is enabled by default. If DPI scaling is enabled on a system with a non-standard DPI setting, the Gui commands automatically scale coordinates and sizes to give controls roughly the same apparent size (but higher resolution). For example, with a DPI of 144 (150%), `Gui Show, w100` would make the Gui 150 pixels wide, but `A_GuiWidth` would still return 100. `A_ScreenDPI` contains the system's current DPI.

DPI scaling only applies to Gui commands, so coordinates coming directly from other sources such as `ControlGetPos` or `WinGetPos` will not work. There are a number of ways to deal with this:

- Avoid using hard-coded coordinates wherever possible. For example, use the `xp`, `xs`, `xm` and `x+m` options for positioning controls and specify height in `rows of text` instead of pixels.
- Enable (`Gui +DPIScale`) and disable (`Gui -DPIScale`) scaling on the fly, as needed. Changing the setting does not affect positions or sizes which have already been set.
- Manually scale the coordinates. For example, `x*(A_ScreenDPI/96)` converts x from logical/Gui coordinates to physical/non-Gui coordinates.

HwndOutputVar [v1.1.04+]: This option stores the window handle (HWND) of the GUI in `OutputVar`. For example: `Gui +HwndMyGuiHwnd`. When within a function, `MyGuiHwnd` is treated as a [function dynamic variable](#). A GUI's HWND is often used with `PostMessage`, `SendMessage`, and `DllCall`. It can also be used directly as an `ahk_id WinTitle` or in place of a GUI name; for example, `Gui %MyGuiHwnd%:Destroy`.

Label [v1.0.44.09+]: Sets custom names for this window's [special labels](#). For example, `Gui MyGui:+LabelMyGui_On` would use the labels `MyGui_OnClose` and `MyGui_OnSize` (if they exist) instead of `MyGuiGuiClose` and `MyGuiGuiSize`. In other words, the string "MyGuiGui" is replaced by "MyGui_On" in the names of all [special labels](#). This can also be used to make multiple windows share the same set of labels (in which case the script may consult `A_Gui` to determine which window launched the subroutine).

LastFound: Sets the window to be the [last found window](#) (though this is

unnecessary in a [Gui thread](#) because it is done automatically), which allows commands such as [WinSet](#) to operate on it even if it is hidden (that is, [DetectHiddenWindows](#) is not necessary). This is especially useful for changing the properties of the window before showing it. For example:

```
Gui +LastFound
WinSet, TransColor, %CustomColor% 150
Gui Show
```

LastFoundExist [v1.0.43.09+]: Unlike other options, LastFoundExist is recognized only when no other options appear on the same line. *+LastFoundExist* is the same as *+LastFound* except that the window is not created if it does not already exist. The main use for this is to detect whether a particular GUI window exists. For example:

```
Gui MyGui:+LastFoundExist
IfWinExist
    MsgBox GUI "MyGui" already exists.
```

MaximizeBox: Enables the maximize button in the title bar. This is also included as part of *Resize* below.

MinimizeBox (present by default): Enables the minimize button in the title bar.

MinSize and **MaxSize** [v1.0.44.13+]: Determines the minimum and/or maximum size of the window, such as when the user drags its edges to resize it. Specify the word *MinSize* and/or *MaxSize* with no suffix to use the window's current size as the limit (if the window has no current size, it will use the size from the first use of [Gui Show](#)). Alternatively, append the width, followed by an X, followed by the height; for example: `Gui +Resize +MinSize640x480`. The dimensions are in pixels, and they specify the size of the window's client area (which excludes borders, title bar, and [menu bar](#)). Specify each number as decimal, not hexadecimal.

Either the width or the height may be omitted to leave it unchanged (e.g. `+MinSize640x` or `+MinSizex480`). Furthermore, Min/MaxSize can be specified more than once to use the window's current size for one dimension and

an explicit size for the other. For example, `+MinSize +MinSize640x` would use the window's current size for the height and 640 for the width.

If `MinSize` and `MaxSize` are never used, the operating system's defaults are used (similarly, `Gui -MinSize -MaxSize` can be used to return to the defaults).

Note: the window must have `+Resize` to allow resizing by the user.

OwnDialogs: `Gui +OwnDialogs` should be specified in each `thread` (such as a `ButtonOK` subroutine) for which subsequently displayed `MsgBox`, `InputBox`, `FileSelectFile`, and `FileSelectFolder` dialogs should be owned by the window. Such dialogs become modal, meaning that the user cannot interact with the GUI window until dismissing the dialog. By contrast, `ToolTip`, `Progress`, and `SplashImage` windows do not become modal even though they become owned; they will merely stay always on top of their owner. In either case, any owned dialog or window is automatically destroyed when its GUI window is `destroyed`.

There is typically no need to turn this setting back off because it does not affect other `threads`. However, if a thread needs to display both owned and unowned dialogs, it may turn off this setting via `Gui -OwnDialogs`.

If no window name prefix is specified -- such as using `Gui +OwnDialogs` rather than `Gui MyGui:+OwnDialogs` -- the `thread's default window` will own the dialogs.

Owner: Use `+Owner` to make the window owned by another. 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. `+Owner` must be used after the window's owner is created, but [v1.1.05] and later allow it to be used before or after the owned window is created. There are two ways to use `+Owner` as shown in these examples:

```
Gui, MyGui:+OwnerMyOtherGui ; Make MyGui owned
by MyOtherGui.
Gui, MyGui:+Owner ; Make MyGui owned by
script's main window to prevent display of a
taskbar button.
```

[v1.1.03+]: `+Owner` can be immediately followed by the `name` or number of an existing Gui or the `HWND` of any top-level window.

Compatibility note: In [v1.1.03] and later, `+Owner` removes the `WS_CHILD` style and sets the `WS_POPUP` style. To set the parent window of a Gui, scripts must either use the `+Parent` option, or override the appropriate styles *after* the `+Owner` option.

To prevent the user from interacting with the owner while one of its owned window is visible, disable the owner via `Gui +Disabled`. Later (when the time comes to cancel or destroy the owned window), re-enable the owner via `Gui -Disabled`. Do this prior to cancel/destroy so that the owner will be reactivated automatically.

Parent [v1.1.03+]: Use `+Parent` immediately followed by the `name` or number of an existing Gui or the `HWND` of any window or control to use it as the parent of this window. To convert the Gui back into a top-level window, use `-Parent`. This option works even after the window is created.

Resize: Makes the window resizable and enables its maximize button in the title bar. To avoid enabling the maximize button, specify `+Resize -MaximizeBox`.

SysMenu (present by default): Specify `-SysMenu` (minus SysMenu) to omit the system menu and icon in the window's upper left corner. This will also omit the minimize, maximize, and close buttons in the title bar.

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 that obey the current theme, turn it back on via `+Theme`.

Note: The Theme 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.

Finally, this setting may be changed for an individual control by specifying

+Theme or -Theme in its options when it is created.

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](#).

(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 styles are not documented here (since they are rarely used), they can be discovered by searching for WS_EX_APPWINDOW at www.microsoft.com.

Gui, Menu [, MenuName]

Attaches a menu bar to the window. Use the [Menu](#) command to create an ordinary menu for this purpose. For example:

```
Menu, FileMenu, Add, &Open`tCtrl+O, MenuFileOpen
; See remarks below about Ctrl+O.
Menu, FileMenu, Add, E&xit, MenuHandler
Menu, HelpMenu, Add, &About, MenuHandler
Menu, MyMenuBar, Add, &File, :FileMenu ; Attach
the two sub-menus that were created above.
Menu, MyMenuBar, Add, &Help, :HelpMenu
Gui, Menu, MyMenuBar
```

In the first line above, notice that `&Open` is followed by `Ctrl+O` (with a tab character in between). This indicates a keyboard shortcut that the user may press instead of selecting the menu item. [v1.1.04+]: If the shortcut uses only the standard modifier key names Ctrl, Alt and Shift, it is automatically registered as a *keyboard accelerator* for the GUI. Single-character accelerators with no modifiers are case-sensitive and can be triggered by unusual means such as IME or `Alt` ██████N.

If a particular key combination does not work automatically, the use of a [context-sensitive hotkey](#) may be required. However, such hotkeys typically cannot be triggered by [Send](#) and are more likely to interfere with other scripts than a standard keyboard accelerator.

To remove a window's current menu bar, use `Gui Menu` (that is, omit the last 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 (however, 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 for everything else.

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 (and in some cases, its submenus) is not supported and will cause an error dialog to be displayed (unless [UseErrorLevel](#) is in effect). Use the following steps to make such changes: 1) detach the menu bar via `Gui Menu` (that is, omit *MenuName*); 2) make the changes; 3) reattach the menu bar via `Gui, Menu, MyMenuBar` .

Gui, Hide / Minimize / Maximize / Restore

`Gui Hide` is equivalent to `Gui Cancel`. The other three commands unhide the window (if necessary) then perform the indicated operation on it. If the window does not exist -- perhaps due to having been destroyed via `Gui Destroy` -- these commands have no effect.

Gui, Flash [, Off]

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 three times because each pair of flashes inverts then restores its appearance:

```
Loop 6
{
    Gui Flash
    Sleep 500 ; It's quite sensitive to this
value; altering it may change the behavior in
unexpected ways.
}
```

Gui, GuiName:Default

Changes the [current thread](#)'s default [GUI window name](#), which is used whenever a window name is not specified for [GuiControl](#), [GuiControlGet](#), and the `Gui` command itself. In the following example, the default window name is changed to "MyGui": `Gui MyGui:Default` . See [thread's default window](#) for more information about the default window.

[v1.1.03+]: *GuiName* can be the [HWND](#) of one of the script's GUI windows. If the window has a name, it becomes the default name and remains so even after the window itself is destroyed. If the window has no name, the default name reverts to 1 when the window is destroyed.

[v1.1.23+]: The built-in variable [A_DefaultGui](#) contains the name or number of the default GUI.

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; that is, "what you see is what you get". SmartGUI Creator is free and can be downloaded from www.autohotkey.com/docs/SmartGUI/

Positioning and Sizing of Controls

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 or later, 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 visibly 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.

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. 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 right or bottom 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. Since negative numbers such as **X-10** are reserved for absolute positioning, to use a negative offset, include a plus sign in front of it. For example: **X+ -10**.

[v1.1.16+]: For **X+** and **Y+**, the letter **M** can be used as a substitute for the window's current [margin](#). For example, **x+m** uses the right edge of the previous

control plus the standard padding distance. **xp y+m** positions a control below the previous control, whereas specifying an X coordinate on its own would normally imply **yp** by default.

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). 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.

xs and **ys**: these 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. For 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; this uses the previous section for itself but establishes a new section for subsequent controls.

Storing and Responding to User Input

V: Variable. Associates a variable with a control. Immediately after the letter V, specify the name of a global variable (or a [ByRef local](#) that points to a global, or [in v1.0.46.01+] a [static variable](#)). For example, specifying `vMyEdit` would store the control's contents in the variable *MyEdit* whenever the [Gui Submit](#) command is used. If a control is not input-capable -- such as a Text control or GroupBox -- associating a variable with it can still be helpful since that variable's name serves as the control's unique identifier for use with [GuiControl](#), [GuiControlGet](#), and [A_GuiControl](#).

Note: [Gui Submit](#) does not change the contents of variables of non-input-capable controls (such as Text and GroupBox), nor certain others as documented in their sections (such as [ListView](#) and [TreeView](#)).

G: Gosub (g-label). Launches a subroutine or function automatically when the user clicks or changes a control. Immediately after the letter G, specify the name of the [label](#) to execute. `gCancel` may be specified to perform an implicit [Gui Cancel](#) (but if a label named "Cancel" exists in the script, it will be executed instead). The subroutine may consult the following built-in variables: [A_Gui](#), [A_GuiControl](#), [A_GuiEvent](#), and [A_EventInfo](#).

[v1.1.20+]: If not a valid label name, a function name can be used instead. Alternatively, the [GuiControl](#) command can be used to associate a [function object](#) with the control. The function can optionally accept the following parameters (where `gCtrlEvent` sets the function):

```
CtrlEvent(CtrlHwnd, GuiEvent, EventInfo,  
ErrLevel:= "")
```

The meanings of the parameters depends on the type of control. Note that if the fourth parameter is declared without a default value, the function will only be called by events which supply four parameters.

Controls: Common Styles and Other Options

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.

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 retrieves the contents of such a control.

C: Color of text (has no effect on `buttons`). Specify the letter `C` followed immediately by a color name (see `color chart`) or RGB value (the `0x` prefix is optional). Examples: `cRed`, `cFF2211`, `c0xFF2211`, `cDefault`.

Disabled: Makes an input-capable control appear in a disabled state, which prevents the user from focusing or modifying its contents. Use `GuiControl Enable` to enable it later. Note: To make an Edit control read-only, specify the string `ReadOnly` instead. Also, the word `Disabled` may optionally be followed immediately by a 0 or 1 to indicate the starting state (0 for enabled and 1 for disabled). In other words, `Disabled` and `Disabled%VarContainingOne%` are the same.

Hidden: The control is initially invisible. Use `GuiControl Show` to show it later. The word `Hidden` may optionally be followed immediately by a 0 or 1 to indicate the starting state (0 for visible and 1 for hidden). In other words, `Hidden` and `Hidden%VarContainingOne%` are the same.

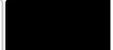
Left: Left-justifies the control's text within its available width. This option affects the following controls: `Text`, `Edit`, `Button`, `Checkbox`, `Radio`, `UpDown`, `Slider`, `Tab`, `Tab2`, `GroupBox`, `DateTime`.

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. This option affects the following controls: `Text`, `Edit`,

Button, Checkbox, Radio, UpDown, Slider, Tab, Tab2, GroupBox, DateTime, Link.

Center: Centers the control's text within its available width. This option affects the following controls: Text, Edit, Button, Checkbox, Radio, Slider, GroupBox.

Section: Starts a new section and saves this control's position for later use with the `xs` and `ys` positioning options described [above](#).

Tabstop: Use `-Tabstop` (i.e. minus Tabstop) to have an input-capable control skipped over when the user presses the `Tab`  navigate.

Wrap: Enables word-wrapping of the control's contents within its available width. Since nearly all control types start off with word-wrapping enabled, use `-Wrap` 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, `HScroll1500` 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 (though the mere presence of `HScroll` makes it possible for the horizontal scroll bar to be added later via `GuiControl, +HScroll1500, MyScrollBar`, which is otherwise impossible).

Controls: Uncommon Styles and Options

BackgroundTrans: 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. Known limitation: BackgroundTrans might not work properly for controls inside a [Tab control](#) that contains a [ListView](#).

-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. When adding a border to an *existing* control, it might be necessary to increase the control's width and height by 1 pixel.

HwndOutputVar [v1.0.46.01+]: When used with [Gui Add](#), this option stores the window handle (HWND) of the newly created control in *OutputVar*. For example: `Gui, Add, Edit, vMyEdit HwndMyEditHwnd`. When within a function, *MyEditHwnd* is treated as a [function dynamic variable](#). A control's HWND is often used with [PostMessage](#), [SendMessage](#), and [DllCall](#). It can also be used directly as an [ahk_id WinTitle](#) (this also works on hidden controls even when [DetectHiddenWindows](#) is Off) or [in v1.1.04+] with [GuiControl](#) and [GuiControlGet](#) as the *ControlID* parameter. On a related note, a parent window's HWND can be retrieved via [Gui MyGui:+HwndOutputVar](#).

Theme: This option can be used to override the window's current theme setting for the newly created control. It has no effect when used on an existing control; however, this may change in a future version. See [Gui +/-Theme](#) for details.

(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 styles are not documented here (since they are rarely used), they can be discovered by searching for `WS_EX_CLIENTEDGE` at www.microsoft.com.

Window Events

The following labels (subroutines) will be automatically associated with a GUI window if they exist in the script:

- [GuiClose](#)
- [GuiEscape](#)
- [GuiSize](#)
- [GuiContextMenu](#)
- [GuiDropFiles](#)

[v1.1.20+]: If a label does not exist for a given event, a function with that name can be called instead. The function can optionally receive the [HWND](#) of the GUI as its first parameter. Some events have additional parameters.

For windows [other than number 1](#), the window's name or number (if it has one) is used as a prefix for the special labels mentioned above; for example, `2GuiEscape` and `2GuiClose` would be the default labels for window number 2, while `MyGuiGuiEscape` and `MyGuiGuiClose` would be the default labels for `MyGui`. To set a custom prefix, use [Gui +Label](#).

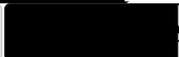
GuiClose: Launched when the window is closed by any of the following: pressing its X button in the title bar, selecting "Close" from its system menu, or closing it with [WinClose](#). If this label is absent, closing the window simply hides it, which is the same effect as [Gui Cancel](#). One of the most common actions to take in response to `GuiClose` is [ExitApp](#); for example:

```
GuiClose:
ExitApp
```

[v1.1.20+]: If `GuiClose` is a function, the GUI is hidden by default. The function can prevent this by returning a non-zero integer, as in the example below:

```
GuiClose(GuiHwnd) { ; Declaring this parameter
is optional.
    MsgBox 4,, Are you sure you want to hide the
```

```
GUI?  
  IfMsgBox No  
    return true ; true = 1  
}
```

GuiEscape: Launched when the user presses **Escape**  GUI window is active. If this label is absent, pressing **Escape**  ct. Known limitation: If the first control in the window is disabled (possibly depending on control type), the GuiEscape label will not be launched. There may be other circumstances that produce this effect.

GuiSize: Launched when the window is resized, minimized, maximized, or restored. The built-in 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, [A_EventInfo](#) and [ErrorLevel](#) will both 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.

A script may use GuiSize to reposition and resize controls in response to the user's resizing of the window. This process can be made easier by using [AutoXYWH\(\)](#) by [tmlinshi](#) and [toralf](#).

When the window is resized (even by the script), GuiSize might not be called immediately. As with other window events, if the current thread is [uninterruptible](#), GuiSize won't be called until the thread becomes interruptible. If the script has just resized the window, follow this example to ensure GuiSize is called immediately:

```
Critical Off ; Even if Critical On was never  
used.  
Sleep -1
```

[v1.1.22.05+]: Gui Show automatically does a **Sleep -1**, so it is generally


```
GuiContextMenu(GuiHwnd, CtrlHwnd, EventInfo,  
IsRightClick, X, Y)
```

CtrlHwnd is blank if the event was not received by a control. *IsRightClick* is true if *A_GuiEvent* is *RightClick*.

GuiDropFiles: 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 following built-in variables are available within *GuiDropFiles*:

1. *A_GuiControl*, which contains the [text or variable name](#) of the control upon which the files were dropped (blank if none).
2. *A_EventInfo* and *ErrorLevel*, which both contain the number of files dropped.
3. *A_GuiX* and *A_GuiY*, which contain the X and Y coordinates of where the files were dropped (relative to the window's upper left corner).
4. *A_GuiEvent*, which contains the names of the files that were dropped, with each filename except the last terminated by a linefeed (`\n`).

To extract the individual files, use a [parsing loop](#) as shown below:

```
; EXAMPLE #1:  
Loop, parse, A_GuiEvent, `n  
{  
    MsgBox, 4,, File number %A_Index%  
is:`n%A_LoopField%.`n`nContinue?  
    IfMsgBox, No, Break  
}  
  
; EXAMPLE #2: To extract only the first file,  
follow this example:  
Loop, parse, A_GuiEvent, `n  
{  
    FirstFile = %A_LoopField%  
    Break  
}
```

```
; EXAMPLE #3: To process the files in
alphabetical order, follow this example:
FileList = %A_GuiEvent%
Sort, FileList
Loop, parse, FileList, `n
    MsgBox File number %A_Index%
is: `n%A_LoopField%.
```

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` .

[v1.1.20+]: If `GuiDropFiles` is a function, the parameters are as shown in the example below. `CtrlHwnd` is blank if files were dropped on the GUI itself. `FileArray` is an [array \(object\)](#) of filenames, where `FileArray[1]` is the first file and `FileArray.MaxIndex()` returns the number of files. A [for-loop](#) can be used to iterate through the files:

```
GuiDropFiles(GuiHwnd, FileArray, CtrlHwnd, X, Y)
{
    for i, file in FileArray
        MsgBox File %i% is: `n%file%
}
```

Detecting and responding to other events: Other types of GUI events can be detected and acted upon via [OnMessage\(\)](#). For example, a script can display context-sensitive help via `ToolTip` whenever the user moves the mouse over particular controls in the window. This is demonstrated in the [GUI ToolTip example](#).

Creating Multiple GUI Windows

To operate upon a window other than the [default](#), include its name or number (or [in v1.1.03+] its [HWND](#)) followed by a colon in front of the sub-command, as in these examples:

```
Gui, MyGui:Add, Text,, Text for about-box.  
Gui, MyGui:Show
```

[Gui MyGui:Default](#) can be used to avoid the need for the "MyGui:" prefix above. In addition, the prefix is not necessary inside a [GUI thread](#) that operates upon the same window that launched the thread.

[v1.1.03+]: Gui names must conform to the same rules as [variable names](#). Any number which is either not between 1 and 99 or is longer than two characters (such as 0x01) must be the HWND of an existing Gui, or the command will fail. The number of windows that can be created is limited only by available system resources.

[v1.1.04+]: Any number of unnamed GUI windows can be created using [Gui, New](#).

GUI Events, Threads, and Subroutines

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 name** for a GUI thread is that of the window that launched the thread. Non-GUI threads use 1 as their default.

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 (even if it is hidden).

Clicking on a control while its [g-label](#) is already running from a prior click will have no effect and the event is discarded. To prevent this, use [Critical](#) as the subroutine's first line (however, this will also buffer/defer other [threads](#) such as the press of a hotkey).

The built-in variables [A_Gui](#) and [A_GuiControl](#) contain the window name and Control ID that launched the current thread. For details see [A_Gui](#) and [A_GuiControl](#).

To have multiple events perform the same subroutine, specify their labels consecutively above the subroutine. For example:

```
GuiEscape:  
GuiClose:  
ButtonCancel:  
ExitApp ; All of the above labels will do this.
```

All GUI [threads](#) start off fresh with the default values for settings such as [SendMode](#). These defaults can be changed in the [auto-execute section](#).

Keyboard Navigation

A GUI window may be navigated via the **Tab**  which moves keyboard focus to the next input-capable control (controls from which the **Tabstop** style has been removed 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 (which most control types have by default) will have keyboard focus.

Certain controls may contain an ampersand (&) to create a keyboard shortcut, which might be displayed in the control's text as an underlined character (depending on system settings). A user activates the shortcut by holding down the **Alt**  when 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**.

Window Appearance

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. For example: `Menu, Tray, Icon, MyIcon.ico`. It is also possible to have a different large icon for a window than its small icon (the large icon is displayed in the alt-tab task switcher). This can be done via `LoadPicture` and `SendMessage`; for example:

```
iconsize := 32 ; Ideal size for alt-tab varies
between systems and OS versions.
hIcon := LoadPicture("My Icon.ico", "Icon1 w"
iconsize " h" iconsize, imgtype)
Gui +LastFound
SendMessage 0x80, 1, hIcon ; 0x80 is
WM_SETICON; and 1 means ICON_BIG (vs. 0 for
ICON_SMALL).
Gui Show
```

Due to OS limitations, 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.

Related topic: [window's margin](#).

General Remarks

Use `GuiControl` and `GuiControlGet` to operate upon individual controls in a GUI window.

Each GUI window may have up to 11,000 controls. However, use caution when creating more than 5000 controls because system instability may occur for certain control types.

Any script that uses the GUI command anywhere is automatically `persistent` (even if the GUI command is never actually executed). It is also single-instance unless the `#SingleInstance` directive has been used to override that.

Related

[GuiControl](#), [GuiControlGet](#), [Menu](#), [Control Types](#), [ListView](#), [TreeView](#), [Control](#), [ControlGet](#), [SplashImage](#), [MsgBox](#), [FileSelectFile](#), [FileSelectFolder](#)

Examples

```
; Example: Achieve an effect similar to  
SplashTextOn:
```

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

```
; Example: A simple input-box that asks for first  
name and last name:
```

```
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: Tab control:

```
Gui, Add, Tab2,, First Tab|Second Tab|Third Tab ;  
Tab2 vs. Tab requires [v1.0.47.05+].  
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 ; Save each control's contents to its  
associated variable.  
MsgBox You  
entered: `n%MyCheckbox%`n%MyRadio%`n%MyEdit%  
ExitApp
```

; Example: ListBox containing files in a
directory:

```
Gui, Add, Text,, Pick a file to launch from the  
list below.`nTo 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_GuiEvent <> 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?`n`n%MyListBox%
```

```
IfMsgBox, 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: Display context-sensitive help (via ToolTip) whenever the user moves the mouse over a particular control:

```
Gui, Add, Edit, vMyEdit
MyEdit_TT := "This is a tooltip for the control
whose variable is MyEdit."
Gui, Add, DropDownList, vMyDDL, Red|Green|Blue
MyDDL_TT := "Choose a color from the drop-down
list."
Gui, Add, Checkbox, vMyCheck, This control has no
tooltip.
Gui, Show
OnMessage(0x200, "WM_MOUSEMOVE")
return

WM_MOUSEMOVE()
{
    static CurrControl, PrevControl, _TT ; _TT is
kept blank for use by the ToolTip command below.
    CurrControl := A_GuiControl
    If (CurrControl <> PrevControl and not
Instr(CurrControl, " "))
    {
        ToolTip ; Turn off any previous tooltip.
        SetTimer, DisplayToolTip, 1000
        PrevControl := CurrControl
    }
    return

DisplayToolTip:
SetTimer, DisplayToolTip, Off
ToolTip % %CurrControl%_TT ; The leading
percent sign tell it to use an expression.
```

```

SetTimer, RemoveToolTip, 3000
return

RemoveToolTip:
SetTimer, RemoveToolTip, Off
ToolTip
return
}

```

```

GuiClose:
ExitApp

```

; Example: On-screen display (OSD) via transparent window:

```

CustomColor = EEAA99 ; Can be any RGB color (it
will be made transparent below).
Gui +LastFound +AlwaysOnTop -Caption +ToolWindow
; +ToolWindow avoids a taskbar button and an alt-
tab menu item.
Gui, Color, %CustomColor%
Gui, Font, s32 ; Set a large font size (32-
point).
Gui, Add, Text, vMyText cLime, XXXXX YYYYYY ; XX &
YY serve to auto-size the window.
; Make all pixels of this color transparent and
make the text itself translucent (150):
WinSet, TransColor, %CustomColor% 150
SetTimer, UpdateOSD, 200
Gosub, UpdateOSD ; Make the first update
immediate rather than waiting for the timer.
Gui, Show, x0 y400 NoActivate ; NoActivate avoids
deactivating the currently active window.

```

```
return
```

```
UpdateOSD:
```

```
MouseGetPos, MouseX, MouseY
```

```
GuiControl,, MyText, X%MouseX%, Y%MouseY%
```

```
return
```

```
; Example: A moving progress bar overlaid on a  
background image.
```

```
Gui, Color, White
```

```
Gui, Add, Picture, x0 y0 h350 w450,
```

```
%A_WinDir%\system32\ntimage.gif
```

```
Gui, Add, Button, Default xp+20 yp+250, Start the  
Bar Moving
```

```
Gui, Add, Progress, vMyProgress w416
```

```
Gui, Add, Text, vMyText wp ; wp means "use width  
of previous".
```

```
Gui, Show
```

```
return
```

```
ButtonStartTheBarMoving:
```

```
Loop, %A_WinDir%\*.*
```

```
{
```

```
    if A_Index > 100
```

```
        break
```

```
    GuiControl,, MyProgress, %A_Index%
```

```
    GuiControl,, MyText, %A_LoopFileName%
```

```
    Sleep 50
```

```
}
```

```
GuiControl,, MyText, Bar finished.
```

```
return
```

```
GuiClose:
```

ExitApp

```
; Example: Simple image viewer:
```

```
Gui, +Resize
Gui, Add, Button, default, &Load New Image
Gui, Add, Radio, ym+5 x+10 vRadio checked, Load
&actual size
Gui, Add, Radio, ym+5 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 - 28 ; Minus 28 to
allow room for borders and margins inside.
    Height = -1 ; "Keep aspect ratio" seems best.
}
GuiControl,, Pic, *w%width% *h%height% %file% ;
```

Load the image.

```
Gui, Show, xCenter y0 AutoSize, %file% ; Resize  
the window to match the picture size.
```

```
return
```

```
GuiClose:
```

```
ExitApp
```

```
; Example: Simple text editor with menu bar.
```

```
; Create the sub-menus for the menu bar:
```

```
Menu, FileMenu, Add, &New, FileNew
```

```
Menu, FileMenu, Add, &Open, FileOpen
```

```
Menu, FileMenu, Add, &Save, FileSave
```

```
Menu, FileMenu, Add, Save &As, FileSaveAs
```

```
Menu, FileMenu, Add ; Separator line.
```

```
Menu, FileMenu, Add, E&xit, FileExit
```

```
Menu, HelpMenu, Add, &About, HelpAbout
```

```
; Create the menu bar by attaching the sub-menus  
to it:
```

```
Menu, MyMenuBar, Add, &File, :FileMenu
```

```
Menu, MyMenuBar, Add, &Help, :HelpMenu
```

```
; Attach the menu bar to the window:
```

```
Gui, Menu, MyMenuBar
```

```
; Create the main Edit control and display the  
window:
```

```
Gui, +Resize ; Make the window resizable.
```

```
Gui, Add, Edit, vMainEdit WantTab W600 R20
```

```
Gui, Show,, Untitled
```

```
CurrentFileName = ; Indicate that there is no  
current file.
```

```
return
```

```
FileNew:
```

```
GuiControl,, MainEdit ; Clear the Edit control.  
return
```

```
FileOpen:
```

```
Gui +OwnDialogs ; Force the user to dismiss the  
FileSelectFile dialog before returning to the main  
window.
```

```
FileSelectFile, SelectedFileName, 3,, Open File,  
Text Documents (*.txt)
```

```
if SelectedFileName = ; No file selected.  
return
```

```
Gosub FileRead  
return
```

```
FileRead: ; Caller has set the variable  
SelectedFileName for us.
```

```
FileRead, MainEdit, %SelectedFileName% ; Read the  
file's contents into the variable.
```

```
if ErrorLevel
```

```
{
```

```
MsgBox Could not open "%SelectedFileName".  
return
```

```
}
```

```
GuiControl,, MainEdit, %MainEdit% ; Put the text  
into the control.
```

```
CurrentFileName = %SelectedFileName%
```

```
Gui, Show,, %CurrentFileName% ; Show file name  
in title bar.
```

```
return
```

```
FileSave:
```

```
if CurrentFileName = ; No filename selected yet,  
so do Save-As instead.
```

```
Goto FileSaveAs
```

```

Gosub SaveCurrentFile
return

FileSaveAs:
Gui +OwnDialogs ; Force the user to dismiss the
FileSelectFile dialog before returning to the main
window.
FileSelectFile, SelectedFileName, S16,, Save File,
Text Documents (*.txt)
if SelectedFileName = ; No file selected.
    return
CurrentFileName = %SelectedFileName%
Gosub SaveCurrentFile
return

SaveCurrentFile: ; Caller has ensured that
CurrentFileName is not blank.
IfExist %CurrentFileName%
{
    FileDelete %CurrentFileName%
    if ErrorLevel
    {
        MsgBox The attempt to overwrite
"%CurrentFileName%" failed.
        return
    }
}
GuiControlGet, MainEdit ; Retrieve the contents
of the Edit control.
FileAppend, %MainEdit%, %CurrentFileName% ; Save
the contents to the file.
; Upon success, Show file name in title bar (in
case we were called by FileSaveAs):
Gui, Show,, %CurrentFileName%
return

HelpAbout:

```

```
Gui, About:+owner1 ; Make the main window (Gui #1) the owner of the "about box".
Gui +Disabled ; Disable main window.
Gui, About:Add, Text,, Text for about box.
Gui, About:Add, Button, Default, OK
Gui, About:Show
return
```

```
AboutButtonOK: ; This section is used by the "about box" above.
```

```
AboutGuiClose:
```

```
AboutGuiEscape:
```

```
Gui, 1:-Disabled ; Re-enable the main window (must be done prior to the next step).
```

```
Gui Destroy ; Destroy the about box.
```

```
return
```

```
GuiDropFiles: ; Support drag & drop.
```

```
Loop, Parse, A_GuiEvent, `n
```

```
{
```

```
    SelectedFileName = %A_LoopField% ; Get the first file only (in case there's more than one).
```

```
    break
```

```
}
```

```
Gosub FileRead
```

```
return
```

```
GuiSize:
```

```
if ErrorLevel = 1 ; The window has been minimized. No action needed.
```

```
    return
```

```
; Otherwise, the window has been resized or maximized. Resize the Edit control to match.
```

```
NewWidth := A_GuiWidth - 20
```

```
NewHeight := A_GuiHeight - 20
```

```
GuiControl, Move, MainEdit, W%NewWidth%
```

```
H%NewHeight%
```

```
return
```

```
FileExit:      ; User chose "Exit" from the File  
menu.
```

```
GuiClose:     ; User closed the window.
```

```
ExitApp
```

GUI Control Types

Table of Contents

- Text, Edit, UpDown, Picture
- Button, Checkbox, Radio
- DropDownList, ComboBox
- ListBox, ListView, TreeView
- Link, Hotkey, DateTime
- MonthCal, Slider, Progress
- GroupBox, Tab3, StatusBar
- ActiveX (e.g. Internet Explorer Control)
- Custom

Text

Description: A region containing borderless text that the user cannot edit. Often used to label other controls. Example:

```
Gui, Add, Text,, Please enter your name:
```

In this case, the last parameter is the string to display. It may contain linefeeds (`\n`) to start new lines. In addition, a single long line can be broken up into several shorter ones by means of a [continuation section](#).

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.

Since the control's contents are in the last parameter of the Gui command, literal commas do not need to be escaped. This is also true for the last parameter of all other commands.

A [g-label](#) such as `gMySubroutine` may be listed in the control's options. This would cause the *MySubroutine* label to be launched automatically whenever the user clicks the text. This can be used to simulate an underlined, blue hyperlink as shown in the following working script:

```
Gui, Font, underline
Gui, Add, Text, cBlue gLaunchGoogle, Click here
to launch Google.

; Alternatively, Link controls can be used:
Gui, Add, Link,, Click <a
href="www.google.com">here</a> to launch Google.
Gui, Font, norm
Gui, Show
return

LaunchGoogle:
```

```
Run www.google.com  
return
```

A double-click can be detected by checking whether [A_GuiEvent](#) contains the word `DoubleClick`.

An ampersand (&) may be used in the text to underline one of its letters. For example:

```
Gui, Add, Text,, &First Name:  
Gui, Add, Edit
```

In the example above, the letter F will be underlined, which allows the user to press the [shortcut key](#)   to set keyboard focus to the first input-capable control that was added after the text control. To instead display a literal ampersand, specify two consecutive ampersands (&&). To disable all special treatment of ampersands, include [0x80](#) in the control's options.

See [general options](#) for other options like *Right*, *Center*, and *Hidden*. See also: [position and sizing of controls](#).

Edit

Description: An area where free-form text can be typed by the user. Example:

```
Gui, Add, Edit, r9 vMyEdit, Text to appear
inside the edit control (omit this parameter to
start off empty).
```

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 treated as part of the input rather than triggering the window's [default button](#).

To start a new line in a multi-line edit control, the last parameter (contents) may contain either a solitary linefeed (``n`) or a carriage return and linefeed (``r`n`). Both methods produce literal ``r`n` pairs inside the Edit control. However, when the control is saved to its variable via [Gui Submit](#) or [GuiControlGet](#), each ``r`n` in the text is always translated to a plain linefeed (``n`). To write the text to a file, follow this example: [FileAppend](#), `%MyEdit%`, `C:\Saved File.txt`.

If the control has word-wrapping enabled (which is the default for multi-line edit controls), any wrapping that occurs as the user types will not produce linefeed characters (only the `Enter` key can do that).

A [g-label](#) such as `gMySubroutine` may be listed in the control's options. This would cause the *MySubroutine* label to be launched automatically whenever the user or the script changes the contents of the control.

TIP: To load a text file into an Edit control, use [FileRead](#) and [GuiControl](#). For example:

```
Gui, Add, Edit, R20 vMyEdit
FileRead, FileContents, C:\My File.txt
```

```
GuiControl,, MyEdit, %FileContents%
```

Edit Options

To remove an option rather than adding it, precede it with a minus sign:

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.

Lowercase: The characters typed by the user are automatically converted to lowercase.

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*).

Number: Prevents the user from typing anything other than digits into the field (however, it is still possible to paste non-digits into it). An alternate way of forcing a numeric entry is to attach an `UpDown` control to the Edit.

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. 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 still be scrolled, selected and copied to the clipboard.

Tn: 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 dialog units (the width of each "dialog unit" is determined by the operating system). If the letter T is used only 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. Note: Tab stops require a multiline edit control.

Uppercase: The characters typed by the user are automatically converted to uppercase.

WantCtrlA [v1.0.44+]: Specify `-WantCtrlA` (minus `WantCtrlA`) to prevent the user's press of `Control` `A` selecting all text in the edit control.

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` be the same as pressing the window's [default button](#) (if any). In this case, the user may press `Control` `Enter` to start a new line.

WantTab: Causes a `Tab` keystroke to produce a tab character rather than navigating to the next control. Without this option, the user may press `Control` `Tab` to produce a tab character inside a multi-line edit control. Note: Although `WantTab` also works in a single-line edit control, each tab character is displayed as an empty-box character (though it is stored as a real tab).

-Wrap (minus wrap): Turns off word-wrapping in a multi-line edit control. Since this style cannot be changed after the control has been created, use one of the following to change it: 1) [Destroy](#) then recreate the window and its control; or 2) Create two overlapping edit controls, one with wrapping enabled and the other without it. The one not currently in use can be kept empty and/or hidden.

See [general options](#) for other options like *Right*, *Center*, and *Hidden*. See also: [position and sizing of controls](#).

A more powerful edit control: HiEdit is a free, multitabbed, large-file edit control consuming very little memory. It can edit both text and binary files. For details and a demonstration, see www.autohotkey.com/forum/topic19141.html

UpDown

Description: A pair of arrow buttons that the user can click to increase or decrease a value. By default, an UpDown control automatically snaps onto the previously added control. This previous control is known as the UpDown's *buddy control*. The most common example is a "spinner", which is an UpDown attached to an [Edit control](#). For example:

```
Gui, Add, Edit
Gui, Add, UpDown, vMyUpDown Range1-10, 5
```

In the example above, the Edit control is the UpDown's buddy control. Whenever the user presses one of the arrow buttons, the number in the Edit control is automatically increased or decreased.

An UpDown's buddy control can also be a [Text control](#) or [ListBox](#). However, due to OS limitations, controls other than these (such as [ComboBox](#) and [DropDownList](#)) might not work properly with [g-labels](#) and other features.

Specify the UpDown's starting position as the last parameter (if omitted, it 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 UpDown. If the UpDown is attached to an Edit control and you do not wish to validate the user's input, it is best to use the UpDown's value rather than the Edit's. This is because the UpDown will always yield an in-range number, even when the user has typed something non-numeric or out-of-range in the Edit control. On a related note, numbers with more than three digits get a [thousands separator](#) (such as comma) by default. These separators are stored in the Edit's output variable but not that of the UpDown.

If the UpDown has a [g-label](#), it will be launched whenever the user clicks one of the arrow buttons or presses an arrow key on the keyboard. Each launch of the g-label also stores the UpDown's position in its [associated output variable](#) (if any).

UpDown Options

Horz: Make's the control's buttons point left/right rather than up/down. By default, *Horz* also makes the control isolated (no buddy). This can be overridden by specifying `Horz 16` in the control's options.

Left: Puts the UpDown on the left side of its buddy rather than the right.

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. The minimum and maximum may be swapped to cause the arrows to move in the opposite of their normal direction. The broadest allowable range is -2147483648-2147483647. Finally, if the buddy control is a [ListBox](#), the range defaults to 32767-0 for verticals and the inverse for horizontals ([Horz](#)).

Wrap: Causes the control to wrap around to the other end of its range when the user attempts to go beyond the minimum or maximum. Without *Wrap*, the control stops when the minimum or maximum is reached.

-16 (minus 16): Causes a vertical UpDown to be isolated; that is, it will have no buddy. This also causes the control to obey any specified width, height, and position rather than conforming to the size of its buddy control. In addition, an isolated UpDown tracks its own position internally. This position can be retrieved normally by means such as [Gui Submit](#).

0x80: Include `0x80` in *Options* to omit the thousands separator that is normally present between every three decimal digits in the buddy control. However, this style is normally not used because the separators are omitted from the number whenever the script retrieves it from the UpDown control itself (rather than its buddy control).

Increments other than 1: In [this script](#), `NumEric` demonstrates how to change an UpDown's increment to a value other than 1 (such as 5 or 0.1).

See also: [position and sizing of controls](#).

Picture (or Pic)

Description: An area containing an image (see last two paragraphs for supported file types). The last parameter is the filename of the image, which is assumed to be in [A_WorkingDir](#) if an absolute path isn't specified. Example:

```
Gui, Add, Picture, w300 h-1, C:\My  
Pictures\Company Logo.gif
```

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 (this width and height also determines which icon to load from a multi-icon *.ICO* file). 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 width and height are set to zero.

A *g-label* such as `gMySubroutine` may be listed in the control's options. This would cause the *MySubroutine* label to be launched automatically whenever the user clicks the picture. A double-click can be detected by checking whether [A_GuiEvent](#) contains the word `DoubleClick`.

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 `0x40000000` (which is `WS_CLIPSIBLINGS`) in the picture's *Options*. This trick also allows a picture to be the background behind a [Tab control](#) or [ListView](#).

Icons, cursors, and animated cursors: Icons and cursors may be loaded from the following types of files: *ICO*, *CUR*, *ANI*, *EXE*, *DLL*, *CPL*, *SCR*, and other types that contain icon resources. To use an icon group other than the first one in the file, include in *Options* the word `Icon` followed by the number of the group. In the following example, the default icon from the second icon group would be used: `Gui, Add, Picture, Icon2, C:\My Application.exe`.

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 GIF, BMP, and icon images. For example, it would load an ICO/GIF that has a transparent background as a transparent bitmap, which allows the [BackgroundTrans](#) option to take effect (but in [v1.1.23+], icons support transparency without `AltSubmit`). 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, BMP, ICO, CUR, and ANI images. On Windows XP or later, additional image formats such as PNG, TIF, 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`

Animated GIFs: Although animated GIF files can be displayed in a picture control, they will not actually be animated. To solve this, use the `AniGIF` DLL (which is free for non-commercial use) as demonstrated at www.autohotkey.com/forum/topic19264.html

[v1.1.23+]: A [bitmap or icon handle](#) can be used instead of a filename. For example, `HBITMAP:%handle%`.

Button

Description: A pushbutton, which can be pressed to trigger an action. In this case, the last parameter is the name of the button (shown on the button itself), which may include linefeeds (`\n`) to start new lines. Example:

```
Gui, Add, Button, Default, OK
```

The example above includes the word **Default** in its *Options* to make "OK" the default button. The default button's action is automatically triggered whenever the user presses  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 Cancel button become the default: `GuiControl, +Default, Cancel`. To later change the window to have no default button, follow this example: `GuiControl, -default, OK`.

An ampersand (&) may be used in the name button to underline one of its letters. For example:

```
Gui, Add, Button,, &Pause
```

In the example above, the letter P will be underlined, which allows the user to press   [shortcut key](#). To display a literal ampersand, specify two consecutive ampersands (&&).

If a button lacks an explicit [g-label](#), an automatic label is assumed. For example, if the first GUI window contains an OK button, the ButtonOK label (if it exists) will be launched when the button is pressed. For GUI windows [other than the first](#), the window number is included in front of the button's automatic label; for example: `2ButtonOK`.

If the text on the button contains spaces or any of the characters in the set `&`r`n`t``, its automatic label omits those characters. For example, a button titled "&Pause" would have an automatic label of ButtonPause. Similarly, a button

titled "Save && Exit" would have an automatic label of ButtonSaveExit (the double-ampersand is used to display a single, literal ampersand).

Known limitation: Certain desktop themes might not display a button's text properly. If this occurs, try including `-wrap` (minus Wrap) in the button's options. However, this also prevents having more than one line of text.

Checkbox

Description: A small box that can be checked or unchecked to represent On/Off, Yes/No, etc. Example:

```
Gui, Add, Checkbox, vShipToBillingAddress, Ship  
to billing address?
```

The last parameter is a label displayed next to the box, which is 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), the control's text will be word-wrapped as needed, and the control's height will be set automatically. The checkbox's [associated output variable](#) (if any) receives the number 1 for checked, 0 for unchecked, and -1 for gray/indeterminate.

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 word **Checked** may optionally be followed immediately by a 0, 1, or -1 to indicate the starting state. In other words, `Checked` and `Checked%VarContainingOne%` are the same.

A [g-label](#) such as `gMySubroutine` may be listed in the control's options. This would cause the *MySubroutine* label to be launched automatically whenever the user clicks or changes the checkbox.

Known limitation: Certain desktop themes might not display a button's text properly. If this occurs, try including `-Wrap` (minus `Wrap`) in the button's options. However, this also prevents having more than one line of text.

Radio

A Radio button is a small empty circle that can be checked (on) or unchecked (off). Example:

```
Gui, Add, Radio, vMyRadioGroup, Wait for all  
items to be in stock before shipping.
```

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 in between, since that automatically starts a new group.

For the last 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), the control's text will be word-wrapped as needed, and the control's height will be set automatically.

Specify the word **Checked** in *Options* to have the button start off in the "on" state. The word **Checked** may optionally be followed immediately by a 0 or 1 to indicate the starting state: 0 for unchecked and 1 for checked. In other words, **Checked** and **Checked%VarContainingOne%** are the same.

The radio button's [associated output variable](#) (if any) receives the number 1 for "on" and 0 for "off". However, 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.

A [g-label](#) such as **gMySubroutine** may be listed in the control's options. This would cause the *MySubroutine* label to be launched automatically whenever the user turns on the button. Unlike the single-variable mode in the previous

paragraph, the g-label must be specified for each button in a radio group for which the label should be launched. This allows the flexibility to ignore the clicks of certain buttons. Finally, a double-click can be detected by checking whether [A_GuiEvent](#) contains the word DoubleClick.

Known limitation: Certain desktop themes might not display a button's text properly. If this occurs, try including `-wrap` (minus Wrap) in the button's options. However, this also prevents having more than one line of text.

DropDownList (or DDL)

Description: A list of choices that is displayed in response to pressing a small button. In this case, the last parameter is a pipe-delimited list of choices such as `Choice1|Choice2|Choice3`. Example:

```
Gui, Add, DropDownList, vColorChoice,  
Black|White|Red|Green|Blue
```

To have one of the items pre-selected when the window first appears, include two pipe characters after it (e.g. `Red|Green||Blue`). Alternatively, include in *Options* the word **Choose** followed immediately by the number to pre-select. For example, `Choose5` would pre-select the fifth item (as with other options, it can also be a variable such as `Choose%Var%`). 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 uppercase or lowercase. Specify the word **Sort** to automatically sort the contents of the list alphabetically (this also affects any items added later via [GuiControl](#)). The Sort option also enables incremental searching whenever the list is dropped down; this allows an item to be selected by typing the first few characters of its name.

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.).

A [g-label](#) such as `gMySubroutine` may be listed in the control's options. This would cause the *MySubroutine* label to be launched automatically whenever the user selects a new item.

Use the [R or H option](#) to control the height of the popup list. For example, specifying `R5` would make the list 5 rows tall, while `H400` would set the total height of the selection field and list to 400 pixels. If both R and H are omitted,

the list will automatically expand to take advantage of the available height of the user's desktop (however, operating systems older than Windows XP will show 3 rows by default).

To set the height of the selection field or list items, use the `CB_SETITEMHEIGHT` message as in the example below:

```
Gui Add, DDL, vcbx w200 hwndhcbx, One||Two
; CB_SETITEMHEIGHT = 0x153
PostMessage, 0x153, -1, 50,, ahk_id %hcbx% ;
Set height of selection field.
PostMessage, 0x153, 0, 50,, ahk_id %hcbx% ;
Set height of list items.
Gui Show, h70, Test
```

The separator between fields may be changed to something other than pipe (`|`). For example `Gui +Delimiter`n` would change it to linefeed and `Gui +DelimiterTab` would change it to tab (``t`).

ComboBox

Description: Same as DropDownList but also permits free-form text to be entered as an alternative to picking an item from the list. Example:

```
Gui, Add, ComboBox, vColorChoice,  
Red|Green|Blue|Black|White
```

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.

A [g-label](#) such as `gMySubroutine` may be listed in the control's options. This would cause the *MySubroutine* label to be launched automatically whenever the user selects a new item.

ListBox

Description: A relatively tall box containing a list of choices that can be selected. In this case, the last parameter is a pipe-delimited list of choices such as `Choice1|Choice2|Choice3`. Example:

```
Gui, Add, ListBox, vColorChoice,  
Red|Green|Blue|Black|White
```

To have list item(s) pre-selected when the window first appears, include two pipe characters after each (the [Multi](#) option 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](#).

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.).

A [g-label](#) such as `gMySubroutine` may be listed in the control's options. This would cause the *MySubroutine* label to be launched automatically whenever the user selects a new item. If the user double-clicks an item, the built-in variable `A_GuiEvent` will contain the string `DoubleClick` rather than `Normal`. Also, the variable `A_EventInfo` will contain the position of the item that was double-clicked (1 is the first item, 2 is the second, etc.).

When adding a large number of items to a `ListBox`, performance may be improved by using `GuiControl, -Redraw, MyListBox` prior to the operation, and `GuiControl, +Redraw, MyListBox` afterward.

ListBox Options

Choose: See [above](#).

Multi: Allows 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%.
}
```

The separator between fields may be changed to something other than pipe (`|`). For example `Gui +Delimiter`n` would change it to linefeed and `Gui +DelimiterTab` would change it to tab (``t`).

ReadOnly: Prevents items from being visibly highlighted when they are selected (but [Gui Submit](#) will still store the selected item).

Sort: Automatically sorts the contents of the list alphabetically (this also affects any items added later via [GuiControl](#)). The Sort option also enables incremental searching, which allows an item to be selected by typing the first few characters of its name.

Tn: The letter T may be used to set tab stops, which can be used to format the text into columns. If the letter T is not used, tab stops are set at every 32 dialog units (the width of each "dialog unit" is determined by the operating system). If the letter T is used only once, tab stops are set at every `n` units across the entire width of the control. For example, `Gui, Add, ListBox, vMyListBox 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, ListBox, vMyListBox 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.

0x100: Include 0x100 in options to turn on the LBS_NOINTEGRALHEIGHT style. This forces the ListBox to be exactly the height specified rather than a height that prevents a partial row from appearing at the bottom. This option also prevents the ListBox from shrinking when its font is changed.

To specify the number of rows of text (or the height and width), see [position and sizing of controls](#).

ListView and TreeView

See separate pages [ListView](#) and [TreeView](#).

Link [v1.1.06+]

Description: A text control that can contain links that can be clicked. Link controls use HTML-like markup language, but they only support the <A> tag. Inside the opening tag, an attribute of the form `href="value"` or `id="value"` may be specified. For example:

```
Gui, Add, Link,, This is a <a
href="https://autohotkey.com">link</a>
Gui, Add, Link,, Links may be used anywhere in
the text like <a id="A">this</a> or <a
id="B">that</a>
```

If the HREF attribute is set and contains a valid executable command or URL, it is executed as if passed to the [Run](#) command. However, an executable command cannot contain double-quote marks. A URL can usually contain percent-encoded characters such as `%22`, but these are interpreted by the user's web browser, not the Link control.

If the control has a [g-label](#) such as `gMySubroutine`, the subroutine is launched whenever a link without a HREF attribute is clicked, or if the command or URL failed to execute. This subroutine may consult the following variables:

- `A_GuiEvent` contains the type of event; currently always the word "Normal".
- `A_EventInfo` contains the 1-based index of the link.
- `ErrorLevel` contains the value of the link's HREF attribute, if any; otherwise, the value of the link's ID attribute or an empty string.

If the g-label is a function, it can accept the following parameters:

```
MyFunction(CtrlHwnd, GuiEvent, LinkIndex, HrefOrID)
```

Hotkey

Description: 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".

```
Gui, Add, Hotkey, vChosenHotkey
```

When the [Gui Submit](#) command is used, the control's [associated output variable](#) (if any) receives the hotkey modifiers and name, which are 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 last parameter as in this example:
`Gui, Add, Hotkey, vChosenHotkey, ^!p`

The only modifiers supported are ^ (Control), ! (Alt), and + (Shift). See the [key list](#) for available key names.

A [g-label](#) such as `gMySubroutine` may be listed in the control's options. This would cause the *MySubroutine* label to be launched automatically whenever the user changes the hotkey. Each launch of the g-label also stores the hotkey in control's [associated output variable](#) (if any).

Note: The g-label is launched even when an incomplete hotkey is present. For example, if the user holds down the `Control` key, the g-label is launched once and the output variable contains only a circumflex (^). When the user completes the hotkey, the label is launched again and the variable contains the complete hotkey.

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

4: Prevent Control

8: Prevent Alt keys

16: Prevent Shift Control

32: Prevent Shift Alt

64: This value is not supported (it will not behave correctly).

128: Prevent Shift Control Alt

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 automatically and visibly substituted.

The Hotkey control has limited capabilities. For example, it does not support mouse/joystick hotkeys or the Win (Win 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 Win

DateTime

Description: A box that looks like a single-line edit control but instead accepts a date and/or time. A drop-down calendar is also provided. Example:

```
Gui, Add, DateTime, vMyDateTime, LongDate
```

The last parameter may be one of the following:

(omitted): When omitted, the locale's short date format is used. For example, in some locales it would look like: 6/1/2005

LongDate: Uses the locale's long date format. For example, in some locales it would look like: Wednesday, June 01, 2005

Time: Shows only the time using the locale's time format. Although the date is not shown, it is still present in the control and will be retrieved along with the time in the `YYYYMMDDHH24MISS` format.

(custom format): Specify any combination of [date and time formats](#). For example, `M/d/yy HH:mm` would look like 6/1/05 21:37. Similarly, `dddd MMMM d, yyyy hh:mm:ss tt` would look like Wednesday June 1, 2005 09:37:45 PM. Letters and numbers to be displayed literally 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, 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.

DateTime Usage

To have a date other than today pre-selected, include in *Options* the word **Choose** followed immediately by a date in `YYYYMMDD` format. For example, `Choose20050531` would pre-select May 31, 2005 (as with other options, it can also be a variable such as `Choose%Var%`). To have no date/time selected,

specify **ChooseNone**. *ChooseNone* also creates a checkbox inside the control that is unchecked whenever the control has no date. Whenever the control has no date, [Gui Submit](#) and [GuiControlGet](#) will retrieve a blank value (empty string).

The time of day may optionally be present. However, it must always be preceded by a date when going into or coming out of the control. The format of the time portion is HH24MISS (hours, minutes, seconds), where HH24 is expressed in 24-hour format; for example, 09 is 9am and 21 is 9pm. Thus, a complete date-time string would have the format [YYYYMMDDHH24MISS](#).

When specifying dates in the YYYYMMDDHH24MISS format, only the leading part needs to be present. 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

Within the drop-down calendar, the today-string at the bottom can be clicked to select today's date. In addition, the year and month name are clickable and allow easy navigation to a new month or year.

Keyboard navigation: Use the     keys, the    keys, and   to increase or decrease the control's values. Use   to move from field to field inside the control. Within the drop-

down calendar, use the arrow keys to move from day to day; use  

  to move backward/backward by one month; use  

   to move backward/backward by one year; and use

  to select the first/last day of the month.

When the [Gui Submit](#) command is used, the control's [associated output variable](#) (if any) receives the selected date and time in [YYYYMMDDHH24MISS](#) format. Both the date and the time are present regardless of whether they were actually visible in the control.

If the control has a [g-label](#), the label is launched whenever the user changes the

date or time. For each launch, the control's [associated output variable](#) (if any) is automatically updated with the currently selected date/time.

DateTime Options

Choose: See [above](#).

Range: Restricts how far back or forward in time the selected date can be. After the word Range, specify the minimum and maximum dates in YYYYMMDD format (with a dash between them). For example, `Range20050101-20050615` would restrict the date to the first 5.5 months of 2005. Either the minimum or maximum may be omitted to leave the control unrestricted in that direction. For example, `Range20010101` would prevent a date prior to 2001 from being selected and `Range-20091231` (leading dash) would prevent a date later than 2009 from being selected. Without the Range option, any date between the years 1601 and 9999 can be selected. The time of day cannot be restricted.

Right: Causes the drop-down calendar to drop down on the right side of the control instead of the left.

1: Specify the number 1 in *Options* to provide an up-down control to the right of the control to modify date-time values, which replaces the button of the drop-down month calendar that would otherwise be available. This does not work in conjunction with the format option LongDate described above.

2: Specify the number 2 in *Options* to provide a checkbox inside the control that the user may uncheck to indicate that no date/time is selected. Once the control is created, this option cannot be changed.

MonthCal

Description: A tall and wide control that displays all the days of the month in calendar format. The user may select a single date or a range of dates. Example:

```
Gui, Add, MonthCal, vMyCalendar
```

To have a date other than today pre-selected, specify it as the last parameter in YYYYMMDD format (e.g. `20050531`). A range of dates may also be pre-selected by including a dash between two dates (e.g. `20050525-20050531`).

It is usually best to omit width (W) and height (H) for a MonthCal because it automatically sizes itself to fit exactly one month. To display more than one month vertically, specify `R2` or higher in *Options*. To display more than one month horizontally, specify `W-2` (W negative two) or higher. These options may both be present to expand in both directions.

The today-string at the bottom of the control can be clicked to select today's date. In addition, the year and month name are clickable and allow easy selection of a new year or month.

Unlike [DateTime](#)'s drop-down calendar, keyboard navigation is generally not supported in a MonthCal.

When the [Gui Submit](#) command is used, the control's [associated output variable](#) (if any) receives the selected date in YYYYMMDD format (without any time portion). However, when the [multi-select](#) option is in effect, the minimum and maximum dates are retrieved with a dash between them (e.g. `20050101-20050108`). If only a single date was selected in a multi-select calendar, the minimum and maximum are both present but identical. [StringSplit](#) can be used to separate the dates. For example, the following would put the minimum in Date1 and the maximum in Date2: `StringSplit, Date, MyMonthCal, -`.

If the MonthCal has a [g-label](#), each launch of it updates the control's [associated output variable](#) (if any) with the currently selected date or range. By default, the

label is launched only when: 1) the user changes the selection; or 2) every two minutes in case a new day has arrived (this behavior is a quirk of the OS). However, if the word `AltSubmit` is in the control's *Options*, the `g-label` is launched more often and the built-in variable `A_GuiEvent` will contain the word `Normal` for a change of the date, the number 1 for a click of a date, and the number 2 when the `MonthCal` releases "mouse capture". For example, if the user double-clicks a new date, the label would be launched five times: Once with `Normal`, twice with 1, and twice with 2. This can be used to detect double clicks by [measuring the time](#) between instances of the number 1.

When specifying dates in the `YYYYMMDD` format, the `MM` and/or `DD` portions may be omitted, in which case they are assumed to be 1. For example, `200205` is seen as `20020501`, and `2005` is seen as `20050101`.

MonthCal Options

Multi: Multi-select. Allows the user to shift-click or click-drag to select a range of adjacent dates (the user may still select a single date too). This option may be specified explicitly or put into effect automatically by means of specifying a selection range when the control is created. For example: `Gui, Add, MonthCal, vMyCal, 20050101-20050108`. Once the control is created, this option cannot be changed.

Range: Restricts how far back or forward in time the calendar can go. After the word `Range`, specify the minimum and maximum dates in `YYYYMMDD` format (with a dash between them). For example, `Range20050101-20050615` would restrict the selection to the first 5.5 months of 2005. Either the minimum or maximum may be omitted to leave the calendar unrestricted in that direction. For example, `Range20010101` would prevent a date prior to 2001 from being selected and `Range-20091231` (leading dash) would prevent a date later than 2009 from being selected. Without the `Range` option, any date between the years 1601 and 9999 can be selected.

4: Specify the number 4 in *Options* to display week numbers (1-52) to the left of each row of days. Week 1 is defined as the first week that contains at least four days.

8: Specify the number 8 in *Options* to prevent the circling of today's date within the control.

16: Specify the number 16 in *Options* to prevent the display of today's date at the bottom of the control.

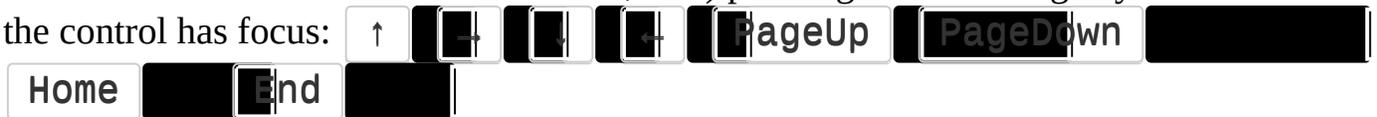
Slider

Description: 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.
Example:

```
Gui, Add, Slider, vMySlider, 50
```

Specify the starting position of the slider as the last parameter. If the last parameter is omitted, the slider starts off at 0 or the number in the allowable range that is closest to 0.

The user may slide the control by the following means: 1) dragging the bar with the mouse; 2) clicking inside the bar's track area with the mouse; 3) turning the mouse wheel while the control has focus; or 4) pressing the following keys while the control has focus:

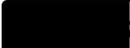


When the [Gui Submit](#) command is used, the control's [associated output variable](#) (if any) receives the current numeric position of the slider. The position is also stored in the output variable whenever the control's [g-label](#) is launched.

If the slider has 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 the word `AltSubmit` is in the control's *Options*, the [g-label](#) is launched for all slider events and the built-in variable `A_GuiEvent` will contain one of the following digits or strings:

- 0: The user pressed the  .
- 1: The user pressed the  .
- 2: The user pressed the .
- 3: The user pressed the .
- 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`  send the slider to the left or top side.

7: The user pressed the `End`  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

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.

Center: The thumb (the bar moved by the user) will be blunt on both ends rather than pointed at one end.

Invert: 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.

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.

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. For example: `Line2` .

NoTicks: Omits tickmarks alongside the track.

Page: Specifies the number of positions to move when the user presses the `PageUp` or `PageDown` word Page, specify number of positions to move. For example: `Page10`.

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.

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 `-Theme` in the control's options).

TickInterval: Provides tickmarks alongside the track 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 1). 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 vertical sliders); `ToolTipTop` or `ToolTipBottom` (for horizontal sliders).

Vertical: Makes the control slide up and down rather than left and right.

The above options can be changed after the control is created via [GuiControl](#).

Progress

Description: A dual-color bar typically used to indicate how much progress has been made toward the completion of an operation. Example:

```
Gui, Add, Progress, w300 h20 cBlue vMyProgress
```

Specify the starting position of the bar as the last parameter (if 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 control's height. For vertical Progress Bars it is equal to the control's width.

Progress Options

Cn: 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, `Range0-`

`1000` would allow a numbers between 0 and 1000; `Range -50 -50` would allow numbers between -50 and 50; and `Range -10 - -5` would allow numbers between -10 and -5.

-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.

Vertical: Makes the bar rise or fall vertically rather than move along horizontally.

The above options can be changed after the control is created via [GuiControl](#).

GroupBox

Description: A rectangular border/frame, often used around other controls to indicate they are related. In this case, the last parameter is the title of the box, which if present is displayed at its upper-left edge. Example:

```
Gui, Add, GroupBox, w400 h300, Geographic  
Criteria
```

By default, a GroupBox's title may have only one line of text. This can be overridden by specifying `Wrap` in Options.

To specify the number of rows inside the control (or its height and width), see [position and sizing of controls](#).

Tab3

Description: A large control containing multiple pages, each of which contains other controls. From this point forward, these pages are referred to as "tabs".

There are three types of Tab control:

- **Tab3** [v1.1.23.00+]: Fixes some issues which affect Tab2 and Tab. Controls are placed within an invisible "tab dialog" which moves and resizes with the tab control. The tab control is themed by default.
- **Tab2** [v1.0.47.05+]: Fixes rare redrawing problems in the original "Tab" control but introduces [some other problems](#).
- **Tab**: Retained for backward compatibility because of [differences in behavior](#) between Tab2/Tab3 and Tab.

Example:

```
Gui, Add, Tab3, ,  
General|View|Appearance|Settings
```

The last parameter above is a pipe-delimited list of tab names. To have one of the tabs pre-selected when the window first appears, include two pipe characters after it (e.g. `Red|Green||Blue`). Alternatively, include in *Options* the word **Choose** followed immediately by the number to pre-select. For example, `Choose5` would pre-select the fifth tab (as with other options, it can also be a variable such as `Choose%Var%`). 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, 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.  
Gui, Tab, Name ; Future controls are owned by  
the tab whose name starts with Name (not case  
sensitive).  
Gui, Tab, Name,, Exact ; Same as above but  
requires exact match (case sensitive too).
```

It is also possible to use any of the examples above to assign controls to a tab or tab-control that does not 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 first sub-control is positioned at the upper-left corner of the tab control's interior (with a standard [margin](#)), and sub-controls beyond the first are positioned beneath the previous control; 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.

V: 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.).

G: A [g-label](#) such as `gMySubroutine` may be listed in the control's options. This would cause the *MySubroutine* label to be launched automatically whenever the user changes to a new tab. If the tab control has both a [g-label](#) and an [output variable](#), whenever the user switches to a new tab, the output variable will be set to the previously selected tab name (or number in the case of [AltSubmit](#)).

Keyboard navigation: The user may press `Control` `PageDown` `PageUp` 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` `Control` `Shift` `Tab` so be used except that they will not work if the currently focused

control is a multi-line Edit control.

Limits: Each window may have no more than 255 tab controls. Each tab control may have no more than 256 tabs (pages). In addition, a tab control may not contain other tab controls.

Tab3 vs. Tab2 vs. Tab

Parent window: The parent window of a control affects the positioning and visibility of the control and tab-key navigation order. If a sub-control is added to an existing Tab3 control, its parent window is the "tab dialog", which fills the tab control's display area. Most other controls, including sub-controls of Tab or Tab2 controls, have no parent other than the GUI window itself.

Positioning: For Tab and Tab2, 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 appropriate for the "buttons" style described below.

For Tab3, sub-controls assigned to a tab *before* the tab control is created behave as though added to a Tab or Tab2 control. All other sub-controls are visible only within the display area of the tab control.

If a Tab3 control is moved, its sub-controls are moved with it. Tab and Tab2 controls do not have this behavior.

In the rare case that [WinMove](#) (or an equivalent DllCall) is used to move a control, the coordinates must be relative to the parent window of the control, which might not be the GUI (see [above](#)). By contrast, [GuiControl Move](#) takes GUI coordinates and [ControlMove](#) takes window coordinates, regardless of the control's parent window.

Autosizing: If not specified by the script, the width and/or height of the Tab3 control are automatically calculated at one of the following times (whichever comes first after the control is created):

- The first time the Tab3 control ceases to be the current tab control. This can occur as a result of calling [Gui, Tab](#) (with or without parameters) or creating another tab control.

- The first time [Gui, Show](#) is called for that particular Gui.

The calculated size accounts for sub-controls which exist when autosizing occurs, plus the default margins. The size is calculated only once, and will not be recalculated even if controls are added later. If the Tab3 control is empty, it receives the same default size as a Tab or Tab2 control.

Tab and Tab2 controls are not autosized; they receive an arbitrary default size.

Tab-key navigation order: The navigation order via the  usually depends on the order in which the controls are created. When tab controls are used, the order also depends on the type of tab control:

- Tab and Tab2 allow their sub-controls to be mixed with other controls within the tab-key order.
- Tab2 puts its tab buttons after its sub-controls in the tab-key order.
- Tab3 groups its sub-controls within the tab-key order and puts them after its tab buttons.

Notification messages (Tab3): Common and [Custom](#) controls typically send notification messages to their [parent window](#). Any WM_COMMAND, WM_NOTIFY, WM_VSCROLL, WM_HSCROLL or WM_CTLCOLOR' messages received by a Tab3 control's [tab dialog](#) are forwarded to the GUI window and can be detected by using [OnMessage](#). If the tab control is themed and the sub-control lacks the [+BackgroundTrans](#) option, WM_CTLCOLORSTATIC is fully handled by the tab dialog and not forwarded. Other notification messages (such as custom messages) are not supported.

Known issues with Tab2:

- [BackgroundTrans](#) has no effect inside a Tab2 control.
- [WebBrowser](#) controls do not redraw correctly.
- [AnimateWindow](#) and possibly other Win32 API calls can cause the tab's controls to disappear.

Known issues with Tab:

- Activating a GUI window by clicking certain parts of its controls, such as scrollbars, might redraw improperly.
- [BackgroundTrans](#) has no effect if the Tab control contains a [ListView](#).

- [WebBrowser](#) controls are invisible.

Tab Options

Choose: See [above](#).

-Background (minus followed by the word background): Overrides the [window's custom background color](#) and uses the system's default Tab control color. Specify `+Theme -Background` to make the Tab control conform to the current desktop theme. However, most control types will look strange inside such a Tab control because their backgrounds will not match that of the tab control. This can be fixed for some control types (such as [Text](#)) by adding `BackgroundTrans` to their options.

Buttons: Creates 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).

Left/Right/Bottom: Specify one of these words to have the tabs on the left, right, or bottom side instead of the top. See [TCS_VERTICAL](#) for limitations on Left and Right.

-Wrap: Prevents 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).

To specify the number of rows of text inside the control (or its height and width), see [position and sizing of controls](#).

Icons in Tabs: An icon may be displayed next to each tab's name/text via [SendMessage](#). This is demonstrated in the forum topic [Icons in tabs](#).

StatusBar [v1.0.44+]

Description: A row of text and/or icons attached to the bottom of a window, which is typically used to report changing conditions. Example:

```
Gui, Add, StatusBar,, Bar's starting text (omit  
to start off empty).  
SB_SetText("There are " . RowCount . " rows  
selected.")
```

The simplest use of a status bar is to call [SB_SetText\(\)](#) whenever something changes that should be reported to the user. To report more than one piece of information, divide the bar into sections via [SB_SetParts\(\)](#). To display icon(s) in the bar, call [SB_SetIcon\(\)](#).

All of the following StatusBar functions operate upon the current thread's [default GUI window](#) (which can be changed via [Gui, 2:Default](#)). If the default window does not exist or has no status bar, all SB functions return 0 to indicate the problem.

SB_SetText(NewText [, PartNumber, Style])

Displays *NewText* in the specified part of the status bar. If *PartNumber* is omitted, it defaults to 1. Otherwise, specify an integer between 1 and 256. If *Style* is omitted, it defaults to 0, which uses a traditional border that makes that part of the bar look sunken. Otherwise, specify 1 to have no border or 2 to have border that makes that part of the bar look raised. Finally, up to two tab characters (``t`) may be present anywhere in *NewText*: anything to the right of the first tab is centered within the part, and anything to the right of the second tab is right-justified. [SB_SetText\(\)](#) returns 1 upon success and 0 upon failure.

SB_SetParts([Width1, Width2, ... Width255])

Divides the bar into multiple sections according to the specified widths (in pixels). If all parameters are omitted, the bar is restored to having only a single, long part. Otherwise, specify the width of each part except the last (the last will

fill the remaining width of the bar). For example, `SB_SetParts(50, 50)` would create three parts: the first two of width 50 and the last one of all the remaining width.

Note: Any parts "deleted" by `SB_SetParts()` will start off with no text the next time they are shown (furthermore, their icons are automatically destroyed).

Upon success, `SB_SetParts()` returns a non-zero value (the status bar's [HWND](#)). Upon failure it returns 0.

`SB_SetIcon(Filename [, IconNumber, PartNumber])`

Displays a small icon to the left of the text in the specified part (if *PartNumber* is omitted, it defaults to 1). *Filename* is the name of an icon (.ICO), cursor (.CUR), or animated cursor (.ANI) file (animated cursors will not actually be animated in the bar). Other sources of icons include the following types of files: EXE, DLL, CPL, SCR, and other types that contain icon resources. To use an icon group other than the first one in the file, specify its number for *IconNumber*. For example, `SB_SetIcon("Shell32.dll", 2)` would use the default icon from the second icon group. If *IconNumber* is negative, its absolute value is assumed to be the resource ID of an icon within an executable file. `SB_SetIcon()` returns the icon's HICON upon success and 0 upon failure. The HICON is a system resource that can be safely ignored by most scripts because it is destroyed automatically when the status bar's window is destroyed. Similarly, any old icon is destroyed when `SB_SetIcon()` replaces it with a new one. This can be avoided via:

```
Gui +LastFound
SendMessage, 0x40F, part_number - 1, my_hIcon,
msctls_statusbar321 ; 0x40F is SB_SETICON.
```

[v1.1.23+]: An [icon handle](#) can be used instead of a filename. For example, `SB_SetIcon("HICON:" handle)`.

[v1.1.27+]: Non-icon image files and [bitmap handles](#) are supported for *Filename*. For example, `SB_SetIcon("HBITMAP:" handle)`.

SB_SetProgress()

Creates and controls a progress bar inside the status bar. This function is available at www.autohotkey.com/forum/topic37754.html

G-Label Notifications

A **g-label** such as `gMySubroutine` may be listed in the control's options. This would cause the *MySubroutine* label to be launched automatically whenever the user clicks on the bar. This subroutine may consult the built-in variables `A_Gui` and `A_GuiControl`. More importantly, it may consult `A_GuiEvent`, which contains one of the following strings (for compatibility with future versions, a script should not assume these are the only possible values):

- **Normal:** The user left-clicked the bar. The variable `A_EventInfo` contains the part number (however, the part number might be a very large integer if the user clicks near the sizing grip at the right side of the bar).
- **RightClick:** The user right-clicked the bar. The variable `A_EventInfo` contains the part number. NOTE: `GuiContextMenu` will not be called for the status bar if it has a g-label. Also, the g-label's `RightClick` event should be used instead of `GuiContextMenu` when the script needs to know which part number the user clicked on (`A_EventInfo`).
- **DoubleClick:** The user double-clicked the bar. The variable `A_EventInfo` contains the part number.
- **R:** The user *double-right*-clicked the bar. The variable `A_EventInfo` contains the part number.

Font and Color

Although the font size, face, and style can be set via `Gui Font` (just like normal controls), the text color cannot be changed. Also, `Gui Color` is not obeyed; instead, the status bar's background color may be changed by specifying in *Options* the word **Background** followed immediately by a color name (see [color chart](#)) or RGB value (the 0x prefix is optional). Examples:

```
BackgroundSilver , BackgroundFFDD99 , BackgroundDefault .
```

Hiding the StatusBar

Upon creation, the bar can be hidden via `Gui, Add, StatusBar, Hidden vMyStatusBar`. To hide it sometime after creation, use `GuiControl, Hide, MyStatusBar`. To show it, use `GuiControl, Show, MyStatusBar`.

Note: Hiding the bar does not reduce the height of the window. If that is desired, one easy way is `Gui, Show, AutoSize`.

Styles (rarely used)

See the [StatusBar styles table](#).

Known Limitations

1) Any control that overlaps the status bar might sometimes get drawn on top of it. One way to avoid this is to dynamically shrink such controls via the [GuiSize label](#). 2) There is a limit of one status bar per window.

Example

The bottom of the [TreeView page](#) demonstrates a multipart status bar.

ActiveX [v1.1.03+]

ActiveX components such as the MSIE browser control can be embedded into a GUI window by following this example:

```
Gui Add, ActiveX, w980 h640 vWB, Shell.Explorer
; The final parameter is the name of the ActiveX
component.
WB.Navigate("https://autohotkey.com/boards/") ;
This is specific to the web browser control.
Gui Show
```

When the control is created, an ActiveX object is stored in the control's associated variable, if it has one. [GuiControlGet](#) can also be used to retrieve the object.

To handle events exposed by the object, use [ComObjConnect](#) as demonstrated below:

```
Gui Add, Edit, w930 r1 vURL,
https://autohotkey.com/boards/
Gui Add, Button, x+6 yp w44 Default, Go
Gui Add, ActiveX, xm w980 h640 vWB,
Shell.Explorer
ComObjConnect(WB, WB_events) ; Connect WB's
events to the WB_events class object.
Gui Show
; Continue on to load the initial page:
ButtonGo:
Gui Submit, NoHide
WB.Navigate(URL)
return

class WB_events
{
```

```
NavigateComplete2(wb, NewURL)
{
    GuiControl,, URL, %NewURL% ; Update the
URL edit control.
}
}

GuiClose:
ExitApp
```

[ComObjType](#) can be used to determine the type of object stored in the control's variable.

Custom [v1.1.10+]

Other controls which are not directly supported by AutoHotkey can be also embedded into a GUI window. In order to do so, the Win32 class name must be specified through the `Class` option in `Gui, Add`. Examples:

```
Gui, Add, Custom, ClassComboBoxEx32 ; Adds a
ComboBoxEx control.
Gui, Add, Custom, ClassScintilla ; Adds a
Scintilla control. Note that the SciLexer.dll
library must be loaded before the control can be
added.
```

AutoHotkey uses the standard Windows control text routines when text is to be retrieved/replaced in the control via `Gui, Add` or `GuiControl/Get`.

G-Label Notifications: A `g-label` such as `gMySubroutine` may be listed in the control's options in order to capture events coming from the control. This subroutine may consult the built-in variables `A_Gui` and `A_GuiControl`. More importantly, it may consult `A_GuiEvent`, which contains one of the following strings (for compatibility with future versions, a script should not assume these are the only possible values):

- **Normal:** A `WM_COMMAND` message was received. `A_EventInfo` contains a control-defined notification code (equivalent to `HIWORD(wParam)` in C/C++).
- **N:** A `WM_NOTIFY` message was received. `A_EventInfo` contains a pointer to the control notification structure (`NMHDR`). A 32-bit signed integer value may be returned to the control by assigning it to `ErrorLevel` (by default, the `g-label` thread's `ErrorLevel` is set to zero). Invalid values are treated as zero.

Here is an example that shows how to add and use an `IP address control`:

```
Gui, Add, Custom, ClassSysIPAddress32 r1 w150
```

```

hwndhIPControl gIPControlEvent
Gui, Add, Button, Default, OK
IPCtrlSetAddress(hwndhIPControl, A_IPAddress1)
Gui, Show
return

GuiClose:
ExitApp

ButtonOK:
Gui, Hide
ToolTip
MsgBox % "You chose "
IPCtrlGetAddress(hwndhIPControl)
ExitApp

IPControlEvent:
if A_GuiEvent = Normal
{
    ; WM_COMMAND was received.

    if (A_EventInfo = 0x0300) ; EN_CHANGE
        ToolTip Control changed!
}
else if A_GuiEvent = N
{
    ; WM_NOTIFY was received.

    ; Get the notification code. Normally this
    field is UInt but the IP address
    ; control uses negative codes, so for
    convenience we read it as a signed int.
    nmhdr_code := NumGet(A_EventInfo +
2*A_PtrSize, "int")
    if (nmhdr_code != -860) ; IPN_FIELDCHANGED
        return
}

```

```

    ; Extract info from the NMIPADDRESS
structure
    iField := NumGet(A_EventInfo + 3*A_PtrSize +
0, "int")
    iValue := NumGet(A_EventInfo + 3*A_PtrSize +
4, "int")
    if iValue >= 0
        ToolTip Field #%iField% modified:
%iValue%
    else
        ToolTip Field #%iField% left empty
}
return

```

```

IPCtrlSetAddress(hControl, IPAddress)
{
    static WM_USER := 0x400
    static IPM_SETADDRESS := WM_USER + 101

    ; Pack the IP address into a 32-bit word for
use with SendMessage.
    IPAddrWord := 0
    Loop, Parse, IPAddress, .
        IPAddrWord := (IPAddrWord * 256) +
A_LoopField
    SendMessage IPM_SETADDRESS, 0, IPAddrWord,,
ahk_id %hControl%
}

```

```

IPCtrlGetAddress(hControl)
{
    static WM_USER := 0x400
    static IPM_GETADDRESS := WM_USER + 102

    VarSetCapacity(AddrWord, 4)
    SendMessage IPM_GETADDRESS, 0, &AddrWord,,
ahk_id %hControl%
}

```

```
    return NumGet(AddrWord, 3, "UChar") "."  
    NumGet(AddrWord, 2, "UChar") "."  
    NumGet(AddrWord, 1, "UChar") "."  
    NumGet(AddrWord, 0, "UChar")  
}
```

Related Pages

[ListView](#), [TreeView](#), [Gui](#), [GuiControl](#), [GuiControlGet](#), [Menu](#)

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 *ControlID* (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, *ControlID* can be either `ClassNN` (the classname and instance number of the control) or the control's text, both of which can be determined via `Window Spy`. When using 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*.

[v1.1.04+]: *ControlID* can be the [HWND](#) of a control.

If the control is not on the default GUI, **the name or HWND of the GUI must also be specified** -- except on [v1.1.20+] when *ControlID* is a `HWND`, since each `HWND` is unique. See [Remarks](#) for details.

Param3

This parameter is omitted except where noted in the list of sub-commands below.

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[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*. Specifically:

Picture: *Param3* should be the filename (or [handle](#)) of the new image to load (see [Gui Picture](#) for supported file types). Zero or more of the following options may be specified immediately in front of the filename: `*wN` (width N), `*hN` (height N), and `*IconN` (icon group number N in a DLL or EXE file). In the following example, the default icon from the second icon group is loaded with a width of 100 and an automatic height via "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. When loading from a multi-icon .ICO file, specifying a width and height also determines which icon to load.

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.

Text/Button/GroupBox/StatusBar/Link: Specify for *Param3* the control's new text. Since the control will not expand automatically, use `GuiControl, Move, MyText, w300` if the control needs to be widened. For [StatusBar](#), this sets the text of the first part only (use `SB_SetText()` for greater flexibility).

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 not a concern 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: `^!c`, `^Numpad1`, `+Home`. The only modifiers supported are `^` (`Control` `Alt` `Shift`) (`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. To check a new button within a radio group that only has one variable, specify for *ControlID* the name/text of the button if it is not the button with which the variable is directly associated.

DateTime/MonthCal: Specify for *Param3* a date-time stamp in `YYYYMMDDHH24MISS` format. Specify `%A_NOW%` to use the current date and time (today). For DateTime controls, *Param3* may be omitted to cause the control to have no date/time selected (if it was created with [that ability](#)). For MonthCal controls, a range may be specified if the control is [multi-select](#).

UpDown/Slider/Progress: *Param3* should be the new position of the control. 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. If the new position would be outside the range of the control, the control is generally set to the nearest valid value.

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 make the control empty, specify only a pipe character (`|`). To have one of the entries pre-selected, include two pipes after it (e.g. `Red|Green||Blue`). The separator between fields may be changed to something other than pipe. For example, `Gui +Delimiter`n` would change it to linefeed and `Gui +DelimiterTab` would change it to tab (`^t`).

Tab controls: In addition to the behavior described in the paragraph above, 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 removing 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 removed by means of `GuiControl, , MyTab, |Red|Blue`, the sub-controls originally associated with Green will now be associated with Blue. Because of this behavior, only tabs at the end should generally be removed. Tabs that are removed in this way can be added back later, at which time they will reclaim their original set of controls.

ListView and **TreeView**: These are not supported when *Sub-command* is blank. Instead, use the built-in [ListView functions](#) and [TreeView functions](#).

GuiControl, Text: Behaves the same as the above except for:

Checkbox/Radio: *Param3* is treated as the new text/caption even if it is -1, 0, or 1.

DateTime: *Param3* is treated as the new [date/time format](#) displayed by the control. If *Param3* is omitted, any custom format is removed and the short-date format is put into effect.

ComboBox: *Param3* is treated as the text to put directly into the ComboBox's Edit control.

GuiControl, 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). (Specify each number as decimal, not hexadecimal.) For example:

```
GuiControl, Move, MyEdit, x10 y20 w200 h100
GuiControl, Move, MyEdit, % "x" VarX+10 "y"
VarY+5 "w" VarW*2 "h" VarH*1.5 ; Uses an
expression via "% " prefix.
```

GuiControl, MoveDraw: Same as "Move" (above) except that it also repaints the region of the GUI window occupied by the control. Although this may cause an unwanted flickering effect when called repeatedly and rapidly, it solves painting artifacts for certain control types such as [GroupBoxes](#). [v1.0.48.04+]: The last parameter may be omitted to redraw the control without moving it.

GuiControl, Focus: Sets keyboard focus to the control. To be effective, the window generally must not be minimized or hidden.

GuiControl, Enable / 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. The word Disable or Enable may optionally be followed immediately by a 0 or 1. A zero causes the effect to be inverted. For example, `Enable` and `Enable%VarContainingOne%` would both enable the control, but `Enable%VarContainingZero%` would disable it.

GuiControl, Hide / Show: Hides or shows the control. For Tab controls, this will also show or hide all of the tab's sub-controls. If you additionally want to prevent a control's shortcut key (underlined letter) from working, disable the control via `GuiControl Disable`. The word Hide or Show may optionally be followed immediately by a 0 or 1. A zero causes the effect to be inverted. For example, `Show` and `Show%VarContainingOne%` would both show the control, but `Show%VarContainingZero%` would hide it.

GuiControl, Delete (not yet implemented): This sub-command does not yet exist. As a workaround, use **Hide** and/or **Disable** (above), or destroy and recreate the entire window via [Gui Destroy](#).

GuiControl, 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. [v1.1.06+]: If *N* is zero, the ListBox, DropDownList or ComboBox's current selection is removed.

Unlike [Control Choose](#), this sub-command will not trigger any [g-label](#) associated with the control unless *N* is preceded by a pipe character (even then, the g-label is triggered only when the new selection is different than the old one, at least for [Tab controls](#)). For 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).

To select or deselect all items in a [multi-select ListBox](#), follow this example:

```
Gui +LastFound ; Avoids the need to specify
WinTitle below.
PostMessage, 0x185, 1, -1, ListBox1 ; Select
all items. 0x185 is LB_SETSEL.
PostMessage, 0x185, 0, -1, ListBox1 ; Deselect
all items.
GuiControl, Choose, ListBox1, 0 ; Deselect all
items [requires v1.1.06+].
```

GuiControl, 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 for details).

GuiControl, Font: Changes the control's font to the typeface, size, color, and style currently in effect for its window. For example:

```
Gui, Font, s18 cRed Bold, Verdana ; If desired,
use a line like this to set a new default font
for the window.
GuiControl, Font, MyEdit ; Put the above font
into effect for a control.
```

GuiControl, +/-Option1 +/-Option2 ...: Add or remove various [control-specific](#) or [general](#) options and styles.

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, some of them cannot be applied or removed after a control has been created. `ErrorLevel` is set to 0 if at least one of the specified changes was successfully applied. Otherwise, it is set to 1 to indicate that none of the changes could be applied. Even if a change is successfully applied, the control might choose to ignore it.

[v1.1.20+]: To set a [function object](#) for handling [the control's events](#), *Param3* must be a single variable reference, as in either of the following examples. Other expressions which return objects are currently unsupported.

```
GuiControl +g, ControlID, %FuncObj%  
GuiControl +g, ControlID, % FuncObj
```

Remarks

To operate upon a window other than the default, include its [name or number](#) (or [in v1.1.03+] its `HWND`) followed by a colon in front of the sub-command as in these examples:

```
GuiControl, MyGui:Show, MyButton  
GuiControl, MyGui:, MyListBox, Item1|Item2
```

This is required even if *ControlID* is a control's associated variable, since any one variable can be used on multiple GUI windows. In [v1.1.20+], the GUI name can be omitted if *ControlID* is a control's `HWND`.

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 name](#) 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

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 behaves as though the name of the output variable was specified. For example, `GuiControlGet, MyEdit` is the same as `GuiControlGet, MyEdit,, MyEdit`.

If the target control has an associated variable, specify the variable's name as the *ControlID* (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 input-capable (such as `GroupBox` or `Text`).

Otherwise, *ControlID* can be either `ClassNN` (the classname and instance number of the control) or the control's text, both of which can be determined via `Window Spy`. When using 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*.

[v1.1.04+]: *ControlID* can be the [HWND](#) of a control.

If the control is not on the default GUI, **the name of the GUI must also be specified** -- except on [v1.1.20+] when *ControlID* is a HWND, since each HWND is unique. See [Remarks](#) for details.

Param4

This parameter is omitted except where noted in the list of sub-commands below.

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[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](#): Retrieves the contents but 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.

[UpDown/Slider/Progress](#): Retrieves the control's current position.

[Tab/DropDownList/ComboBox/ListBox](#): 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. For a [multi-select ListBox](#), the output uses the window's [current delimiter](#).

[ListView](#) and [TreeView](#): These are not supported when *Sub-command* is blank. Instead, use the built-in [ListView functions](#) and [TreeView functions](#).

[StatusBar](#): Retrieves only the first part's text.

[ActiveX](#): Retrieves a new wrapper object for the control's ActiveX component.

Note: To unconditionally retrieve the text/caption of a [CheckBox](#), [Radio](#), [DropDownList](#) or [ComboBox](#) rather than its contents, specify the word *Text* for *Param4*.

GuiControlGet, OutputVar, 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*. For example:

```
GuiControlGet, MyEdit, Pos  
MsgBox The X coordinate is %MyEditX%. The Y  
coordinate is %MyEditY%. The width is %MyEditW%.  
The height is %MyEditH%.
```

Within a [function](#), to create a set of variables that is global instead of local, [declare](#) *OutputVar* as a global variable prior to using this command (the converse is true for [assume-global](#) functions). However, it is often also necessary to declare each variable in the set, due to a [common source of confusion](#).

GuiControlGet, OutputVar, Focus: Retrieves the control identifier (ClassNN) 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 usage: `GuiControlGet, focused_control, focus`.

GuiControlGet, OutputVar, FocusV [v1.0.43.06+]: Same as *Focus* (above) except that it retrieves the name of the focused control's [associated variable](#). If that control lacks an associated variable, the first 63 characters of the control's text/caption is retrieved instead (this is most often used to avoid giving each button a variable name).

GuiControlGet, OutputVar, Enabled: Retrieves 1 if the control is enabled or 0 if it is disabled.

GuiControlGet, OutputVar, Visible: Retrieves 1 if the control is visible or 0 if it is hidden.

GuiControlGet, OutputVar, Hwnd [v1.0.46.16+]: Retrieves the window handle (HWND) of the control. A control's HWND is often used with [PostMessage](#), [SendMessage](#), and [DllCall](#). Note: [HwndOutputVar](#) is usually a more concise way to get the HWND.

GuiControlGet, OutputVar, Name [v1.1.03+]: Retrieves the name of the control's [associated variable](#) if it has one, otherwise *OutputVar* is made blank.

Remarks

To operate upon a window other than the default (see below), include its name or number followed by a colon in front of the sub-command as in these examples:

```
GuiControlGet, MyEdit, MyGui:
GuiControlGet, MyEdit, MyGui:Pos
GuiControlGet, OutputVar, MyGui:Focus
```

This is required even if *ControlID* is a control's associated variable, since any one variable can be used on multiple GUI windows. In [v1.1.20+], the GUI name can be omitted if *ControlID* is a control's HWND.

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 name](#) 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
```

ListView

Table of Contents

- [Introduction and Simple Example](#)
- [Options and Styles](#)
- [View Modes: Report \(default\), Icon, Tile, Small-Icon, and List.](#)
- [Built-in Functions:](#)
 - [Row functions \(adding, modifying, and deleting rows\)](#)
 - [Column functions](#)
 - [Getting data out of a ListView](#)
- [G-Label Notifications](#)
- [ImageLists \(the means by which icons are added to a ListView\)](#)
- [ListView Remarks](#)
- [Examples](#)

Introduction and Simple Example

A List-View is one of the most elaborate controls provided by the operating system. In its most recognizable form, it displays a tabular view of rows and columns, the most common example of which is Explorer's list of files and folders (detail view).

Though it may be elaborate, a ListView's basic features are easy to use. The syntax for creating a ListView is:

```
Gui, Add, ListView, Options,  
ColumnTitle1|ColumnTitle2|...
```

Here is a working script that creates and displays a ListView containing a list of files in the user's "My Documents" folder:

```
; Create the ListView with two columns, Name and  
Size:  
Gui, Add, ListView, r20 w700 gMyListView,  
Name|Size (KB)  
  
; Gather a list of file names from a folder and  
put them into the ListView:  
Loop, %A_MyDocuments%\*.*  
    LV_Add("", A_LoopFileName, A_LoopFileSizeKB)  
  
LV_ModifyCol() ; Auto-size each column to fit  
its contents.  
LV_ModifyCol(2, "Integer") ; For sorting  
purposes, indicate that column 2 is an integer.  
  
; Display the window and return. The script will  
be notified whenever the user double clicks a  
row.
```

```
Gui, Show
return

MyListView:
if A_GuiEvent = DoubleClick
{
    LV_GetText(RowText, A_EventInfo) ; Get the
text from the row's first field.
    ToolTip You double-clicked row number
%A_EventInfo%. Text: "%RowText%"
}
return

GuiClose: ; Indicate that the script should
exit automatically when the window is closed.
ExitApp
```

Options and Styles for "*Gui, Add, ListView, Options*"

AltSubmit: Notifies the script for more types of ListView events than normal. In other words, the g-label is launched more often. See [ListView Notifications](#) for details.

Background: Specify the word Background followed immediately by a color name (see [color chart](#)) or RGB value (the 0x prefix is optional). Examples: `BackgroundSilver` , `BackgroundFFDD99` . If this option is not present, the ListView initially defaults to the background color set by the last parameter of [Gui Color](#) (or if none, the system's default background color). Specifying `BackgroundDefault` applies the system's default background color (usually white). For example, a ListView can be restored to the default color via `GuiControl, +BackgroundDefault, MyListView` .

C: Text color. Specify the letter C followed immediately by a color name (see [color chart](#)) or RGB value (the 0x prefix is optional). Examples: `cRed` , `cFF2211` , `c0xFF2211` , `cDefault` .

Checked: Provides a checkbox at the left side of each row. When [adding](#) a row, specify the word *Check* in its options to have the box to start off checked instead of unchecked. The user may either click the checkbox or press the spacebar to check or uncheck a row.

Count: Specify the word Count followed immediately by the total number of rows that the ListView will ultimately contain. This is not a limit: rows beyond the count can still be added. Instead, this option serves as a hint to the control that allows it to allocate memory only once rather than each time a row is added, which greatly improves row-adding performance (it may also improve sorting performance). To improve performance even more, use `GuiControl, -Redraw, MyListView` prior to adding a large number of rows. Afterward, use `GuiControl, +Redraw, MyListView` to re-enable redrawing (which also repaints the control).

Grid: Provides horizontal and vertical lines to visually indicate the boundaries between rows and columns.

Hdr: Specify `-Hdr` (minus Hdr) to omit the special top row that contains column titles. To make it visible later, use `GuiControl, +Hdr, MyListView`.

LV: Specify the string LV followed immediately by the number of an [extended ListView style](#). These styles are entirely separate from generic extended styles. For example, specifying `-E0x200` would remove the generic extended style `WS_EX_CLIENTEDGE` to eliminate the control's default border. By contrast, specifying `-LV0x20` would remove [LVS_EX_FULLROWSELECT](#).

LV0x10: Specify `-LV0x10` to prevent the user from dragging column headers to the left or right to reorder them. However, it is usually not necessary to do this because the physical reordering of columns does not affect the column order seen by the script. For example, the first column will always be column 1 from the script's point of view, even if the user has physically moved it to the right of other columns.

LV0x20: Specify `-LV0x20` to require that a row be clicked at its first field to select it (normally, a click on *any* field will select it). The advantage of this is that it makes it easier for the user to drag a rectangle around a group of rows to select them.

Multi: Specify `-Multi` (minus Multi) to prevent the user from selecting more than one row at a time.

NoSortHdr: Prevents the header from being clickable. It will take on a flat appearance rather than its normal button-like appearance. Unlike most other ListView styles, this one cannot be changed after the ListView is created.

NoSort: Turns off the automatic sorting that occurs when the user clicks a column header. However, the header will still behave visually like a button (unless NoSortHdr has been specified). In addition, the g-label will still receive the [ColClick notification](#), to which it can respond with a custom sort or other action.

ReadOnly: Specify `-ReadOnly` (minus ReadOnly) to allow editing of the text in the first column of each row. To edit a row, select it then press the `F2`  key (see the [WantF2](#) option below). Alternatively, you can click a row once to select it, wait at least half a second, then click the same row again to edit it.

R: Rows of height (upon creation). Specify the letter R followed immediately by the number of rows for which to make room inside the control. For example, `R10` would make the control 10 rows tall. If the ListView is created with a [view mode](#) other than report view, the control is sized to fit rows of icons instead of rows of text. Note: adding [icons](#) to a ListView's rows will increase the height of each row, which will make this option inaccurate.

Sort: The control is kept alphabetically sorted according to the contents of the first column.

SortDesc: Same as above except in descending order.

WantF2 [v1.0.44+]: Specify `-WantF2` (minus WantF2) to prevent an `F2` keystroke from [editing](#) the currently focused row. This setting is ignored unless `-ReadOnly` is also in effect. Regardless of this setting, the g-label still receives `F2` [notifications](#).

(Unnamed numeric styles): Since styles other than the above are rarely used, they do not have names. See the [ListView styles table](#) for a list.

View Modes

A `ListView` has five viewing modes, of which the most common is report view (which is the default). To use one of the other views, specify its name in the options list. The view can also be changed after the control is created; for example: `GuiControl, +IconSmall, MyListView`.

Icon: Shows a large-icon view. In this view and all the others except *Report*, the text in columns other than the first is not visible. To display icons in this mode, the `ListView` must have a large-icon `ImageList` assigned to it.

Tile: Shows a large-icon view but with ergonomic differences such as displaying each item's text to the right of the icon rather than underneath it. `Checkboxes` do not function in this view. Also, attempting to show this view on operating systems older than Windows XP has no effect.

IconSmall: Shows a small-icon view.

List: Shows a small-icon view in list format, which displays the icons in columns. The number of columns depends on the width of the control and the width of the widest text item in it.

Report: Switches back to report view, which is the initial default. For example: `GuiControl, +Report, MyListView`.

Built-in Functions for ListView

All of the ListView functions operate upon the current thread's [default GUI window](#) (which can be changed via `Gui, 2:Default`). If the default window does not exist or has no ListView controls, all functions return zero to indicate the problem.

If the window has more than one ListView control, by default the functions operate upon the one most recently added. To change this, specify `Gui, ListView, ListViewName`, where *ListViewName* is the name of the ListView's [associated variable](#), its ClassNN as shown by Window Spy or [in v1.1.04+] its HWND. Once changed, all existing and future [threads](#) will use the indicated ListView. [v1.1.23+]: [A_DefaultListView](#) contains the current setting.

When the phrase "row number" is used on this page, it refers to a row's current position within the ListView. The top row is 1, the second row is 2, and so on. After a row is added, its row number tends to change due to sorting, deleting, and inserting of other rows. Therefore, to locate specific row(s) based on their contents, it is usually best to use `LV_GetText()` in a loop.

Row Functions

LV_Add([Options, Field1, Field2, ...])

Adds a new row to the bottom of the list. The parameters *Field1* and beyond are the columns of the new row, which can be text or numeric (including numeric [expression](#) results). To make any field blank, specify "" or the equivalent. If there are too few fields to fill all the columns, the columns at the end are left blank. If there are too many fields, the fields at the end are completely ignored.

Upon failure, LV_Add() returns 0. Upon success, it returns the new [row number](#), which is not necessarily the last row if the ListView has the [Sort](#) or [SortDesc](#) style.

Row Options

The *Options* parameter is a string containing zero or more words from the list below (not case sensitive). Separate each word from the next with a space or tab. To remove an option, precede it with a minus sign. To add an option, a plus sign is permitted but not required.

Check: Shows a checkmark in the row (if the ListView has [checkboxes](#)). To later uncheck it, use `LV_Modify(RowNumber, "-Check")`.

Col: Specify the word Col followed immediately by the column number at which to begin applying the parameters *Col1* and beyond. This is most commonly used with [LV_Modify\(\)](#) to alter individual fields in a row without affecting those that lie to their left.

Focus: Sets keyboard focus to the row (often used in conjunction with [Select](#)). To later de-focus it, use `LV_Modify(RowNumber, "-Focus")`.

Icon: Specify the word Icon followed immediately by the number of this row's icon, which is displayed in the left side of the first column. If this option is absent, the first icon in the [ImageList](#) is used. To display a blank icon, specify a number that is larger than the number of icons in the ImageList. If the control lacks a small-icon ImageList, no icon is displayed nor is any space reserved for

one in [report view](#).

Select: Selects the row. To later deselect it, use `LV_Modify(RowNumber, "-Select")`. When selecting rows, it is usually best to ensure that at least one row always has the [focus property](#) because that allows the Apps key to display its [context menu](#) (if any) near the focused row. The word *Select* may optionally be followed immediately by a 0 or 1 to indicate the starting state. In other words, both `"Select"` and `"Select"` . `VarContainingOne` are the same (the period used here is the [concatenation operator](#)). This technique also works with *Focus* and *Check* above.

Vis [v1.0.44+]: Ensures that the specified row is completely visible by scrolling the ListView, if necessary. This has an effect only for `LV_Modify()`; for example: `LV_Modify(RowNumber, "Vis")`.

LV_Insert(RowNumber [, Options, Col1, Col2, ...])

Behaves identically to `LV_Add()` except for its different first parameter, which specifies the row number for the newly inserted row. Any rows at or beneath *RowNumber* are shifted downward to make room for the new row. If *RowNumber* is greater than the number of rows in the list (even as high as 2147483647), the new row is added to the end of the list. For *Options*, see [row options](#).

LV_Modify(RowNumber [, Options, NewCol1, NewCol2, ...])

Modifies the attributes and/or text of a row, and returns 1 upon success and 0 upon failure. If *RowNumber* is 0, all rows in the control are modified (in this case the function returns 1 on complete success and 0 if any part of the operation failed). When only the first two parameters are present, only the row's attributes and not its text are changed. Similarly, if there are too few parameters to cover all the columns, the columns at the end are not changed. The [ColN option](#) may be used to update specific columns without affecting the others. For other options, see [row options](#).

LV_Delete([RowNumber])

If the parameter is omitted, **all** rows in the ListView are deleted. Otherwise, only

the specified *RowNumber* is deleted. It returns 1 upon success and 0 upon failure.

Column Functions

LV_ModifyCol([ColumnNumber, Options, ColumnTitle])

Modifies the attributes and/or text of the specified column and its header. The first column is number 1 (not 0). If all parameters are omitted, the width of every column is adjusted to fit the contents of the rows. If only the first parameter is present, only the specified column is auto-sized. Auto-sizing has no effect when not in Report (Details) view. This function returns 1 upon success and 0 upon failure.

Column Options

The *Options* parameter is a string containing zero or more words from the list below (not case sensitive). Separate each word from the next with a space or tab. To remove an option, precede it with a minus sign. To add an option, a plus sign is permitted but not required.

Column Options: General

N: Specify for N the new width of the column, in pixels. This number can be unquoted if is the only option. For example, the following are both valid:
`LV_ModifyCol(1, 50)` and `LV_ModifyCol(1, "50 Integer")`.

Auto: Adjusts the column's width to fit its contents. This has no effect when not in Report (Details) view.

AutoHdr: Adjusts the column's width to fit its contents and the column's header text, whichever is wider. If applied to the last column, it will be made at least as wide as all the remaining space in the ListView. It is usually best to apply this setting only after the rows have been added because that allows any newly-arrived vertical scroll bar to be taken into account when sizing the last column. This option has no effect when not in Report (Details) view.

Icon: Specify the word Icon followed immediately by the number of the [ImageList's](#) icon to display next to the column header's text. Specify `-ICON` (minus icon) to remove any existing icon.

IconRight: Puts the icon on the right side of the column rather than the left.

Column Options: Data Type

Float: For sorting purposes, indicates that this column contains floating point numbers (hexadecimal format is not supported). Sorting performance for Float and Text columns is up to 25 times slower than it is for integers.

Integer: For sorting purposes, indicates that this column contains integers. To be sorted properly, each integer must be 32-bit; that is, within the range -2147483648 to 2147483647. If any of the values are not integers, they will be considered zero when sorting (unless they start with a number, in which case that number is used). Numbers may appear in either decimal or hexadecimal format (e.g. `0xF9E0`).

Text: Changes the column back to text-mode sorting, which is the initial default for every column. Only the first 8190 characters of text are significant for sorting purposes (except for the *Logical option*, in which case the limit is 4094).

Column Options: Alignment / Justification

Center: Centers the text in the column. To center an Integer or Float column, specify the word Center after the word Integer or Float.

Left: Left-justifies the column's text, which is the initial default for every column. On older operating systems, the first column might have a forced left-justification.

Right: Right-justifies the column's text. This attribute need not be specified for Integer and Float columns because they are right-justified by default. That default can be overridden by specifying something such as `"Integer Left"` or `"Float Center"`.

Column Options: Sorting

Case: The sorting of the column is case sensitive (affects only `text` columns). If the options *Case*, *CaseLocale*, and *Logical* are all omitted, the uppercase letters A-Z are considered identical to their lowercase counterparts for the purpose of the sort.

CaseLocale [v1.0.43.03+]: The sorting of the column is case insensitive based on the current user's locale (affects only [text](#) columns). For example, most English and Western European locales treat the letters A-Z and ANSI letters like Ä and Ü as identical to their lowercase counterparts. This method also uses a "word sort", which treats hyphens and apostrophes in such a way that words like "coop" and "co-op" stay together.

Desc: Descending order. The column starts off in descending order the first time the user sorts it.

Logical [v1.0.44.12+]: Same as *CaseLocale* except that any sequences of digits in the text are treated as true numbers rather than mere characters. For example, the string "T33" would be considered greater than "T4". *Logical* requires Windows XP or later (on older OSes, *CaseLocale* is automatically used instead). In addition, *Logical* and *Case* are currently mutually exclusive: only the one most recently specified will be in effect.

NoSort: Prevents a user's click on this column from having any automatic sorting effect. To disable sorting for all columns rather than only a subset, include [NoSort](#) in the ListView's options. If the ListView has a g-label, the [ColClick notification](#) will still be received when the user clicks a no-sort column.

Sort: Immediately sorts the column in ascending order (even if it has the [Desc](#) option).

SortDesc: Immediately sorts the column in descending order.

Uni: Unidirectional sort. This prevents a second click on the same column from reversing the sort direction.

[LV_InsertCol\(ColumnNumber \[, Options, ColumnTitle\]\)](#)

Creates a new column, inserting it as the specified *ColumnNumber* (shifting any other columns to the right to make room). The first column is 1 (not 0). If *ColumnNumber* is larger than the number of columns currently in the control, the new column is added to the end of the list. The newly inserted column starts off with empty contents beneath it unless it is the first column, in which case it inherits the old first column's contents and the old first column acquires blank

contents. The new column's attributes -- such as whether or not it uses [integer sorting](#) -- always start off at their defaults unless changed via [Options](#). This function returns the new column's position number (or 0 upon failure). The maximum number of columns in a ListView is 200.

LV_DeleteCol(ColumnNumber)

Deletes the specified column and all of the contents beneath it. It returns 1 upon success and 0 upon failure. Once a column is deleted, the column numbers of any that lie to its right are reduced by 1. Consequently, calling `LV_DeleteCol(2)` twice would delete the second and third columns. On operating systems older than Windows XP, attempting to delete the original first column might fail and return 0.

Getting Data Out of a ListView

LV_GetCount(["Selected | Column"])

When the parameter is omitted, the function returns the total number of rows in the control. When the parameter is "S" or "Selected", the count includes only the selected/highlighted rows. When the parameter is "Col" or "Column", the function returns the number of columns in the control. This function is always instantaneous because the control keeps track of these counts.

This function is often used in the top line of a Loop, in which case the function would get called only once (prior to the first iteration). For example:

```
Loop % LV_GetCount()  
{  
    LV_GetText(RetrievedText, A_Index)  
    if InStr(RetrievedText, "some filter text")  
        LV_Modify(A_Index, "Select") ; Select  
each row whose first field contains the filter-  
text.  
}
```

To retrieve the widths of a ListView's columns -- for uses such as saving them to an INI file to be remembered between sessions -- follow this example:

```
Gui +LastFound  
Loop % LV_GetCount("Column")  
{  
    SendMessage, 4125, A_Index - 1, 0,  
SysListView321 ; 4125 is LVM_GETCOLUMNWIDTH.  
    MsgBox Column %A_Index%'s width is  
%ErrorLevel%.  
}
```

LV_GetNext([StartingRowNumber, "Checked | Focused"])

Returns the row number of the next selected, checked, or focused row. If none is found, zero is returned. If *StartingRowNumber* is omitted or less than 1, the search begins at the top of the list. Otherwise, the search begins at the row after *StartingRowNumber*. If the second parameter is omitted, the function searches for the next selected/highlighted row. Otherwise, specify "C" or "Checked" to find the next checked row; or "F" or "Focused" to find the focused row (there is never more than one focused row in the entire list, and sometimes there is none at all). The following example reports all selected rows in the ListView:

```
RowNumber = 0 ; This causes the first loop
iteration to start the search at the top of the
list.
Loop
{
    RowNumber := LV_GetNext(RowNumber) ; Resume
the search at the row after that found by the
previous iteration.
    if not RowNumber ; The above returned zero,
so there are no more selected rows.
        break
    LV_GetText(Text, RowNumber)
    MsgBox The next selected row is
    #%RowNumber%, whose first field is "%Text%".
}
```

An alternate method to find out if a particular row number is checked is the following:

```
Gui +LastFound
SendMessage, 4140, RowNumber - 1, 0xF000,
SysListView321 ; 4140 is LVM_GETITEMSTATE.
0xF000 is LVIS_STATEIMAGEMASK.
IsChecked := (ErrorLevel >> 12) - 1 ; This sets
IsChecked to true if RowNumber is checked or
```

false otherwise.

LV_GetText(OutputVar, RowNumber [, ColumnNumber])

Retrieves the text at the specified *RowNumber* and *ColumnNumber* and stores it in *OutputVar*. If *ColumnNumber* is omitted, it defaults to 1 (the text in the first column). If *RowNumber* is 0, the column header text is retrieved. If the text is longer than 8191, only the first 8191 characters are retrieved. The function returns 1 upon success and 0 upon failure. Upon failure, *OutputVar* is also made blank.

Column numbers seen by the script are not altered by any dragging and dropping of columns the user may have done. For example, the original first column is still number 1 even if the user drags it to the right of other columns.

G-Label Notifications (Primary)

A **g-label** such as `gMySubroutine` may be listed in the control's options. This would cause the *MySubroutine* label to be launched automatically whenever the user performs an action in the control. This subroutine may consult the built-in variables `A_Gui` and `A_GuiControl` to find out which window and ListView generated the event. More importantly, it may consult `A_GuiEvent`, which contains one of the following strings or letters (for compatibility with future versions, a script should not assume these are the only possible values):

DoubleClick: The user has double-clicked within the control. The variable `A_EventInfo` contains the focused row number. `LV_GetNext()` can be used to instead get the first *selected* row number, which is 0 if the user double-clicked on empty space.

R: The user has *double-right*-clicked within the control. The variable `A_EventInfo` contains the focused row number.

ColClick: The user has clicked a column header. The variable `A_EventInfo` contains the column number, which is the original number assigned when the column was created; that is, it does not reflect any dragging and dropping of columns done by the user. One possible response to a column click is to sort by a hidden column (zero width) that contains data in a sort-friendly format (such as a YYYYMMDD integer date). Such a hidden column can mirror some other column that displays the same data in a more friendly format (such as MM/DD/YY). For example, a script could hide column 3 via `LV_ModifyCol(3, 0)`, then disable automatic sorting in the visible column 2 via `LV_ModifyCol(2, "NoSort")`. Then in response to the ColClick notification for column 2, the script would sort the ListView by the hidden column via `LV_ModifyCol(3, "Sort")`.

D: The user has attempted to start dragging a row or icon (there is currently no built-in support for dragging rows or icons). The variable `A_EventInfo` contains the focused row number. [v1.0.44+]: This notification occurs even without `AltSubmit`.

d (lowercase D): Same as above except a right-click-drag rather than a left-drag.

e (lowercase E): The user has finished editing the first field of a row (the user may edit it only when the ListView has `-ReadOnly` in its options). The variable `A_EventInfo` contains the row number.

G-Label Notifications (Secondary)

If the ListView has the word AltSubmit in its [options](#), its g-label is launched more often and **A_GuiEvent** may contain the following additional values:

Normal: The user has left-clicked a row. The variable A_EventInfo contains the focused row number.

RightClick: The user has right-clicked a row. The variable A_EventInfo contains the focused row number. In most cases, it is best not to display a menu in response to this. Instead, use the [GuiContextMenu](#) label because it also recognizes the Apps key. For example:

```
GuiContextMenu: ; Launched in response to a
right-click or press of the Apps key.
if A_GuiControl <> MyListView ; This check is
optional. It displays the menu only for clicks
inside the ListView.
    return
; Show the menu at the provided coordinates,
A_GuiX and A_GuiY. These should be used
; because they provide correct coordinates even
if the user pressed the Apps key:
Menu, MyContextMenu, Show, %A_GuiX%, %A_GuiY%
return
```

A: A row has been activated, which by default occurs when it is double clicked. The variable A_EventInfo contains the row number.

C: The ListView has released mouse capture.

E: The user has begun editing the first field of a row (the user may edit it only when the ListView has [-ReadOnly](#) in its options). The variable A_EventInfo contains the row number.

F: The ListView has received keyboard focus.

f (lowercase F): The ListView has lost keyboard focus.

I: Item changed. A row has changed by becoming selected/deselected, checked/unchecked, etc. If the user selects a new row, at least two such notifications are received: one for the de-selection of the previous row, and one for the selection of the new row. [v1.0.44+]: The variable `A_EventInfo` contains the row number. [v1.0.46.10+]: `ErrorLevel` contains zero or more of the following letters to indicate how the item changed: S (select) or s (de-select), and/or F (focus) or f (de-focus), and/or C (checkmark) or c (uncheckmark). For example, SF means that the row has been selected and focused. To detect whether a particular letter is present, use a [parsing loop](#) or the case-sensitive option of `InStr()`; for example: `InStr(ErrorLevel, "S", true)`. Note: For compatibility with future versions, a script should not assume that "SsFfCc" are the only possible letters. Also, specifying [Critical](#) as the [g-label](#)'s first line ensures that all "I" notifications are received (otherwise, some might be lost if the script cannot keep up with them).

K: The user has pressed a key while the ListView has focus. `A_EventInfo` contains the virtual key code of the key, which is a number between 1 and 255. This can be translated to a key name or character via [GetKeyName](#). For example, `key := GetKeyName(Format("vk{:x}", A_EventInfo))`. On most keyboard layouts, keys A-Z can be translated to the corresponding character via `Chr(A_EventInfo)`. **F2**  strokes are received regardless of [WantF2](#). However, the **Enter**  is not received; to receive it, use a default button as described [below](#).

M: Marquee. The user has started to drag a selection-rectangle around a group of rows or icons.

S: The user has begun scrolling the ListView.

s (lowercase S): The user has finished scrolling the ListView.

ImageList (the means by which icons are added to a ListView)

An Image-List is a group of identically sized icons stored in memory. Upon creation, each ImageList is empty. The script calls `IL_Add()` repeatedly to add icons to the list, and each icon is assigned a sequential number starting at 1. This is the number to which the script refers to display a particular icon in a row or column header. Here is a working example that demonstrates how to put icons into a ListView's rows:

```
Gui, Add, ListView, h200 w180, Icon &
Number|Description ; Create a ListView.
ImageListID := IL_Create(10) ; Create an
ImageList to hold 10 small icons.
LV_SetImageList(ImageListID) ; Assign the above
ImageList to the current ListView.
Loop 10 ; Load the ImageList with a series of
icons from the DLL.
    IL_Add(ImageListID, "shell32.dll", A_Index)
Loop 10 ; Add rows to the ListView (for
demonstration purposes, one for each icon).
    LV_Add("Icon" . A_Index, A_Index, "n/a")
LV_ModifyCol("Hdr") ; Auto-adjust the column
widths.
Gui Show
return

GuiClose: ; Exit the script when the user
closes the ListView's GUI window.
ExitApp
```

IL_Create([InitialCount, GrowCount, LargeIcons?])

Creates a new ImageList, initially empty, and returns the unique ID of the ImageList (or 0 upon failure). *InitialCount* is the number of icons you expect to put into the list immediately (if omitted, it defaults to 2). *GrowCount* is the

number of icons by which the list will grow each time it exceeds the current list capacity (if omitted, it defaults to 5). *LargeIcons* should be a numeric value: If non-zero, the ImageList will contain large icons. If zero, it will contain small icons (this is the default when omitted). Icons added to the list are scaled automatically to conform to the system's dimensions for small and large icons.

LV_SetImageList(ImageListID [, 0|1|2])

This function is normally called prior to adding any rows to the ListView. It sets the [ImageList](#) whose icons will be displayed by the ListView's rows (and optionally, its columns). *ImageListID* is the number returned from a previous call to [IL_Create\(\)](#). If the second parameter is omitted, the type of icons in the ImageList is detected automatically as large or small. Otherwise, specify 0 for large icons, 1 for small icons, and 2 for state icons (state icons are not yet directly supported, but they could be used via [SendMessage](#)).

A ListView may have up to two ImageLists: small-icon and/or large-icon. This is useful when the script allows the user to switch to and from the large-icon view. To add more than one ImageList to a ListView, call [LV_SetImageList\(\)](#) a second time, specifying the *ImageListID* of the second list. A ListView with both a large-icon and small-icon ImageList should ensure that both lists contain the icons in the same order. This is because the same ID number is used to reference both the large and small versions of a particular icon.

Although it is traditional for all [viewing modes](#) except Icon and Tile to show small icons, this can be overridden by passing a large-icon list to [LV_SetImageList](#) and specifying 1 (small-icon) for the second parameter. This also increases the height of each row in the ListView to fit the large icon.

If successful, [LV_SetImageList\(\)](#) returns the *ImageListID* that was previously associated with the ListView (or 0 if none). Any such detached ImageList should normally be destroyed via [IL_Destroy\(ImageListID\)](#).

IL_Add(ImageListID, Filename [, IconNumber, ResizeNonIcon?])

Adds an icon or picture to the specified *ImageListID* and returns the new icon's index (1 is the first icon, 2 is the second, and so on). *Filename* is the name of an icon (.ICO), cursor (.CUR), or animated cursor (.ANI) file (animated cursors will not actually be animated when displayed in a ListView). Other sources of

icons include the following types of files: EXE, DLL, CPL, SCR, and other types that contain icon resources. To use an icon group other than the first one in the file, specify its number for *IconNumber*. If *IconNumber* is negative, its absolute value is assumed to be the resource ID of an icon within an executable file. In the following example, the default icon from the second icon group would be used: `IL_Add(ImageListID, "C:\My Application.exe", 2)`.

Non-icon images such as BMP, GIF and JPG may also be loaded. However, in this case the last two parameters should be specified to ensure correct behavior: *IconNumber* should be the mask/transparency color number (0xFFFFFFFF [the color white] might be best for most pictures); and *ResizeNonIcon* should be non-zero to cause the picture to be scaled to become a single icon, or zero to divide up the image into however many icons can fit into its actual width.

All operating systems support GIF, JPG, BMP, ICO, CUR, and ANI images. On Windows XP or later, additional image formats such as PNG, TIF, 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

[v1.1.23+]: A [bitmap or icon handle](#) can be used instead of a filename. For example, `HBITMAP:%handle%`.

IL_Destroy(ImageListID)

Deletes the specified ImageList and returns 1 upon success and 0 upon failure. It is normally not necessary to destroy ImageLists because once attached to a ListView, they are destroyed automatically when the ListView or its parent window is destroyed. However, if the ListView shares ImageLists with other ListViews (by having `0x40` in its options), the script should explicitly destroy the ImageList after destroying all the ListViews that use it. Similarly, if the script replaces one of a ListView's old ImageLists with a new one, it should explicitly destroy the old one.

Listview Remarks

The [Gui Submit](#) command has no effect on a ListView control. Therefore, the script may use the ListView's [associated variable](#) (if any) to store other data without concern that it will ever be overwritten.

After a column is sorted -- either by means of the user clicking its header or the script calling `LV_ModifyCol(1, "Sort")` -- any subsequently added rows will appear at the bottom of the list rather than obeying the sort order. The exception to this is the [Sort](#) and [SortDesc](#) styles, which move newly added rows into the correct positions.

To detect when the user has pressed  ListView has focus, use a [default button](#) (which can be hidden if desired). For example:

```
Gui, Add, Button, Hidden Default, OK
...
ButtonOK:
GuiControlGet, FocusedControl, FocusV
if FocusedControl <> MyListView
    return
MsgBox % "Enter was pressed. The focused row
number is " . LV_GetNext(0, "Focused")
return
```

In addition to navigating from row to row with the keyboard, the user may also perform incremental search by typing the first few characters of an item in the first column. This causes the selection to jump to the nearest matching row.

Although any length of text can be stored in each field of a ListView, only the first 260 characters are displayed.

Although the maximum number of rows in a ListView is limited only by available system memory, row-adding performance can be greatly improved as described in the [Count](#) option.

A picture may be used as a background around a ListView (that is, to frame the ListView). To do this, create the [picture control](#) after the ListView and include `0x40000000` (which is `WS_CLIPSIBLINGS`) in the picture's *Options*.

A script may create more than one ListView per window. To operate upon a ListView other than the default one, see [built-in functions](#).

It is best not to insert or delete columns directly with [SendMessage](#). This is because the program maintains a collection of [sorting preferences](#) for each column, which would then get out of sync. Instead, use the [built-in column functions](#).

To perform actions such as resizing, hiding, or changing the font of a ListView, use [GuiControl](#).

To extract text from external ListViews (those not owned by the script), use [ControlGet List](#).

Related

[TreeView](#), [Other Control Types](#), [Gui](#), [GuiContextMenu](#), [GuiControl](#), [GuiControlGet](#), [ListView styles table](#)

Examples

```
; Select or de-select all rows by specifying 0 as  
the row number:
```

```
LV_Modify(0, "Select")    ; Select all.  
LV_Modify(0, "-Select")  ; De-select all.  
LV_Modify(0, "-Check")   ; Uncheck all the  
checkboxes.
```

```
; Auto-size all columns to fit their contents:  
LV_ModifyCol() ; There are no parameters in this  
mode.
```

```
; MAIN EXAMPLE  
; The following is a working script that is more  
elaborate than the one near the top of this page.  
; It displays the files in a folder chosen by the  
user, with each file assigned the icon associated  
with  
; its type. The user can double-click a file, or  
right-click one or more files to display a context  
menu.
```

```
; Allow the user to maximize or drag-resize the  
window:
```

```
Gui +Resize
```

```
; Create some buttons:
```

```
Gui, Add, Button, Default gButtonLoadFolder, Load  
a folder
```

```
Gui, Add, Button, x+20 gButtonClear, Clear List
```

```
Gui, Add, Button, x+20, Switch View
```

```
; Create the ListView and its columns:
Gui, Add, ListView, xm r20 w700 vMyListView
gMyListView, Name|In Folder|Size (KB)|Type
LV_ModifyCol(3, "Integer") ; For sorting,
indicate that the Size column is an integer.

; Create an ImageList so that the ListView can
display some icons:
ImageListID1 := IL_Create(10)
ImageListID2 := IL_Create(10, 10, true) ; A list
of large icons to go with the small ones.

; Attach the ImageLists to the ListView so that it
can later display the icons:
LV_SetImageList(ImageListID1)
LV_SetImageList(ImageListID2)

; Create a popup menu to be used as the context
menu:
Menu, MyContextMenu, Add, Open, ContextOpenFile
Menu, MyContextMenu, Add, Properties,
ContextProperties
Menu, MyContextMenu, Add, Clear from ListView,
ContextClearRows
Menu, MyContextMenu, Default, Open ; Make "Open"
a bold font to indicate that double-click does the
same thing.

; Display the window and return. The OS will
notify the script whenever the user
; performs an eligible action:
Gui, Show
return
```

ButtonLoadFolder:

```

Gui +OwnDialogs ; Forces user to dismiss the
following dialog before using main window.
FileSelectFolder, Folder,, 3, Select a folder to
read:
if not Folder ; The user canceled the dialog.
    return

; Check if the last character of the folder name
is a backslash, which happens for root
; directories such as C:\. If it is, remove it to
prevent a double-backslash later on.
StringRight, LastChar, Folder, 1
if LastChar = \
    StringTrimRight, Folder, Folder, 1 ; Remove
the trailing backslash.

; Calculate buffer size required for SHFILEINFO
structure.
sfi_size := A_PtrSize + 8 + (A_IsUnicode ? 680 :
340)
VarSetCapacity(sfi, sfi_size)

; Gather a list of file names from the selected
folder and append them to the ListView:
GuiControl, -Redraw, MyListView ; Improve
performance by disabling redrawing during load.
Loop %Folder%\*.*
{
    FileName := A_LoopFileFullPath ; Must save it
to a writable variable for use below.

    ; Build a unique extension ID to avoid
characters that are illegal in variable names,
    ; such as dashes. This unique ID method also
performs better because finding an item
    ; in the array does not require search-loop.
    SplitPath, FileName,,, FileExt ; Get the

```

```

file's extension.
  if FileExt in EXE,ICO,ANI,CUR
  {
      ExtID := FileExt ; Special ID as a
placeholder.
      IconNumber = 0 ; Flag it as not found so
that these types can each have a unique icon.
  }
  else ; Some other extension/file-type, so
calculate its unique ID.
  {
      ExtID = 0 ; Initialize to handle
extensions that are shorter than others.
      Loop 7 ; Limit the extension to 7
characters so that it fits in a 64-bit value.
      {
          StringMid, ExtChar, FileExt, A_Index,
1
          if not ExtChar ; No more characters.
break
          ; Derive a Unique ID by assigning a
different bit position to each character:
          ExtID := ExtID | (Asc(ExtChar) << (8 *
(A_Index - 1)))
      }
      ; Check if this file extension already has
an icon in the ImageLists. If it does,
      ; several calls can be avoided and loading
performance is greatly improved,
      ; especially for a folder containing
hundreds of files:
      IconNumber := IconArray%ExtID%
  }
  if not IconNumber ; There is not yet any icon
for this extension, so load it.
  {
      ; Get the high-quality small-icon

```

associated with this file extension:

```
    if not DllCall("Shell32\SHGetFileInfo" .
(A_IsUnicode ? "W":"A"), "Str", FileName
    , "UInt", 0, "Ptr", &sfi, "UInt",
sfi_size, "UInt", 0x101) ; 0x101 is
SHGFI_ICON+SHGFI_SMALLICON
        IconNumber = 9999999 ; Set it out of
bounds to display a blank icon.
    else ; Icon successfully loaded.
    {
        ; Extract the hIcon member from the
structure:
        hIcon := NumGet(sfi, 0)
        ; Add the HICON directly to the small-
icon and large-icon lists.
        ; Below uses +1 to convert the
returned index from zero-based to one-based:
        IconNumber :=
DllCall("ImageList_ReplaceIcon", "Ptr",
ImageListID1, "Int", -1, "Ptr", hIcon) + 1
        DllCall("ImageList_ReplaceIcon",
"Ptr", ImageListID2, "Int", -1, "Ptr", hIcon)
        ; Now that it's been copied into the
ImageLists, the original should be destroyed:
        DllCall("DestroyIcon", "Ptr", hIcon)
        ; Cache the icon to save memory and
improve loading performance:
        IconArray%ExtID% := IconNumber
    }
}

; Create the new row in the ListView and
assign it the icon number determined above:
LV_Add("Icon" . IconNumber, A_LoopFileName,
A_LoopFileDir, A_LoopFileSizeKB, FileExt)
}
GuiControl, +Redraw, MyListView ; Re-enable
```

```
redrawing (it was disabled above).
LV_ModifyCol() ; Auto-size each column to fit its
contents.
LV_ModifyCol(3, 60) ; Make the Size column at
little wider to reveal its header.
return
```

ButtonClear:

```
LV_Delete() ; Clear the ListView, but keep icon
cache intact for simplicity.
return
```

ButtonSwitchView:

```
if not IconView
    GuiControl, +Icon, MyListView ; Switch to
icon view.
else
    GuiControl, +Report, MyListView ; Switch back
to details view.
IconView := not IconView ; Invert in
preparation for next time.
return
```

MyListView:

```
if A_GuiEvent = DoubleClick ; There are many
other possible values the script can check.
{
    LV_GetText(FileName, A_EventInfo, 1) ; Get the
text of the first field.
    LV_GetText(FileDir, A_EventInfo, 2) ; Get the
text of the second field.
    Run %FileDir%\%FileName%,, UseErrorLevel
    if ErrorLevel
        MsgBox Could not open
"%FileDir%\%FileName%".
}
```

```
return
```

```
GuiContextMenu: ; Launched in response to a  
right-click or press of the Apps key.  
if A_GuiControl <> MyListView ; Display the menu  
only for clicks inside the ListView.  
return  
; Show the menu at the provided coordinates,  
A_GuiX and A_GuiY. These should be used  
; because they provide correct coordinates even if  
the user pressed the Apps key:  
Menu, MyContextMenu, Show, %A_GuiX%, %A_GuiY%  
return
```

```
ContextOpenFile: ; The user selected "Open" in  
the context menu.  
ContextProperties: ; The user selected  
"Properties" in the context menu.  
; For simplicity, operate upon only the focused  
row rather than all selected rows:  
FocusedRowIndex := LV_GetNext(0, "F") ; Find the  
focused row.  
if not FocusedRowIndex ; No row is focused.  
return  
LV_GetText(FileName, FocusedRowIndex, 1) ; Get  
the text of the first field.  
LV_GetText(FileDir, FocusedRowIndex, 2) ; Get  
the text of the second field.  
IfInString A_ThisMenuItem, Open ; User selected  
"Open" from the context menu.  
Run %FileDir%\%FileName%,, UseErrorLevel  
else ; User selected "Properties" from the  
context menu.  
Run Properties "%FileDir%\%FileName%",,  
UseErrorLevel  
if ErrorLevel  
MsgBox Could not perform requested action on
```

```
"%FileDir%\%FileName%".
```

```
return
```

```
ContextClearRows: ; The user selected "Clear" in  
the context menu.
```

```
RowNumber = 0 ; This causes the first iteration  
to start the search at the top.
```

```
Loop
```

```
{
```

```
    ; Since deleting a row reduces the RowNumber  
of all other rows beneath it,
```

```
    ; subtract 1 so that the search includes the  
same row number that was previously
```

```
    ; found (in case adjacent rows are selected):
```

```
    RowNumber := LV_GetNext(RowNumber - 1)
```

```
    if not RowNumber ; The above returned zero,  
so there are no more selected rows.
```

```
        break
```

```
    LV_Delete(RowNumber) ; Clear the row from the  
ListView.
```

```
}
```

```
return
```

```
GuiSize: ; Expand or shrink the ListView in  
response to the user's resizing of the window.
```

```
if A_EventInfo = 1 ; The window has been  
minimized. No action needed.
```

```
    return
```

```
; Otherwise, the window has been resized or  
maximized. Resize the ListView to match.
```

```
GuiControl, Move, MyListView, % "W" . (A_GuiWidth  
- 20) . " H" . (A_GuiHeight - 40)
```

```
return
```

```
GuiClose: ; When the window is closed, exit the  
script automatically:
```

```
ExitApp
```


TreeView [v1.0.44+]

Table of Contents

- [Introduction and Simple Example](#)
- [Options and Styles](#)
- [Built-in Functions:](#)
 - [Add/modify/delete items](#)
 - [Getting data out of a TreeView](#)
- [G-Label Notifications](#)
- [Remarks](#)
- [Longer Example](#)

Introduction and Simple Example

A Tree-View displays a hierarchy of items by indenting child items beneath their parents. The most common example is Explorer's tree of drives and folders.

The syntax for creating a TreeView is:

```
Gui, Add, TreeView, Options
```

Here is a working script that creates and displays a simple hierarchy of items:

```
Gui, Add, TreeView
P1 := TV_Add("First parent")
P1C1 := TV_Add("Parent 1's first child", P1) ;
Specify P1 to be this item's parent.
P2 := TV_Add("Second parent")
P2C1 := TV_Add("Parent 2's first child", P2)
P2C2 := TV_Add("Parent 2's second child", P2)
P2C2C1 := TV_Add("Child 2's first child", P2C2)

Gui, Show ; Show the window and its TreeView.
return

GuiClose: ; Exit the script when the user
closes the TreeView's GUI window.
ExitApp
```

Options and Styles for "*Gui, Add, TreeView, Options*"

AltSubmit: Notifies the script for more types of TreeView events than normal. In other words, the g-label is launched more often. See [TreeView Notifications](#) for details.

Background: Specify the word Background followed immediately by a color name (see [color chart](#)) or RGB value (the 0x prefix is optional). Examples: BackgroundSilver , BackgroundFFDD99 . If this option is not present, the TreeView initially defaults to the background color set by the last parameter of [Gui Color](#) (or if none, the system's default background color). Specifying BackgroundDefault applies the system's default background color (usually white). For example, a TreeView can be restored to the default color via GuiControl, +BackgroundDefault, MyTreeView .

Buttons: Specify -Buttons (minus Buttons) to avoid displaying a plus or minus button to the left of each item that has children.

C: Text color. Specify the letter C followed immediately by a color name (see [color chart](#)) or RGB value (the 0x prefix is optional). Examples: cRed , cFF2211 , c0xFF2211 , cDefault .

Checked: Provides a checkbox at the left side of each item. When [adding](#) an item, specify the word *Check* in its options to have the box to start off checked instead of unchecked. The user may either click the checkbox or press the spacebar to check or uncheck an item. To discover which items in a TreeView are currently checked, call [TV_GetNext\(\)](#) or [TV_Get\(\)](#).

HScroll: Specify -HScroll (minus HScroll) to disable horizontal scrolling in the control (in addition, the control will not display any horizontal scroll bar).

ImageList: This is the means by which icons are added to a TreeView. Specify the word *ImageList* followed immediately by the ImageListID returned from a previous call to [IL_Create\(\)](#). This option has an effect only when creating a TreeView (however, [TV_SetImageList\(\)](#) does not have this limitation). Here is a working example:

```

ImageListID := IL_Create(10) ; Create an
ImageList with initial capacity for 10 icons.
Loop 10 ; Load the ImageList with some standard
system icons.
    IL_Add(ImageListID, "shell32.dll", A_Index)
Gui, Add, TreeView, ImageList%ImageListID%
TV_Add("Name of Item", 0, "Icon4") ; Add an
item to the TreeView and give it a folder icon.
Gui Show

```

Lines: Specify `-Lines` (minus Lines) to avoid displaying a network of lines connecting parent items to their children. However, removing these lines also prevents the plus/minus buttons from being shown for top-level items.

ReadOnly: Specify `-ReadOnly` (minus ReadOnly) to allow editing of the text/name of each item. To edit an item, select it then press the `F2`  (see the [WantF2](#) option below). Alternatively, you can click an item once to select it, wait at least half a second, then click the same item again to edit it. After being edited, an item can be alphabetically repositioned among its siblings via the following example:

```

Gui, Add, TreeView, -ReadOnly gMyTree
; ...
MyTree:
if (A_GuiEvent == "e") ; The user has finished
editing an item (use == for case sensitive
comparison).
    TV_Modify(TV_GetParent(A_EventInfo), "Sort")
; This works even if the item has no parent.
return

```

R: Rows of height (upon creation). Specify the letter R followed immediately by the number of rows for which to make room inside the control. For example, `R10` would make the control 10 items tall.

WantF2: Specify `-WantF2` (minus WantF2) to prevent an `F2`  stroke from [editing](#) the currently selected item. This setting is ignored unless `-ReadOnly` is also in effect. Regardless of this setting, the g-label still receives `F2`  [notifications](#).

(Unnamed numeric styles): Since styles other than the above are rarely used, they do not have names. See the [TreeView styles table](#) for a list.

Built-in Functions for TreeViews

All of the TreeView functions operate upon the current thread's [default GUI window](#) (which can be changed via `Gui, 2:Default`). If the default window does not exist or has no TreeView controls, all functions return zero to indicate the problem.

If the window has more than one TreeView control, by default the functions operate upon the one most recently added. To change this, specify `Gui, TreeView, TreeViewName`, where *TreeViewName* is the name of the TreeView's [associated variable](#), its ClassNN as shown by Window Spy or [in v1.1.04+] its HWND. Once changed, all existing and future [threads](#) will use the indicated TreeView. [v1.1.23+]: [A_DefaultTreeView](#) contains the current setting.

TV_SetImageList(ImageListID [, 0|2]) [v1.1.02+]

Sets or replaces the TreeView's [ImageList](#). *ImageListID* is the number returned from a previous call to [IL_Create\(\)](#). The second parameter is normally omitted, in which case it defaults to 0. Otherwise, specify 2 for state icons (which are not yet directly supported, but could be used via [SendMessage](#)). If successful, `TV_SetImageList()` returns the *ImageListID* that was previously associated with the TreeView (or 0 if none). Any such detached ImageList should normally be destroyed via [IL_Destroy\(ImageListID\)](#).

Add, Modify, and Delete Items

TV_Add(Name, [ParentItemID, Options])

Adds a new item to the TreeView and returns its unique Item ID number (or 0 upon failure). *Name* is the displayed text of the item, which can be text or numeric (including numeric [expression](#) results). *ParentItemID* is the ID number of the new item's parent (omit it or specify 0 to add the item at the top level). When adding a large number of items, performance can be improved by using `GuiControl, -Redraw, MyTreeView` before adding the items, and `GuiControl, +Redraw, MyTreeView` afterward.

Options for TV_Add() and TV_Modify()

The *Options* parameter is a string containing zero or more words from the list below (not case sensitive). Separate each word from the next with a space or tab. To remove an option, precede it with a minus sign. To add an option, a plus sign is permitted but not required.

Bold: Displays the item's name in a bold font. To later un-bold the item, use `TV_Modify(ItemID, "-Bold")`.

Check: Shows a checkmark to the left of the item (if the TreeView has [checkboxes](#)). To later uncheck it, use `TV_Modify(ItemID, "-Check")`. The word *Check* may optionally be followed immediately by a 0 or 1 to indicate the starting state. In other words, both `"Check"` and `"Check"` . `VarContainingOne` are the same (the period used here is the [concatenation operator](#)).

Expand: Expands the item to reveal its children (if any). To later collapse the item, use `TV_Modify(ItemID, "-Expand")`. If there are no children, `TV_Modify()` returns 0 instead of the item's ID. By contrast, `TV_Add()` marks the item as expanded in case children are added to it later. Unlike "Select" (below), expanding an item does not automatically expand its parent. Finally, the word *Expand* may optionally be followed immediately by a 0 or 1 to indicate the starting state. In other words, both `"Expand"` and `"Expand"` . `VarContainingOne` are the same.

First | Sort | N: These options apply only to [TV_Add\(\)](#). They specify the new item's position relative to its siblings (a *sibling* is any other item on the same level). If none of these options is present, the new item is added as the last/bottom sibling. Otherwise, specify *First* to add the item as the first/top sibling, or specify *Sort* to insert it among its siblings in alphabetical order. If a plain integer (**N**) is specified, it is assumed to be ID number of the sibling after which to insert the new item (if integer N is the only option present, it does not have to be enclosed in quotes).

Icon: Specify the word *Icon* followed immediately by the number of this item's icon, which is displayed to the left of the item's name. If this option is absent, the first icon in the [ImageList](#) is used. To display a blank icon, specify a number that is larger than the number of icons in the [ImageList](#). If the control lacks an [ImageList](#), no icon is displayed nor is any space reserved for one.

Select: Selects the item. Since only one item at a time can be selected, any previously selected item is automatically de-selected. In addition, this option reveals the newly selected item by expanding its parent(s), if necessary. To find out the current selection, call [TV_GetSelection\(\)](#).

Sort: For [TV_Modify\(\)](#), this option alphabetically sorts the children of the specified item. To instead sort all top-level items, use `TV_Modify(0, "Sort")`. If there are no children, 0 is returned instead of the ID of the modified item.

Vis: Ensures that the item is completely visible by scrolling the [TreeView](#) and/or expanding its parent, if necessary.

VisFirst: Same as above except that the [TreeView](#) is also scrolled so that the item appears at the top, if possible. This option is typically more effective when used with [TV_Modify\(\)](#) than with [TV_Add\(\)](#).

[TV_Modify\(ItemID \[, Options, NewName\]\)](#)

Modifies the attributes and/or name of an item. It returns the item's own ID upon success or 0 upon failure (or partial failure). When only the first parameter is present, the specified item is [selected](#). When *NewName* is omitted, the current name is left unchanged. For *Options*, see the list above.

TV_Delete([ItemID])

If *ItemID* is omitted, **all** items in the TreeView are deleted. Otherwise, only the specified *ItemID* is deleted. It returns 1 upon success and 0 upon failure.

Getting Data Out of a TreeView

TV_GetSelection()

Returns the selected item's ID number.

TV_GetCount()

Returns the total number of items in the control. This function is always instantaneous because the control keeps track of the count.

TV_GetParent(ItemID)

Returns the specified item's parent as an item ID. Items at the top level have no parent and thus return 0.

TV_GetChild(ParentItemID)

Returns the ID number of the specified item's first/top child (or 0 if none).

TV_GetPrev(ItemID)

Returns the ID number of the sibling above the specified item (or 0 if none).

TV_GetNext([ItemID, "Checked | Full"])

This has the following modes:

1. When all parameters are omitted, it returns the ID number of the first/top item in the TreeView (or 0 if none).
2. When the only first parameter (ItemID) is present, it returns the ID number of the sibling below the specified item (or 0 if none). If the first parameter is 0, it returns the ID number of the first/top item in the TreeView (or 0 if none).
3. When the second parameter is "Full" or "F", the next item is retrieved regardless of its relationship to the specified item. This allows the script to

easily traverse the entire tree, item by item. For example:

```
ItemID = 0 ; Causes the loop's first
iteration to start the search at the top of
the tree.
Loop
{
    ItemID := TV_GetNext(ItemID, "Full") ;
Replace "Full" with "Checked" to find all
checkmarked items.
    if not ItemID ; No more items in tree.
        break
    TV_GetText(ItemText, ItemID)
    MsgBox The next Item is %ItemID%, whose
text is "%ItemText%".
}
```

4. When the second parameter is either "Check", "Checked", or "C", the same behavior as above is used except that any item without a checkmark is skipped over. This allows all checkmarked items in the TreeView to be retrieved, one by one.

TV_GetText(OutputVar, ItemID)

Retrieves the text/name of the specified *ItemID* and stores it in *OutputVar*. If the text is longer than 8191, only the first 8191 characters are retrieved. Upon success, the function returns the item's own ID. Upon failure, it returns 0 (and *OutputVar* is also made blank).

TV_Get(ItemID, "Expand | Check | Bold")

If the specified item has the specified attribute, its own *ItemID* is returned. Otherwise, 0 is returned. For the second parameter, specify "E", "Expand", or "Expanded" to determine if the item is currently **expanded** (that is, its children are being displayed); specify "C", "Check", or "Checked" to determine if the item has a **checkmark**; or specify "B" or "Bold" to determine if the item is currently **bold** in font.

Tip: Since an IF-statement sees any non-zero value as "true", the following two lines are functionally identical:

1. `if TV_Get(ItemID, "Checked") = ItemID`
2. `if TV_Get(ItemID, "Checked")`

G-Label Notifications (Primary)

A **g-label** such as `gMySubroutine` may be listed in the control's options. This would cause the *MySubroutine* label to be launched automatically whenever the user performs an action in the control. This subroutine may consult the built-in variables `A_Gui` and `A_GuiControl` to find out which window and TreeView generated the event. More importantly, it may consult `A_GuiEvent`, which contains one of the following strings or letters (for compatibility with future versions, a script should not assume these are the only possible values):

DoubleClick: The user has double-clicked an item. The variable `A_EventInfo` contains the item ID.

D: The user has attempted to start dragging an item (there is currently no built-in support for this). The variable `A_EventInfo` contains the item ID.

d (lowercase D): Same as above except a right-click-drag rather than a left-drag.

e (lowercase E): The user has finished editing an item (the user may edit items only when the TreeView has `-ReadOnly` in its options). The variable `A_EventInfo` contains the item ID.

S: A new item has been selected, either by the user or the script itself. The variable `A_EventInfo` contains the newly selected item ID.

G-Label Notifications (Secondary)

If the TreeView has the word `AltSubmit` in its `options`, its g-label is launched more often and `A_GuiEvent` may contain the following additional values:

Normal: The user has left-clicked an item. The variable `A_EventInfo` contains the item ID.

RightClick: The user has right-clicked an item. The variable `A_EventInfo` contains the item ID. In most cases, it is best not to display a menu in response to this. Instead, use the `GuiContextMenu` label because it also recognizes the Apps key. For example:

```
GuiContextMenu: ; Launched in response to a
right-click or press of the Apps key.
if A_GuiControl <> MyTreeView ; This check is
optional. It displays the menu only for clicks
inside the TreeView.
    return
; Show the menu at the provided coordinates,
A_GuiX and A_GuiY. These should be used
; because they provide correct coordinates even
if the user pressed the Apps key:
Menu, MyContextMenu, Show, %A_GuiX%, %A_GuiY%
return
```

E: The user has begun editing an item (the user may edit items only when the TreeView has `-ReadOnly` in its options). The variable `A_EventInfo` contains the item ID.

F: The TreeView has received keyboard focus.

f (lowercase F): The TreeView has lost keyboard focus.

K: The user has pressed a key while the TreeView has focus. `A_EventInfo` contains the virtual key code of the key, which is a number between 1 and 255. If

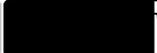
the key is alphabetic, on most keyboard layouts it can be translated to the corresponding character via `Chr(A_EventInfo)`. F2 strokes are received regardless of `WantF2`. However, the `Enter` stroke is not received; to receive it, use a default button as described [below](#).

+ (plus sign): An item has been expanded to reveal its children. The variable `A_EventInfo` contains the item ID.

- (minus sign): An item has been collapsed to hide its children. The variable `A_EventInfo` contains the item ID.

Remarks

The [Gui Submit](#) command has no effect on a TreeView control. Therefore, the script may use the TreeView's [associated variable](#) (if any) to store other data without concern that it will ever be overwritten.

To detect when the user has pressed   TreeView has focus, use a [default button](#) (which can be hidden if desired). For example:

```
Gui, Add, Button, Hidden Default, OK
...
ButtonOK:
GuiControlGet, FocusedControl, FocusV
if FocusedControl <> MyTreeView
    return
MsgBox % "Enter was pressed. The selected item
ID is " . TV_GetSelection()
return
```

In addition to navigating from item to item with the keyboard, the user may also perform incremental search by typing the first few characters of an item's name. This causes the selection to jump to the nearest matching item.

Although any length of text can be stored in each item of a TreeView, only the first 260 characters are displayed.

Although the theoretical maximum number of items in a TreeView is 65536, item-adding performance will noticeably decrease long before then. This can be alleviated somewhat by using the redraw tip described in [TV_Add\(\)](#).

Unlike [ListViews](#), a TreeView's ImageList is not automatically destroyed when the TreeView is destroyed. Therefore, a script should call [IL_Destroy\(ImageListID\)](#) after destroying a TreeView's window if the ImageList will not be used for anything else. However, this is not necessary if the script will soon be exiting because all ImageLists are automatically destroyed at that time.

A script may create more than one TreeView per window. To operate upon a TreeView other than the default one, see [built-in functions](#).

To perform actions such as resizing, hiding, or changing the font of a TreeView, use [GuiControl](#).

Tree View eXtension (TVX) extends TreeViews to support moving, inserting and deleting. It is demonstrated at www.autohotkey.com/forum/topic19021.html

Related

[ListView](#), [Other Control Types](#), [Gui](#), [GuiContextMenu](#), [GuiControl](#), [GuiControlGet](#), [TreeView styles table](#)

Example

```
; The following is a working script that is more
elaborate than the one near the top of this page.
; It creates and displays a TreeView containing
all folders in the all-users Start Menu. When the
; user selects a folder, its contents are shown in
a ListView to the right (like Windows Explorer).
; In addition, a StatusBar control shows
information about the currently selected folder.
```

```
; The following folder will be the root folder for
the TreeView. Note that loading might take a long
; time if an entire drive such as C:\ is
specified:
```

```
TreeRoot = %A_StartMenuCommon%
TreeViewWidth := 280
ListViewWidth := A_ScreenWidth - TreeViewWidth -
30
```

```
; Allow the user to maximize or drag-resize the
window:
```

```
Gui +Resize
```

```
; Create an ImageList and put some standard system
icons into it:
```

```
ImageListID := IL_Create(5)
```

```
Loop 5
```

```
    IL_Add(ImageListID, "shell32.dll", A_Index)
```

```
; Create a TreeView and a ListView side-by-side to
behave like Windows Explorer:
```

```
Gui, Add, TreeView, vMyTreeView r20
```

```
w%TreeViewWidth% gMyTreeView
```

```
ImageList%ImageListID%
```

```
Gui, Add, ListView, vMyListView r20
```

```

w%ListViewWidth% x+10, Name|Modified

; Set the ListView's column widths (this is
optional):
Col2Width = 70 ; Narrow to reveal only the
YYYYMMDD part.
LV_ModifyCol(1, ListViewWidth - Col2Width - 30) ;
Allows room for vertical scrollbar.
LV_ModifyCol(2, Col2Width)

; Create a Status Bar to give info about the
number of files and their total size:
Gui, Add, StatusBar
SB_SetParts(60, 85) ; Create three parts in the
bar (the third part fills all the remaining
width).

; Add folders and their subfolders to the tree.
Display the status in case loading takes a long
time:
SplashTextOn, 200, 25, TreeView and StatusBar
Example, Loading the tree...
AddSubFoldersToTree(TreeRoot)
SplashTextOff

; Display the window and return. The OS will
notify the script whenever the user performs an
eligible action:
Gui, Show,, %TreeRoot% ; Display the source
directory (TreeRoot) in the title bar.
return

AddSubFoldersToTree(Folder, ParentItemID = 0)
{
    ; This function adds to the TreeView all
subfolders in the specified folder.
    ; It also calls itself recursively to gather

```

nested folders to any depth.

```
    Loop %Folder%\*.*, 2 ; Retrieve all of
Folder's sub-folders.
```

```
        AddSubFoldersToTree(A_LoopFileFullPath,
TV_Add(A_LoopFileName, ParentItemID, "Icon4"))
    }
```

MyTreeView: ; This subroutine handles user actions (such as clicking).

```
if A_GuiEvent <> S ; i.e. an event other than
"select new tree item".
```

```
    return ; Do nothing.
```

```
    ; Otherwise, populate the ListView with the
contents of the selected folder.
```

```
    ; First determine the full path of the selected
folder:
```

```
TV_GetText(SelectedItemText, A_EventInfo)
```

```
ParentID := A_EventInfo
```

```
Loop ; Build the full path to the selected
folder.
```

```
{
```

```
    ParentID := TV_GetParent(ParentID)
```

```
    if not ParentID ; No more ancestors.
```

```
        break
```

```
    TV_GetText(ParentText, ParentID)
```

```
    SelectedItemText =
```

```
%ParentText%\%SelectedItemText%
```

```
}
```

```
SelectedFullPath = %TreeRoot%\%SelectedItemText%
```

```
    ; Put the files into the ListView:
```

```
LV_Delete() ; Clear all rows.
```

```
GuiControl, -Redraw, MyListView ; Improve
performance by disabling redrawing during load.
```

```
FileCount = 0 ; Init prior to loop below.
```

```
TotalSize = 0
```

```
Loop %SelectedFullPath%\*.* ; For simplicity,
```

this omits folders so that only files are shown in the ListView.

```
{
    LV_Add("", A_LoopFileName,
A_LoopFileTimeModified)
    FileCount += 1
    TotalSize += A_LoopFileSize
}
GuiControl, +Redraw, MyListView
```

; Update the three parts of the status bar to show info about the currently selected folder:

```
SB_SetText(FileCount . " files", 1)
SB_SetText(Round(TotalSize / 1024, 1) . " KB", 2)
SB_SetText(SelectedFullPath, 3)
return
```

GuiSize: ; Expand/shrink the ListView and TreeView in response to user's resizing of window.
if A_EventInfo = 1 ; The window has been minimized. No action needed.

```
return
```

; Otherwise, the window has been resized or maximized. Resize the controls to match.

```
GuiControl, Move, MyTreeView, % "H" . (A_GuiHeight - 30) ; -30 for StatusBar and margins.
```

```
GuiControl, Move, MyListView, % "H" . (A_GuiHeight - 30) . " W" . (A_GuiWidth - TreeViewWidth - 30)
```

```
return
```

GuiClose: ; Exit the script when the user closes the TreeView's GUI window.

```
ExitApp
```

Image Handles [v1.1.23+]

To use an icon or bitmap handle in place of an image filename, use the following syntax:

```
HBITMAP:bitmap-handle
```

```
HICON:icon-handle
```

Replace *bitmap-handle* or *icon-handle* with the actual handle value. For example, `hicon:%handle%` (or `"hicon:" handle` in an [expression](#)), where *handle* is a variable containing an icon handle.

The following commands support this syntax:

- [Gui Add, Picture](#) (and [GuiControl](#) when setting a Picture control's content).
- [IL_Add](#)
- [LoadPicture](#)
- [SB_SetIcon](#)
- [ImageSearch](#)
- [Menu Tray, Icon](#) or [Menu MenuName, Icon](#)
- [SplashImage](#)

A bitmap or icon handle is a numeric value which identifies a bitmap or icon in memory. The majority of scripts never need to deal with handles, as in most cases AutoHotkey takes care of loading the image from file and freeing it when it is no longer needed. The syntax shown above is intended for use when the script obtains an icon or bitmap handle from another source, such as by sending the WM_GETICON message to a window. It can also be used in combination with [LoadPicture](#) to avoid loading an image from file multiple times.

By default, AutoHotkey treats the handle as though it loaded the image from file - for example, a bitmap used on a Picture control is deleted when the GUI is destroyed, and an image will generally be deleted immediately if it needs to be resized. To avoid this, put an asterisk between the colon and handle. For example: `hbitmap: *%handle%` (or `"hbitmap: *" handle` in an [expression](#)). With the exception of [ImageSearch](#), this forces the command to take

a copy of the image.

Examples

```
; Show a menu of the first n files matching a
pattern, and their icons.
pattern = %A_ScriptDir%\*
n = 15

; Allocate memory for a SHFILEINFOW struct.
VarSetCapacity(fileinfo, fsize := A_PtrSize +
688)

Loop, Files, %pattern%, FD
{
    ; Add a menu item for each file.
    Menu F, Add, %A_LoopFileName%, donothing

    ; Get the file's icon.
    if DllCall("shell32\SHGetFileInfow", "WStr",
A_LoopFileFullPath
        , "UInt", 0, "Ptr", &fileinfo, "UInt",
fsize, "UInt", 0x100)
    {
        hicon := NumGet(fileinfo, 0, "Ptr")
        ; Set the menu item's icon.
        Menu F, Icon, %A_Index%&, HICON:%hicon%
        ; Because we used ":" and not ":*", the
icon will be automatically
        ; freed when the program exits or if the
menu or item is deleted.
    }
}
until A_Index = n
Menu F, Show
donothing:
```

return

See also: [LoadPicture](#).

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. If *Prompt* is long, it can be broken up into several shorter lines by means of a [continuation section](#), which might improve readability and maintainability.

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. This parameter can be an [expression](#).

Height

If this parameter is blank or omitted, the starting height of the window will be 189. This parameter can be an [expression](#).

X, Y

The X and Y coordinates of the window (use 0,0 to move it to the upper left corner of the desktop), which can be [expressions](#). 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

Timeout in seconds (can contain a decimal point or be an [expression](#)). 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. *OutputVar* will still be set to what the user entered.

Default

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

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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.

[ErrorLevel](#) is set to 1 if the user presses the CANCEL button, 0 if the user presses OK, or 2 if the dialog times out. In all three cases, *OutputVar* is set to the value entered. This allows the CANCEL button to perform a function other than CANCEL should the script designer wish it.

A GUI window may display a modal InputBox by means of [Gui +OwnDialogs](#). A modal InputBox prevents the user from interacting with the GUI window until the InputBox is dismissed.

Related

[GUI](#), [Input](#), [MsgBox](#), [FileSelectFile](#), [FileSelectFolder](#), [SplashTextOn](#), [ToolTip](#)

Example

```
InputBox, password, Enter Password, (your input  
will be hidden), hide  
InputBox, UserInput, Phone Number, Please enter a  
phone number., , 640, 480  
if ErrorLevel  
    MsgBox, CANCEL was pressed.  
else  
    MsgBox, You entered "%UserInput%"
```

LoadPicture [v1.1.23+]

Loads a picture from file and returns a bitmap or icon handle.

```
Handle := LoadPicture(Filename [, Options, ByRef  
ImageType])
```

Parameters

Filename

The filename of the picture, which is usually assumed to be in [A_WorkingDir](#) if an absolute path isn't specified. If the name of a DLL or EXE file is given without a path, it may be loaded from the directory of the current executable (AutoHotkey.exe or a compiled script) or a system directory.

Options

A string of zero or more of the following options, with each separated from the last by a space or tab:

Wn and **Hn**: The width and height to load the image at, where *n* is an integer. If one dimension is omitted or -1, it is calculated automatically based on the other dimension, preserving aspect ratio. If both are omitted, the image's original size is used. If either dimension is 0, the original size is used for that dimension. For example: "w80 h50", "w48 h-1" or "w48" (preserve aspect ratio), "h0 w100" (use original height but override width).

Iconn: Indicates which icon to load from a file with multiple icons (generally an EXE or DLL file). For example, "Icon2" loads the file's second icon. [v1.1.27+]: Any supported image format can be converted to an icon by specifying "Icon1". However, the icon is converted back to a bitmap if the *ImageType* parameter is omitted.

GDI+: Use GDI+ to load the image, if available. For example, "GDI+

w100" .

ImageType

If a variable is specified, it is assigned a number indicating the type of handle being returned: 0 (IMAGE_BITMAP), 1 (IMAGE_ICON) or 2 (IMAGE_CURSOR).

If this parameter is omitted or not a variable, the return value is always a bitmap handle (icons/cursors are converted if necessary). This is because reliably using or deleting an icon/cursor/bitmap handle requires knowing which type it is.

Remarks

LoadPicture also supports [the handle syntax](#), such as for creating a resized image based on an icon or bitmap which has already been loaded into memory, or converting an icon to a bitmap by omitting *ImageType*.

If the image needs to be freed from memory, call whichever function is appropriate for the type of handle.

```
if (not ImageType) ; IMAGE_BITMAP (0) or the
ImageType parameter was omitted.
    DllCall("DeleteObject", "ptr", Handle)
else if (ImageType = 1) ; IMAGE_ICON
    DllCall("DestroyIcon", "ptr", Handle)
else if (ImageType = 2) ; IMAGE_CURSOR
    DllCall("DestroyCursor", "ptr", Handle)
```

Related

[Image Handles](#)

Example

```
; Pre-load and reuse some images.
```

```

Pics := []
; Find some pictures to display.
Loop, Files, %A_WinDir%\Web\Wallpaper\*.jpg, R
{
    ; Load each picture and add it to the array.
    Pics.Push(LoadPicture(A_LoopFileFullPath))
}
if !Pics.Length()
{
    ; If this happens, edit the path on the Loop
line above.
    MsgBox, No pictures found! Try a different
directory.
    ExitApp
}
; Add the picture control, preserving the aspect
ratio of the first picture.
Gui, Add, Pic, w600 h-1 vPic +Border, %
"HBITMAP:*" Pics.1
Gui, Show
Loop
{
    ; Switch pictures!
    GuiControl, , Pic, % "HBITMAP:*"
Pics[Mod(A_Index, Pics.Length()+1)]
    Sleep 3000
}
return
GuiClose:
GuiEscape:
ExitApp

```

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.

```
Menu, MenuName, Cmd [, P3, P4, P5]
```

Parameters

MenuName

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. For 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.

MenuItemName

The name or position of a menu item. Some common rules apply to this parameter across all sub-commands which use it:

To underline one of the letters in a menu item's name, precede that letter with an ampersand (&). When the menu is displayed, such an item 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`

When referring to an existing menu or menu item, the name is not case sensitive but any ampersands must be included. For example: `&Open`

[v1.1.23+]: To identify an existing item by its position in the menu, write the item's position followed by an ampersand. For example, **1&** indicates the first item.

Add or Change Items in a Menu

Add [, MenuItemName, Label-or-Submenu, Options]: 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* does not yet exist, it will be added to the menu. Otherwise, *MenuItemName* is updated with the newly specified *Label-or-Submenu*.

To add a menu separator line, omit all three parameters.

The label subroutine is run as a new [thread](#) when the user selects the menu item (similar to [Gosub](#) and [hotkey subroutines](#)). If *Label-or-Submenu* is omitted, *MenuItemName* will be used as both the label and the menu item's name.

[v1.1.20+]: If it is not the name of an existing label, *Label-or-Submenu* can be the name of a function, or a single variable reference containing a [function object](#). For example, `%FuncObj%` or `% FuncObj`. Other expressions which return objects are currently unsupported. The function can optionally define parameters as shown below:

```
FunctionName(ItemName, ItemPos, MenuName)
```

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. For example:

```
Menu, MySubmenu, add, Item1
Menu, tray, add, This Menu Item Is A Submenu,
:MySubmenu
```

If not omitted, *Options* must be a space- or tab-delimited list of one or more of the following options:

Replace <i>n</i> with the menu item's thread priority , e.g. <code>P1</code> . If this option is omitted when adding a menu item, the priority will be
--

<i>Pn</i>	0, which is the standard default. If omitted when updating a menu item, the item's priority will not be changed. Use a decimal (not hexadecimal) number as the priority.
+Radio	[v1.1.23+]: If the item is checked, a bullet point is used instead of a check mark.
+Right	[v1.1.23+]: The item is right-justified within the menu bar. This only applies to menu bars , not popup menus or submenus.
+Break	[v1.1.23+]: The item begins a new column in a popup menu.
+BarBreak	[v1.1.23+]: As above, but with a dividing line between columns.

The plus sign (+) is optional and can be replaced with minus (-) to remove the option, as in `-Radio`. Options are not case sensitive.

To change an existing item's options without affecting its label or submenu, simply omit the *Label-or-Submenu* parameter.

Insert [*ItemToInsertBefore*, *NewItemName*, *Label-or-Submenu*, *Options*]

[v1.1.23+]: Inserts a new item before the specified item. Usage is identical to *Add* (above), except for the additional *ItemToInsertBefore* parameter, which is the name of an existing item or a position between 1 and [the current number of items](#) plus 1. Items can also be appended by omitting *ItemToInsertBefore* (by writing two consecutive commas). Unlike *Add*, *Insert* creates a new item even if *NewItemName* matches the name of an existing item.

Delete [*MenuItemName*]: Deletes *MenuItemName* from the menu. Standard menu items such as Exit (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. Deleting a menu also causes the current [Win32 menu](#) of its parent and submenus to be destroyed, to be recreated later as needed.

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]: Renames *MenuItemName* to *NewName* (if *NewName* is blank, *MenuItemName* will be converted into a separator line). The menu item's current target label or submenu is unchanged. [v1.1.23+]: A separator line can be converted to a normal item by specifying the position& of the separator and a non-blank *NewName*, and then using the *Add* command to give the item a label or submenu.

Check, MenuItemName: Adds a visible checkmark in the menu next to *MenuItemName* (if there isn't one already).

Uncheck, MenuItemName: Removes the checkmark (if there is one) from *MenuItemName*.

ToggleCheck, MenuItemName: Adds a checkmark if there wasn't one; otherwise, removes it.

Enable, MenuItemName: Allows the user to once again select *MenuItemName* if it 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 TRAY is currently purely cosmetic). 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 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).

Set or Remove a Menu Item's Icon [AHK_L 17+]

Icon, MenuItemName, FileName [, IconNumber, IconWidth]: Sets *MenuItemName*'s icon. *FileName* can either be an icon file or any image in a format supported by AutoHotkey. To use an icon group other than the first one in the file, specify its number for *IconNumber* (if omitted, it defaults to 1). If *IconNumber* is negative, its absolute value is assumed to be the resource ID of an icon within an executable file. Specify the desired width of the icon in *IconWidth*. If the icon group indicated by *IconNumber* contains multiple icon sizes, the closest match is used and the icon is scaled to the specified size. See the Examples section for usage examples.

Currently it is necessary to specify "actual size" when setting the icon to preserve transparency on Windows Vista and later. For example:

```
Menu, MenuName, Icon, MenuItemName,  
Filename.png,, 0
```

Known limitation: Icons on Gui menu bars are positioned incorrectly on Windows XP and older.

[v1.1.23+]: A [bitmap or icon handle](#) can be used instead of a filename. For example, `HBITMAP:%handle%`.

NoIcon, MenuItemName: Removes *MenuItemName*'s icon, if any.

Change the Tray Icon or ToolTip (MenuName must be TRAY)

Icon [*FileName*, *IconNumber*, 1]: Changes the script's [tray icon](#) to one of the ones from *FileName*. The following types of files are supported: ICO, CUR, ANI, EXE, DLL, CPL, SCR, and other types that contain icon resources. To use an icon group other than the first one in the file, specify its number for *IconNumber* (if omitted, it defaults to 1). For example, 2 would load the default icon from the second icon group. If *IconNumber* is negative, its absolute value is assumed to be the resource ID of an icon within an executable file. Specify an asterisk (*) for *FileName* to restore the script to its default icon.

The last parameter: Specify 1 for the last parameter to freeze the icon, or 0 to unfreeze it (or leave it blank to keep the frozen/unfrozen state unchanged). When the icon has been frozen, [Pause](#) and [Suspend](#) will not change it. Note: To freeze or unfreeze the *current* icon, use 1 or 0 as in the following example: `Menu, Tray, Icon, , , 1`.

Changing the tray icon also changes the icon displayed by [InputBox](#), [Progress](#), and subsequently-created [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` (with no parameters).

Slight distortion may occur when loading tray icons from file types other than .ICO. This is especially true for 16x16 icons. To prevent this, store the desired tray icon inside a .ICO file.

There are some icons built into the operating system's DLLs and CPLs that might be useful. For example: `Menu, Tray, Icon, Shell32.dll, 174`.

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).

[v1.1.23+]: An [icon handle](#) can be used instead of a filename. For example, `Menu Tray, Icon, HICON: *%handle%`. The asterisk is required as the

icon must be "loaded" twice: once for the small icon and again for the large icon.

[v1.1.27+]: Non-icon image files and [bitmap handles](#) are supported for *Filename*. For example, `Menu Tray, Icon, HBITMAP: *%handle%`.

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]: Changes the tray icon's tooltip -- which is displayed when the mouse hovers over it. To create a multi-line tooltip, use the linefeed character (`\n`) in between each line, e.g. `Line1\nLine2`. Only the first 127 characters of *Text* are displayed, and *Text* is truncated at the first tab character, if present. 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).

Miscellaneous Commands

Show [, X, Y]: 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.

Color, ColorValue [, Single]: 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.

Click, ClickCount: 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). For example: `Menu, Tray, Click, 1`.

MainWindow: 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", which allows viewing of the script's source code and other info. *MenuName* must be TRAY.

NoMainWindow (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]: 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, Tray, 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*.

Win32 Menus

As items are added to a menu or modified, the name and other properties of each item are recorded by the Menu command, but the actual [Win32 menu](#) is not constructed immediately. This occurs when the menu or its parent menu is attached to a GUI or shown, either for the first time or if the menu has been "destroyed" since it was last shown. Any of the following can cause this Win32 menu to be destroyed, along with any parent menus and submenus:

- Deleting a menu.
- Replacing an item's submenu with a label or a different menu.
- Prior to [v1.1.27.03], calling **NoStandard** (if the standard items were present) or **DeleteAll**.

Any modifications which are made to the menu directly by Win32 API calls only apply to the current "instance" of the menu, and are lost when the menu is destroyed.

Each menu item is assigned an ID when it is first added to the menu. Scripts cannot rely on an item receiving a particular ID, but can retrieve the ID of an item by using `GetMenuItemID` as shown in the [MenuGetHandle example](#). This ID cannot be used with the Menu command, but can be used with various [Win32 functions](#).

Remarks

The names of menus and menu items can be up to 260 characters long.

Separator lines can be added to the menu by using `Menu, MenuName, Add` (i.e. omit all other parameters). [v1.1.23+]: To delete separator lines individually, identify them by their position in the menu. For example, use `Menu, MenuName, Delete, 3&` if there are two items preceding the separator. Alternatively, use `Menu, MenuName, DeleteAll` and then re-add your custom menu items.

New 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`.

The standard menu items such as "Pause Script" and "Suspend Hotkeys" cannot be individually operated upon by any menu sub-command.

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 (in such cases it is usually better to use [#NoTrayIcon](#)).

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. To instead buffer such events until later, use [Critical](#) as the subroutine's first line (however, this will also buffer/defer other threads such as the press of a hotkey).

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

The built-in variables [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 that label refer to the above variables to determine what action to take.

To keep a non-hotkey, non-[GUI](#) script running -- such as one that contains only custom menus or menu items -- use [#Persistent](#).

Related

[GUI](#), [Threads](#), [Thread](#), [Critical](#), [#NoTrayIcon](#), [Gosub](#), [Return](#), [SetTimer](#),
[#Persistent](#)

Examples

```
; EXAMPLE #1: This is a working script that adds a  
new menu item to the bottom of the tray icon menu.
```

```
#Persistent ; Keep the script running until the  
user exits it.
```

```
Menu, tray, add ; Creates a separator line.
```

```
Menu, tray, add, Item1, MenuHandler ; Creates a  
new menu item.
```

```
return
```

```
MenuHandler:
```

```
MsgBox You selected %A_ThisMenuItem% from menu  
%A_ThisMenu%.
```

```
return
```

```
; EXAMPLE #2: This is a working script that  
creates a popup menu that is displayed when the  
user presses the Win-Z hotkey.
```

```
; Create the popup menu by adding some items to  
it.
```

```
Menu, MyMenu, Add, Item1, MenuHandler
```

```
Menu, MyMenu, Add, Item2, MenuHandler
```

```
Menu, MyMenu, Add ; Add a separator line.
```

```
; Create another menu destined to become a submenu  
of the above menu.
```

```
Menu, Submenu1, Add, Item1, MenuHandler
```

```
Menu, Submenu1, Add, Item2, MenuHandler
```

```
; Create a submenu in the first menu (a right-  
arrow indicator). When the user selects it, the  
second menu is displayed.
```

```
Menu, MyMenu, Add, My Submenu, :Submenu1
```

```
Menu, MyMenu, Add ; Add a separator line below  
the submenu.
```

```
Menu, MyMenu, Add, Item3, MenuHandler ; Add  
another menu item beneath the submenu.
```

```
return ; End of script's auto-execute section.
```

```
MenuHandler:
```

```
MsgBox You selected %A_ThisMenuItem% from the menu  
%A_ThisMenu%.
```

```
return
```

```
#z::Menu, MyMenu, Show ; i.e. press the Win-Z  
hotkey to show the menu.
```

```
; EXAMPLE #3: This is a working script that  
demonstrates 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
```

; EXAMPLE #4: This is a working script that adds icons to its menu items.

```
Menu, FileMenu, Add, Script Icon, MenuHandler
Menu, FileMenu, Add, Suspend Icon, MenuHandler
Menu, FileMenu, Add, Pause Icon, MenuHandler
```

```
Menu, FileMenu, Icon, Script Icon, %A_AhkPath%, 2
;Use the 2nd icon group from the file
Menu, FileMenu, Icon, Suspend Icon, %A_AhkPath%,
-206 ;Use icon with resource identifier 206
Menu, FileMenu, Icon, Pause Icon, %A_AhkPath%,
-207 ;Use icon with resource identifier 207
Menu, MyMenuBar, Add, &File, :FileMenu
Gui, Menu, MyMenuBar
Gui, Add, Button, gExit, Exit This Example
Gui, Show
MenuHandler:
Return

Exit:
ExitApp
```

MenuGetHandle [v1.1.23+]

Retrieves the [Win32 menu](#) handle of a menu.

```
Handle := MenuGetHandle(MenuName)
```

Parameters

MenuName

The name of an existing menu. Menu names are not case sensitive.

Remarks

The returned handle is valid only until the Win32 menu is destroyed. Once the menu is destroyed, the operating system may reassign the handle value to any menus subsequently created by the script or any other program. Conditions which can cause the menu to be destroyed are listed under [Win32 Menus](#).

Related

[Menu](#), [MenuGetName](#)

Examples

```
Menu MyMenu, Add, Item 1, no
Menu MyMenu, Add, Item 2, no
Menu MyMenu, Add, Item B, no

; Retrieve the number of items in a menu.
item_count := DllCall("GetMenuItemCount", "ptr",
MenuGetHandle("MyMenu"))

; Retrieve the ID of the last item.
last_id := DllCall("GetMenuItemID", "ptr",
```

```
MenuGetHandle("MyMenu"), "int", item_count-1)
```

```
MsgBox, MyMenu has %item_count% items, and its  
last item has ID %last_id%.
```

```
no:
```

```
return
```

MenuGetName [v1.1.23+]

Retrieves the name of a menu given a handle to its underlying [Win32 menu](#).

```
MenuName := MenuGetName(Handle)
```

Parameters

Handle

▮ A Win32 menu handle (HMENU).

Remarks

Only the menu's current handle is recognized. When the menu's underlying Win32 menu is destroyed and recreated, the handle changes. For details, see [Win32 Menus](#).

Related

[Menu](#), [MenuGetHandle](#)

MsgBox

Displays the specified text in a small window containing one or more buttons (such as Yes and No).

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 linefeed 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`.

If *Text* is long, it can be broken up into several shorter lines by means of a [continuation section](#), which might improve readability and maintainability.

Options

Indicates the type of message box and the possible button combinations. If blank or omitted, it defaults to 0. See the tables below for allowed values.

This parameter must be either a literal number or [in v1.1.06+] a forced expression such as `% Options`. Any other non-blank value will not be recognized as this parameter, but instead as part of *Text* in the single-parameter mode.

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, which can contain a decimal point but is not an expression by default. [v1.1.06+]: This can be a forced expression such as `% mins*60`.

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`.

Known limitation: If the `MsgBox` contains only an OK button, `IfMsgBox` will think that the OK button was pressed if the `MsgBox` times out while its own `thread` is interrupted by another.

Values for the *Options* parameter

The *Options* parameter can be a combination (sum) of values from the following groups.

Group #1: Buttons

To indicate the buttons displayed in the message box, add **one** of the following values:

Function	Decimal Value	Hex Value
OK (that is, only an OK button is displayed)	0	0x0
OK/Cancel	1	0x1
Abort/Retry/Ignore	2	0x2
Yes/No/Cancel	3	0x3
Yes/No	4	0x4
Retry/Cancel	5	0x5
Cancel/Try Again/Continue	6	0x6

Group #2: Icon

To display an icon in the message box, add **one** of the following values:

Function	Decimal Value	Hex Value
Icon Hand (stop/error)	16	0x10
Icon Question	32	0x20
Icon Exclamation	48	0x30
Icon Asterisk (info)	64	0x40

Group #3: Default Button

To indicate the default button, add **one** of the following values:

Function	Decimal Value	Hex Value
Makes the 2nd button the default	256	0x100
Makes the 3rd button the default	512	0x200
Makes the 4th button the default (requires the Help button to be present)	768	0x300

Group #4: Modality

To indicate the modality of the dialog box, add **one** of the following values:

Function	Decimal Value	Hex Value
System Modal (always on top)	4096	0x1000
Task Modal	8192	0x2000
Always-on-top (style WS_EX_TOPMOST) (like System Modal but omits title bar icon)	262144	0x40000

Group #5: Other Options

To specify other options, add **one or more** of the following values:

Function	Decimal Value	Hex Value
Adds a Help button (see remarks below)	16384	0x4000
Make the text right-justified	524288	0x80000
Right-to-left reading order for Hebrew/Arabic	1048576	0x100000

Remarks

The tables above are used by adding up the values you wish to be present in the MsgBox. For example, to specify a Yes/No box with the default button being No instead of Yes, the *Options* value would be 256+4 (260). In hex, it would be 0x100+0x4 (0x104).

MsgBox has smart comma handling, so it is usually not necessary to [escape](#) commas in the *Text* parameter.

To determine which button the user pressed in the most recent MsgBox, use the [IfMsgBox](#) command. For example:

```
MsgBox, 4,, Would you like to continue? (press  
Yes or No)  
IfMsgBox Yes  
    MsgBox You pressed Yes.  
else  
    MsgBox You pressed No.
```

The names of the buttons can be customized by following [this example](#).

Tip: Pressing **Control**   a MsgBox window is active will copy its text to the clipboard. This applies to all MsgBoxes, not just those produced by AutoHotkey.

Using MsgBox with GUI windows: A GUI window may display a *modal* MsgBox by means of [Gui +OwnDialogs](#). A *modal* MsgBox prevents the user

from interacting with the GUI window until the MsgBox is dismissed. In such a case, it is not necessary to specify the System Modal or Task Modal options from the table above.

When [Gui +OwnDialogs](#) is *not* in effect, the Task Modal option (8192) can be used to disable all the script's windows until the user dismisses the MsgBox.

The Help button: When the Help button option (16384) is present in *Options*, pressing the Help button will have no effect unless both of the following are true:

1. The MsgBox is owned by a GUI window by means of [Gui +OwnDialogs](#).
2. The script is monitoring the WM_HELP message (0x53). For example: `OnMessage(0x53, "WM_HELP")`. When the WM_HELP() function is called, it may guide the user by means such as showing another window or MsgBox.

The Close button (in MsgBox's title bar): Since the MsgBox window is a built-in feature of the operating system, its **X** button is enabled only when certain buttons are present. 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.

Related

[IfMsgBox](#), [InputBox](#), [FileSelectFile](#), [FileSelectFolder](#), [ToolTip](#), [GUI](#)

Example

```
MsgBox This is the 1-parameter method. Commas (,)
do not need to be escaped.
MsgBox, 4, , This is the 3-parameter method.
Commas (,) do not need to be escaped.
MsgBox, 4, , Do you want to continue? (Press YES
or NO)
IfMsgBox No
    return
MsgBox, 4, , 4-parameter method: this MsgBox will
```

```
time out in 5 seconds. Continue?, 5
IfMsgBox Timeout
    MsgBox You didn't press YES or NO within the
5-second period.
else IfMsgBox No
    return
```

```
; By preceding any parameter with "% ", it becomes
an expression. In the following example,
; math is performed, a pseudo-array element is
accessed, and a function is called. All these
; items are concatenated via the "." operator to
form a single string displayed by MsgBox:
MsgBox % "New width for object #" . A_Index . "
is: " . RestrictWidth(ObjectWidth%A_Index% *
ScalingFactor)
```

```
; The following example alerts the user that a
MsgBox is going to steal focus (in case the user
is typing).
SplashTextOn,,, A MsgBox is about to appear.
Sleep 3000
SplashTextOff
MsgBox The backup process has completed.
```

OnMessage()

Specifies a [function](#) or [function object](#) to call automatically when the script receives the specified message.

```
OnMessage(MsgNumber [, Function, MaxThreads])
```

Parameters

MsgNumber

The number of the message to monitor or query, which should be between 0 and 4294967295 (0xFFFFFFFF). If you do not wish to monitor a [system message](#) (that is, one below 0x400), it is best to choose a number greater than 4096 (0x1000) to the extent you have a choice. This reduces the chance of interfering with messages used internally by current and future versions of AutoHotkey.

Function

A [function](#)'s name or, in [v1.1.20+], a [function object](#). To pass a literal function name, it must be enclosed in quotes.

How the function is registered and the return value of OnMessage depend on whether this parameter is a string or a function object. See [Function Name vs Object](#) for details.

MaxThreads [v1.0.47+]

This integer is normally omitted, in which case the monitor function is limited to one [thread](#) at a time. This is usually best because otherwise, the script would process messages out of chronological order whenever the monitor function interrupts itself. Therefore, as an alternative to *MaxThreads*, consider using *Critical* as described [below](#).

[v1.1.20+]: By default, when multiple functions are registered for a single *MsgNumber*, they are called in the order that they were registered. To register a function to be called before any previously registered functions,

specify a negative value for *MaxThreads*. For example, `OnMessage(Msg, Fn, -2)` registers `Fn` to be called before any other functions previously registered for *Msg*, and allows *Fn* a maximum of 2 threads. However, if the function is already registered, the order will not change unless it is unregistered and then re-registered.

Function Name vs Object

`OnMessage`'s return value and behavior depends on whether the *Function* parameter is a function name or an object.

Function Name

For backward compatibility, at most one function can be registered by name to monitor each unique *MsgNumber* -- this is referred to as the "legacy" monitor.

When the legacy monitor is first registered, whether it is called before or after previously registered monitors depends on the *MaxThreads* parameter. Updating the monitor to call a different function does not affect the order unless the monitor is unregistered first.

This registers or updates the current legacy monitor for *MsgNumber* (omit the quote marks if passing a variable):

```
Name := OnMessage(MsgNumber, "FunctionName")
```

The return value is one of the following:

- An empty string on [failure](#).
- The name of the previous function, if there was one.
- Otherwise, the name of the new function.

This unregisters the current legacy monitor for *MsgNumber* (if any) and returns its name (blank if none):

```
Name := OnMessage(MsgNumber, "")
```

This returns the name of the current legacy monitor for *MsgNumber* (blank if none):

```
Name := OnMessage(MsgNumber)
```

Function Object

Any number of [function objects](#) (including [normal functions](#)) can monitor a given *MsgNumber*.

Either of these two lines registers a function object to be called **after** any previously registered functions:

```
OnMessage(MsgNumber, FuncObj)      ; Option 1  
OnMessage(MsgNumber, FuncObj, 1)   ; Option 2  
(MaxThreads = 1)
```

This registers a function object to be called **before** any previously registered functions:

```
OnMessage(MsgNumber, FuncObj, -1)
```

To unregister a function object, specify 0 for *MaxThreads*:

```
OnMessage(MsgNumber, FuncObj, 0)
```

Failure

Failure occurs when *Function*:

1. is not an object, the name of a user-defined function, or an empty string;
2. is known to require more than four parameters; or

3. in [v1.0.48.05] or older, has any [ByRef](#) or [optional](#) parameters.

In [v1.1.19.03] or older, failure also occurs if the script attempts to monitor a new message when there are already 500 messages being monitored.

If *Function* is an object, an exception is thrown on failure. Otherwise, an empty string is returned.

The Function's Parameters

A [function](#) assigned to monitor one or more messages can accept up to four parameters:

```
MyMessageMonitor(wParam, lParam, msg, hwnd)
{
    ... body of function...
}
```

Although the names you give the parameters do not matter, the following information is sequentially assigned to them:

Parameter #1: The message's WPARAM value.

Parameter #2: The message's LPARAM value.

Parameter #3: The message number, which is useful in cases where a function monitors more than one message.

Parameter #4: The HWND (unique ID) of the window or control to which the message was sent. The HWND can be used with [ahk_id](#).

WPARAM and LPARAM are unsigned 32-bit integers (from 0 to $2^{32}-1$) or signed 64-bit integers (from -2^{63} to $2^{63}-1$) depending on whether the exe running the script is 32-bit or 64-bit. For 32-bit scripts, if an incoming parameter is intended to be a signed integer, any negative numbers can be revealed by following this example:

```
if (A_PtrSize = 4 && wParam > 0x7FFFFFFF) ;
    Checking A_PtrSize ensures the script is 32-bit.
    wParam := -(~wParam) - 1
```

You can omit one or more parameters from the end of the list if the corresponding information is not needed. For example, a function defined as `MyMsgMonitor(wParam, lParam)` would receive only the first two parameters, and one defined as `MyMsgMonitor()` would receive none of them.

Additional Information Available to the Function

In addition to the parameters received above, the function may also consult the values in the following built-in variables:

- `A_Gui`: Blank unless the message was sent to a GUI window or control, in which case `A_Gui` is the [Gui Window number](#) (this window is also set as the function's [default GUI window](#)).
- `A_GuiControl`: Blank unless the message was sent to a GUI control, in which case it contains the control's variable name or other value as described at [A_GuiControl](#). Some controls never receive certain types of messages. For example, when the user clicks a [text control](#), the operating system sends `WM_LBUTTONDOWN` to the parent window rather than the control; consequently, `A_GuiControl` is blank.
- `A_GuiX` and `A_GuiY`: Both contain `-2147483648` if the incoming message was sent via [SendMessage](#). If it was sent via [PostMessage](#), they contain the mouse cursor's coordinates (relative to the screen) at the time the message was posted.
- `A_EventInfo`: Contains `0` if the message was sent via [SendMessage](#). If sent via [PostMessage](#), it contains the [tick-count time](#) the message was posted.

A monitor function's [last found window](#) starts off as the parent window to which the message was sent (even if it was sent to a control). If the window is hidden but not a GUI window (such as the script's main window), turn on [DetectHiddenWindows](#) before using it. For example:

```
DetectHiddenWindows On
MsgParentWindow := WinExist() ; This stores the
unique ID of the window to which the message was
sent.
```

What the Function Should Return

If a monitor function uses [Return](#) without any parameters, or it specifies a blank value such as "" (or it never uses [Return](#) at all), the incoming message goes on to be processed normally when the function finishes. The same thing happens if the function [Exits](#) or causes a runtime error such as [running](#) a nonexistent file. By contrast, returning an integer causes it to be sent immediately as a reply; that is, the program does not process the message any further. For example, a function monitoring `WM_LBUTTONDOWN` (0x201) may return an integer to prevent the target window from being notified that a mouse click has occurred. In many cases (such as a message arriving via [PostMessage](#)), it does not matter which integer is returned; but if in doubt, 0 is usually safest.

The range of valid return values depends on whether the exe running the script is 32-bit or 64-bit. Non-empty return values must be between -2^{31} and $2^{32}-1$ for 32-bit scripts (`A_PtrSize = 4`) and between -2^{63} and $2^{63}-1$ for 64-bit scripts (`A_PtrSize = 8`).

[v1.1.20+]: If there are multiple functions monitoring a given message number, they are called one by one until one returns a non-empty value.

General Remarks

Unlike a normal function-call, the arrival of a monitored message calls the function as a new [thread](#). Because of this, the function starts off fresh with the default values for settings such as [SendMode](#) and [DetectHiddenWindows](#). These defaults can be changed in the [auto-execute section](#).

Messages sent to a control (rather than being posted) are not monitored because the system routes them directly to the control behind the scenes. This is seldom an issue for system-generated messages because most of them are posted.

Any script that calls `OnMessage` anywhere is automatically [persistent](#). It is also single-instance unless `#SingleInstance` has been used to override that.

If a message arrives while its function is still running due to a previous arrival of the same message, the function will not be called again (except if [MaxThreads](#) is greater than 1); instead, the message will be treated as unmonitored. If this is undesirable, a message greater than or equal to 0x312 can be buffered until its

function completes by specifying [Critical](#) as the first line of the function. Alternatively, [Thread Interrupt](#) can achieve the same thing as long as it lasts long enough for the function to finish. By contrast, a message less than 0x312 cannot be buffered by Critical or Thread Interrupt (however [in v1.0.46+] Critical may help because it checks messages [less often](#), which gives the function more time to finish). The only way to guarantee that no such messages are missed is to ensure the function finishes in under 6 milliseconds (though this limit can be raised via [Critical 30](#)). One way to do this is to have it queue up a future thread by [posting](#) to its own script a monitored message number higher than 0x312. That message's function should use [Critical](#) as its first line to ensure that its messages are buffered.

If a monitored message that is numerically less than 0x312 arrives while the script is absolutely uninterruptible -- such as while a [menu](#) is displayed, a [KeyDelay/MouseDelay](#) is in progress, or the clipboard is being [opened](#) -- the message's function will not be called and the message will be treated as unmonitored. By contrast, a monitored message of 0x312 or higher will be buffered during these uninterruptible periods; that is, its function will be called when the script becomes interruptible.

If a monitored message numerically less than 0x312 arrives while the script is uninterruptible merely due to the settings of [Thread Interrupt](#) or [Critical](#), the current thread will be interrupted so that the function can be called. By contrast, a monitored message of 0x312 or higher will be buffered until the thread finishes or becomes interruptible.

The [priority](#) of OnMessage threads is always 0. Consequently, no messages are monitored or buffered when the current thread's priority is higher than 0.

Caution should be used when monitoring system messages (those below 0x400). For example, if a monitor function does not finish quickly, the response to the message might take longer than the system expects, which might cause side-effects. Unwanted behavior may also occur if a monitor function returns an integer to suppress further processing of a message, but the system expected different processing or a different response.

When the script is displaying a system dialog such as [MsgBox](#), any message posted to a control is not monitored. For example, if the script is displaying a MsgBox and the user clicks a button in a GUI window, the

WM_LBUTTONDOWN message is sent directly to the button without calling the monitor function.

Although an external program may post messages directly to a script's thread via PostThreadMessage() or other API call, this is not recommended because the messages would be lost if the script is displaying a system window such as a [MsgBox](#). Instead, it is usually best to post or send the messages to the script's main window or one of its GUI windows.

Related

[RegisterCallback\(\)](#), [OnExit](#), [OnClipboardChange](#), [Post/SendMessage](#), [Functions](#), [List of Windows Messages](#), [Threads](#), [Critical](#), [DllCall\(\)](#)

Examples

```
; Example: The following is a working script that  
monitors mouse clicks in a GUI window.
```

```
; Related topic: GuiContextMenu
```

```
Gui, Add, Text,, Click anywhere in this window.  
Gui, Add, Edit, w200 vMyEdit  
Gui, Show  
OnMessage(0x201, "WM_LBUTTONDOWN")  
return
```

```
WM_LBUTTONDOWN(wParam, lParam)  
{  
    X := lParam & 0xFFFF  
    Y := lParam >> 16  
    if A_GuiControl  
        Control := "`n(in control " . A_GuiControl  
    . ")"  
    ToolTip You left-clicked in Gui window  
    #%A_Gui% at client coordinates %X%X%Y%.%Control%  
}
```

```
GuiClose:
ExitApp
```

```
; Example: The following script detects system shutdown/logoff and allows you to abort it (this is
```

```
; reported NOT to work on Windows Vista and later).
```

```
; Related topic: OnExit
```

```
; The following DllCall is optional: it tells the OS to shut down this script first (prior to all other applications).
```

```
DllCall("kernel32.dll\SetProcessShutdownParameters", UInt, 0x4FF, UInt, 0)
```

```
OnMessage(0x11, "WM_QUERYENDSESSION")
```

```
return
```

```
WM_QUERYENDSESSION(wParam, lParam)
```

```
{
```

```
    ENDESESSION_LOGOFF = 0x80000000
```

```
    if (lParam & ENDESESSION_LOGOFF) ; User is logging off.
```

```
        EventType = Logoff
```

```
    else ; System is either shutting down or restarting.
```

```
        EventType = Shutdown
```

```
    MsgBox, 4,, %EventType% in progress. Allow it?
```

```
    IfMsgBox Yes
```

```
        return true ; Tell the OS to allow the shutdown/logoff to continue.
```

```
    else
```

```
        return false ; Tell the OS to abort the shutdown/logoff.
```

```
}
```

```
; Example: Have a script receive a custom message
and up to two numbers from some other script or
program
; (to send strings rather than numbers, see the
example after this one).
```

```
OnMessage(0x5555, "MsgMonitor")
OnMessage(0x5556, "MsgMonitor")
```

```
MsgMonitor(wParam, lParam, msg)
{
    ; Since returning quickly is often important,
it is better to use a Tooltip than
    ; something like MsgBox that would prevent the
function from finishing:
    Tooltip Message %msg% arrived: `nWPARAM:
%wParam%`nLPARAM: %lParam%
}
```

```
; The following could be used inside some other
script to run the function inside the above
script:
SetTitleMatchMode 2
DetectHiddenWindows On
if WinExist("Name of Receiving Script.ahk
ahk_class AutoHotkey")
    PostMessage, 0x5555, 11, 22 ; The message is
sent to the "last found window" due to WinExist()
above.
DetectHiddenWindows Off ; Must not be turned off
until after PostMessage.
```

```
; Example: Send a string of any length from one
script to another. This is a working example.
; To use it, save and run both of the following
scripts then press Win+Space to show an
```

```
; InputBox that will prompt you to type in a string.
```

```
; Save the following script as "Receiver.ahk" then launch it:
```

```
#SingleInstance
```

```
OnMessage(0x4a, "Receive_WM_COPYDATA") ; 0x4a is WM_COPYDATA
```

```
return
```

```
Receive_WM_COPYDATA(wParam, lParam)
```

```
{
```

```
    StringAddress := NumGet(lParam + 2*A_PtrSize)
```

```
; Retrieves the CopyDataStruct's lpData member.
```

```
    CopyOfData := StrGet(StringAddress) ; Copy the string out of the structure.
```

```
; Show it with ToolTip vs. MsgBox so we can return in a timely fashion:
```

```
    ToolTip %A_ScriptName%`nReceived the following string:`n%CopyOfData%
```

```
    return true ; Returning 1 (true) is the traditional way to acknowledge this message.
```

```
}
```

```
; Save the following script as "Sender.ahk" then launch it. After that, press the Win+Space hotkey.
```

```
TargetScriptTitle = Receiver.ahk ahk_class
```

```
AutoHotkey
```

```
#space:: ; Win+Space hotkey. Press it to show an InputBox for entry of a message string.
```

```
InputBox, StringToSend, Send text via WM_COPYDATA, Enter some text to Send:
```

```
if ErrorLevel ; User pressed the Cancel button.
```

```
    return
```

```
result := Send_WM_COPYDATA(StringToSend,
```

```

TargetScriptTitle)
if result = FAIL
    MsgBox SendMessage failed. Does the following
WinTitle exist?:`n%TargetScriptTitle%
else if result = 0
    MsgBox Message sent but the target window
responded with 0, which may mean it ignored it.
return

Send_WM_COPYDATA(ByRef StringToSend, ByRef
TargetScriptTitle) ; ByRef saves a little memory
in this case.
; This function sends the specified string to the
specified window and returns the reply.
; The reply is 1 if the target window processed
the message, or 0 if it ignored it.
{
    VarSetCapacity(CopyDataStruct, 3*A_PtrSize, 0)
; Set up the structure's memory area.
    ; First set the structure's cbData member to
the size of the string, including its zero
terminator:
    SizeInBytes := (StrLen(StringToSend) + 1) *
(A_IsUnicode ? 2 : 1)
    NumPut(SizeInBytes, CopyDataStruct, A_PtrSize)
; OS requires that this be done.
    NumPut(&StringToSend, CopyDataStruct,
2*A_PtrSize) ; Set lpData to point to the string
itself.
    Prev_DetectHiddenWindows :=
A_DetectHiddenWindows
    Prev_TitleMatchMode := A_TitleMatchMode
    DetectHiddenWindows On
    SetTitleMatchMode 2
    TimeoutTime = 4000 ; Optional. Milliseconds
to wait for response from receiver.ahk. Default is
5000

```

```
    ; Must use SendMessage not PostMessage.
    SendMessage, 0x4a, 0, &CopyDataStruct,,
%TargetScriptTitle%,,,, %TimeOutTime% ; 0x4a is
WM_COPYDATA.
    DetectHiddenWindows %Prev_DetectHiddenWindows%
; Restore original setting for the caller.
    SetTitleMatchMode %Prev_TitleMatchMode%
; Same.
    return ErrorLevel ; Return SendMessage's
reply back to our caller.
}
```

```
; Example: See the WinLIRC client script for a
demonstration of how to use OnMessage() to receive
; notification when data has arrived on a network
connection.
```

Progress / SplashImage

Creates or updates a window containing a progress bar or an image.

```
SplashImage, Off
```

```
SplashImage [, ImageFile, Options, SubText, MainText,  
WinTitle, FontName]
```

```
Progress, Off
```

```
Progress, ProgressParam1 [, SubText, MainText,  
WinTitle, FontName]
```

Parameters

ImageFile

If this is the word OFF, the window is destroyed. If this is the word SHOW, the window is shown if it is currently hidden.

Otherwise, this is the file 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).

[AHK_L 59+]: Any image format supported by Gui may be used with SplashImage.

ImageFile is assumed to be in [%A_WorkingDir%](#) if an absolute path isn't specified. If *ImageFile* and *Options* 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 *SubText*, *MainText*, and *WinTitle*.

For newly created windows, if *ImageFile* is blank or there is a problem loading the image, the window will be displayed without the picture.

[v1.1.23+]: A [bitmap or icon handle](#) can be used instead of a filename.

For example, `HBITMAP:%handle%`.

ProgressParam1

If the progress window already exists: If *Param1* is the word OFF, the window is destroyed. If *Param1* is the word SHOW, the window is shown if it is currently hidden.

Otherwise, if *Param1* is a pure number, its bar's position is changed to that value. If *Param1* is blank, its bar position will be unchanged but its text will be updated to reflect any new strings provided in *SubText*, *MainText*, and *WinTitle*. 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 *Param1* is a string of zero or more options from the list below.

Options

A string of zero or more options from the list further 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 *MainText* 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 *MainText*. 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 *MainText*'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 [WinMove](#).

FontName

The name of the font to use for both *MainText* and *SubText*. 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.

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**.

Pn: For Progress windows, specify for **n** the starting position of the bar (the default is 0 or the number in the allowable range that is closest to 0). To later change the position of the bar, follow this example: `Progress, 50`.

Rx-y: For Progress windows, specify for **x-y** the numerical range of the bar (if the **R** option is not present, the default is 0-100). For example, `R0-1000` would allow a numbers between 0 and 1000; `R-50-50` would allow numbers between -50 and 50; and `R-10--5` would allow numbers between -10 and -5.

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. This setting also prevents a GUI window's `Gui +OwnDialogs` setting from taking ownership of a Splash or Progress 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.

Hide: The window will be initially hidden. Use `Progress Show` or `SplashImage Show` to show it later.

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). For 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. Specify -1 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. Specify -1 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 `cb1A00FF` 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. For 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, use [WinSet TransColor](#).

	Color name	RGB value
	Black	000000
	Silver	C0C0C0
	Gray	808080
	White	FFFFFF
	Maroon	800000
	Red	FF0000
	Purple	800080
		

	Fuchsia	FF00FF
	Green	008000
	Lime	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 or later, 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.

A GUI window may take ownership of a Progress or Splash window by means of [Gui +OwnDialogs](#). This causes the Splash or Progress to stay always on top of its owner. In addition, the Progress or Splash is automatically destroyed when its GUI window is destroyed.

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 18-point
Courier text:
Progress, m2 b fs18 zh0, This is the Text.`nThis
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
fs18, 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, %A_WinDir%\system32\ntimage.gif, A,,,
Installation
Loop, %A_WinDir%\system32\*.*
{
    Progress, %A_Index%, %A_LoopFileName%,
Installing..., Draft Installation
    Sleep, 50
```

```
if A_Index = 100  
    break
```

```
}
```

; There is similar example at the bottom of the [GUI page](#). Its advantage is that it uses only a single window and it gives you more control over window layout.

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. This parameter can be an [expression](#).

Height

The height in pixels of the window (not including its title bar). Default 0 (i.e. just the title bar will be shown). This parameter can be an [expression](#).

Title

The title of the window. Default empty (blank).

Text

The text of the window. Default empty (blank). If *Text* is long, it can be broken up into several shorter lines by means of a [continuation section](#), which might improve readability and maintainability.

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. For 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, , , Displays only a title bar.  
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
```

Standard Windows Fonts

For use with the [Progress](#), [GUI](#), and [SplashImage](#) commands. The column "Exists Since" represents the first Windows version in which the font was included. Common monospaced (fixed-width) fonts in this table are Consolas, Courier, Courier New, Fixedsys, Lucida Console and Terminal. The availability of some fonts may depend on the language of the operating system.

Recommend Fonts are highlighted in yellow. Note that fonts recommended here are fonts which are readable and have been included since Windows XP or earlier. Using such fonts ensures that they are displayed on most operating systems.

Click on a column header to sort the contents of the table by it.

Font Name	Mainly Designed For	Exists Since
Aharoni	Hebrew	2000
Aldhabi	Arabic	8
Andalus	Arabic	2000
Angsana New	Thai	XP
AngsanaUPC	Thai	XP
Aparajita	Indic	7
Arabic Typesetting	Arabic	Vista
Arial	Latin, Greek, Cyrillic	95
Arial Black	Latin, Greek, Cyrillic	98
Arial Nova	Latin, Greek, Cyrillic	10
Bahnschrift	Latin	10
Batang	Korean	2000
BatangChe	Korean	Vista

Browallia New	Thai	2000
BrowalliaUPC	Thai	2000
Calibri	Latin, Greek, Cyrillic	Vista
Calibri Light	Latin, Greek, Cyrillic	8
Cambria	Latin, Greek, Cyrillic	Vista
Cambria Math	Symbols	Vista
Candara	Latin, Greek, Cyrillic	Vista
Comic Sans MS	Latin, Greek, Cyrillic	95
Consolas	Latin, Greek, Cyrillic	Vista
Constantia	Latin, Greek, Cyrillic	Vista
Corbel	Latin, Greek, Cyrillic	Vista
Cordia New	Thai	2000
CordiaUPC	Thai	2000
Courier	Latin, Greek, Cyrillic	95
Courier New	Latin, Greek, Cyrillic	95
DaunPenh	Other	Vista
David	Hebrew	2000
DengXian	Chinese	10
DFKai-SB	Chinese	2000
DilleniaUPC	Thai	2000
DokChampa	Other	Vista
Dotum	Korean	Vista
DotumChe	Korean	Vista
Ebrima	African	7
Estrangelo Edessa	Other	XP

EucrosiaUPC	Thai	2000
Euphemia	Other	Vista
FangSong	Chinese	Vista
Fixedsys	Latin	95
Franklin Gothic Medium	Latin, Greek, Cyrillic	XP
FrankRuehl	Hebrew	2000
FreesiaUPC	Thai	2000
Gabriola	Latin, Greek, Cyrillic	7
Gadugi	Other	8
Gautami	Other	XP
Georgia	Latin, Greek, Cyrillic	2000
Georgia Pro	Latin, Greek, Cyrillic	10
Gill Sans Nova	Latin, Greek, Cyrillic	10
Gisha	Hebrew	Vista
Gulim	Korean	2000
GulimChe	Korean	Vista
Gungsuh	Korean	Vista
GungsuhChe	Korean	Vista
HoloLens MDL2 Assets	Symbols	10
Impact	Latin, Greek, Cyrillic	98
IrisUPC	Thai	2000
Iskoola Pota	Other	Vista
JasmineUPC	Thai	2000

Javanese Text	Javanese	8.1
KaiTi	Chinese	2000
Kalinga	Other	Vista
Kartika	Other	XP
Khmer UI	Other	7
KodchiangUPC	Thai	2000
Kokila	Other	7
Lao UI	Other	7
Latha	Indic	XP
Leelawadee	Thai	Vista
Leelawadee UI	Thai	8.1
Levenim MT	Hebrew	98
LilyUPC	Thai	2000
Lucida Console	Latin, Greek, Cyrillic	98
Lucida Sans	Latin, Greek, Cyrillic	98
Lucida Sans Unicode	Latin, Greek, Cyrillic	98
Malgun Gothic	Korean	Vista
Mangal	Indic	XP
Marlett	Symbols	95
Meiryo	Japanese	Vista
Meiryo UI	Japanese	7
Microsoft Himalaya	Other	Vista
Microsoft JhengHei	Chinese	Vista
Microsoft JhengHei UI	Chinese	8
Microsoft New Tai Lue	Other	7

Microsoft PhagsPa	Other	7
Microsoft Sans Serif	Latin, Greek, Cyrillic	2000
Microsoft Tai Le	Other	7
Microsoft Uighur	Other	Vista
Microsoft YaHei	Chinese	Vista
Microsoft YaHei UI	Chinese	8
Microsoft Yi Baiti	Other	Vista
MingLiU	Chinese	Vista
MingLiU_HKSCS	Chinese	Vista
MingLiU_HKSCS-ExtB	Chinese	Vista
MingLiU-ExtB	Chinese	Vista
Miriam	Hebrew	2000
Miriam Fixed	Hebrew	2000
Modern	Latin	95
Mongolian Baiti	Other	Vista
MoolBoran	Other	Vista
MS Gothic	Japanese	2000
MS Mincho	Japanese	2000
MS PGothic	Japanese	Vista
MS PMincho	Japanese	Vista
MS Serif	Latin	95
MS Sans Serif	Latin	95
MS UI Gothic	Japanese	Vista
MV Boli	Other	XP

Myanmar Text	Other	8
Narkisim	Hebrew	2000
Neue Haas Grotesk Text Pro	Latin	10
Nirmala UI	Indic	8
NSimSun	Chinese	Vista
Nyala	African	Vista
Palatino Linotype	Latin, Greek, Cyrillic	2000
Plantagenet Cherokee	Other	Vista
PMingLiU	Chinese	2000
PMingLiU-ExtB	Chinese	Vista
Raavi	Indic	XP
Rockwell Nova	Latin, Greek, Cyrillic	10
Rod	Hebrew	2000
Roman	Latin	98
Sakkal Majalla	Arabic	7
Sanskrit Text	Other	10
Script	Latin	98
Segoe MDL2 Assets	Symbols	10
Segoe Print	Latin, Greek, Cyrillic	Vista
Segoe Script	Latin, Greek, Cyrillic	Vista
Segoe UI	Latin, Greek, Cyrillic	Vista
Segoe UI Emoji	Symbols	10
Segoe UI Historic	Other	10
Segoe UI Light	Latin, Greek, Cyrillic	7

Segoe UI Semibold	Latin, Greek, Cyrillic	7
Segoe UI Semilight	Latin, Greek, Cyrillic	8
Segoe UI Symbol	Symbols	7
Shonar Bangla	Indic	7
Shruti	Indic	XP
SimHei	Chinese	2000
Simplified Arabic	Arabic	2000
Simplified Arabic Fixed	Arabic	2000
SimSun	Chinese	2000
SimSun-ExtB	Chinese	Vista
Sitka Banner	Latin, Greek, Cyrillic	8.1
Sitka Display	Latin, Greek, Cyrillic	8.1
Sitka Heading	Latin, Greek, Cyrillic	8.1
Sitka Small	Latin, Greek, Cyrillic	8.1
Sitka Subheading	Latin, Greek, Cyrillic	8.1
Sitka Text	Latin, Greek, Cyrillic	8.1
Small Fonts	Latin	95
Sylfaen	Other	XP
Symbol	Symbols	95
System	Latin	95
Tahoma	Latin, Greek, Cyrillic	95
Terminal	Latin	95
Times New Roman	Latin, Greek, Cyrillic	95
Traditional Arabic	Arabic	2000
Trebuchet MS	Latin, Greek, Cyrillic	2000

Tunga	Indic	XP
UD Digi Kyokasho N-B	Japanese	10
UD Digi Kyokasho NK-B	Japanese	10
UD Digi Kyokasho NK-R	Japanese	10
UD Digi Kyokasho NP-B	Japanese	10
UD Digi Kyokasho NP-R	Japanese	10
UD Digi Kyokasho N-R	Japanese	10
Urdu Typesetting	Arabic	8
Utsaah	Indic	7
Vani	Indic	7
Verdana	Latin, Greek, Cyrillic	95
Verdana Pro	Latin, Greek, Cyrillic	10
Vijaya	Indic	7
Vrinda	Indic	XP
Webdings	Symbols	98
Wingdings	Symbols	95
Yu Gothic	Japanese	8.1
Yu Gothic UI	Japanese	8.1
Yu Mincho	Japanese	8.1

Styles Usable by the **Gui** and **GuiControl** Commands

Table of Contents

- Styles Common to Gui/Parent Windows and Most Control Types
- Text | Edit | UpDown | Picture
- Button | Checkbox | Radio | GroupBox
- DropDownList | ComboBox
- ListBox | ListView | TreeView
- DateTime | MonthCal
- Slider | Progress | Tab | StatusBar

Common styles

default for GUI window		WS_POPUP , WS_CAPTION , WS_SYSMENU , WS_MINIMIZEBOX
forced for GUI window		WS_CLIPSIBLINGS
WS_BORDER	0x800000	+/-Border. Creates a window that has a thin-line border.
WS_POPUP	0x80000000	Creates a pop-up window. This style cannot be used with the WS_CHILD style.
WS_CAPTION	0xC00000	+/-Caption. Creates a window that has a title bar. This style is a numerical combination of WS_BORDER and WS_DLGFRAME .
WS_DISABLED	0x80000000	+/-Disabled. Creates a window that is initially disabled.
WS_DLGFRAME	0x400000	Creates a window that has a border of a style typically used with dialog boxes.
WS_GROUP	0x20000	+/-Group. Indicates that this control is the first one in a group of controls. This style is automatically applied to manage the "only one at a time" behavior of radio buttons. In the rare case where two groups of radio

		buttons are added consecutively (with no other control types in between them), this style may be applied manually to the first control of the second radio group, which splits it off from the first.
WS_HSCROLL	0x100000	Creates a window that has a horizontal scroll bar.
WS_MAXIMIZE	0x1000000	Creates a window that is initially maximized.
WS_MAXIMIZEBOX	0x10000	+/-MaximizeBox. Creates a window that has a maximize button. Cannot be combined with the WS_EX_CONTEXTHELP style. The WS_SYSMENU style must also be specified.
WS_MINIMIZE	0x20000000	Creates a window that is initially minimized.
WS_MINIMIZEBOX	0x20000	+/-MinimizeBox. Creates a window that has a minimize button. Cannot be combined with the WS_EX_CONTEXTHELP style. The WS_SYSMENU style must also be specified.
WS_OVERLAPPED	0	Creates an overlapped window. An overlapped window has a title bar and a border. Same as the WS_TILED style.
		Creates an overlapped window with the

WS_OVERLAPPEDWINDOW	0xCF0000	<p>WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX, and WS_MAXIMIZEBOX styles. Same as the WS_TILEDWINDOW style.</p>
WS_POPUPWINDOW	0x80880000	<p>Creates a pop-up window with WS_BORDER, WS_POPUP, and WS_SYSMENU styles. The WS_CAPTION and WS_POPUPWINDOW styles must be combined to make the window menu visible.</p>
WS_SIZEBOX	0x40000	<p>+/--Resize. Creates a window that has a sizing border. Same as the WS_THICKFRAME style.</p>
WS_SYSMENU	0x80000	<p>+/--SysMenu. Creates a window that has a window menu on its title bar. The WS_CAPTION style must also be specified.</p>
WS_TABSTOP	0x10000	<p>+/--Tabstop. Specifies a control that can receive the keyboard focus when the user presses the  key. Pressing the  key changes the keyboard focus to the next control with the WS_TABSTOP</p>

		style.
WS_THICKFRAME	0x40000	Creates a window that has a sizing border. Same as the WS_SIZEBOX style
WS_VSCROLL	0x200000	Creates a window that has a vertical scroll bar.
WS_VISIBLE	0x10000000	Creates a window that is initially visible.
WS_CHILD	0x40000000	Creates a child window. A window with this style cannot have a menu bar. This style cannot be used with the WS_POPUP style.

Text styles

default		None
forced		None
SS_BLACKFRAME	0x7	Specifies a box with a frame drawn in the same color as the window frames. This color is black in the default color scheme.
SS_BLACKRECT	0x4	Specifies a rectangle filled with the current window frame color. This color is black in the default color scheme.
SS_CENTER	0x1	+/-Center. Specifies a simple rectangle and centers the error value text in the rectangle. The control automatically wraps words that extend past the end of a line to the beginning of the next centered line.
SS_ETCHEDFRAME	0x12	Draws the frame of the static control using the EDGE_ETCHED edge style.
SS_ETCHEDHORZ	0x10	Draws the top and bottom edges of the static control using the EDGE_ETCHED edge style.
SS_ETCHEDVERT	0x11	Draws the left and right edges of the static control using the EDGE_ETCHED edge style.
SS_GRAYFRAME	0x8	Specifies a box with a frame drawn with the same color as the screen background (desktop). This color is gray in the default color scheme
		Specifies a rectangle filled with the

SS_GRAYRECT	0x5	current screen background color. This color is gray in the default color scheme.
SS_LEFT	0	+/-Left. This is the default. It specifies a simple rectangle and left-aligns the text in the rectangle. The text is formatted before it is displayed. Words that extend past the end of a line are automatically wrapped to the beginning of the next left-aligned line. Words that are longer than the width of the control are truncated.
SS_LEFTNOWORDWRAP	0xC	+/-Wrap. Specifies a rectangle and left-aligns the text in the rectangle. Tabs are expanded, but words are not wrapped. Text that extends past the end of a line is clipped.
SS_NOPREFIX	0x80	Prevents interpretation of any ampersand (&) characters in the control's text as accelerator prefix characters. This can be useful when file names or other strings that might contain an ampersand (&) must be displayed within a text control
SS_NOTIFY	0x100	Sends the parent window the STN_CLICKED notification when the user clicks the control.
SS_RIGHT	0x2	+/-Right. Specifies a rectangle and right-aligns the specified text in the rectangle.
SS_SUNKEN	0x1000	Draws a half-sunken border around a static control.
		Specifies a box with a frame drawn

SS_WHITEFRAME	0x9	with the same color as the window background. This color is white in the default color scheme.
SS_WHITERECT	0x6	Specifies a rectangle filled with the current window background color. This color is white in the default color scheme.

Edit styles

default		<p>WS_TABSTOP and WS_EX_CLIENTEDGE (extended style E0x200)</p> <p>If an Edit is auto-detected as multi-line due to its starting contents containing multiple lines, its height being taller than 1 row, or its row-count having been explicitly specified as greater than 1, the following styles will be applied by default: WS_VSCROLL, ES_WANTRETURN, and ES_AUTOVSCROLL</p> <p>If an Edit is auto-detected as a single line, it defaults to having ES_AUTOHSCROLL.</p>
forced		None
ES_AUTOHSCROLL	0x80	+/-Wrap for multi-line edits, and +/-Limit for single-line edits. Automatically scrolls text to the right by 10 characters when the user types a character at the end of the line. When the user presses the  key, the control scrolls all text back to the zero position.
ES_AUTOVSCROLL	0x40	Scrolls text up one page when the user presses the  the last line.
ES_CENTER	0x1	+/-Center. Centers text in a multiline edit control.
ES_LOWERCASE	0x10	+/-Lowercase. Converts all characters to lowercase as they are typed into the edit

		control.
ES_NOHIDSEL	0x100	Negates the default behavior for an edit control. The default behavior hides the selection when the control loses the input focus and inverts the selection when the control receives the input focus. If you specify ES_NOHIDSEL , the selected text is inverted, even if the control does not have the focus.
ES_NUMBER	0x2000	+/-Number. Prevents the user from typing anything other than digits in the control.
ES_OEMCONVERT	0x400	This style is most useful for edit controls that contain file names.
ES_MULTILINE	0x4	+/-Multi. Designates a multiline edit control. The default is a single-line edit control.
ES_PASSWORD	0x20	+/-Password. Displays a masking character in place of each character that is typed into the edit control, which conceals the text.
ES_READONLY	0x800	+/-ReadOnly. Prevents the user from typing or editing text in the edit control.
ES_RIGHT	0x2	+/-Right. Right-aligns text in a multiline edit control.
ES_UPPERCASE	0x8	+/-Uppercase. Converts all characters to uppercase as they are typed into the edit control.
ES_WANTRETURN	0x1000	+/-WantReturn. Specifies that a carriage return be inserted when the user presses the  typing text into a multiline edit control in a dialog box. If you do not specify this style, pressing the  has the same effect as

pressing the dialog box's default push button. This style has no effect on a single-line edit control.

UpDown styles

default		UDS_ARROWKEYS, UDS_ALIGNRIGHT, UDS_SETBUDDYINT, and UDS_AUTOBUDDY.
forced		None
UDS_WRAP	0x1	Named option "Wrap". Causes the control to wrap around to the other end of its range when the user attempts to go beyond the minimum or maximum. Without <i>Wrap</i> , the control stops when the minimum or maximum is reached.
UDS_SETBUDDYINT	0x2	Causes the UpDown control to set the text of the buddy control (using the WM_SETTEXT message) when the position changes. However, if the buddy is a ListBox, the ListBox's current selection is changed instead.
UDS_ALIGNRIGHT	0x4	Named option "Right" (default). Positions UpDown on the right side of its buddy control.
UDS_ALIGNLEFT	0x8	Named option "Left". Positions UpDown on the left side of its buddy control.
UDS_AUTOBUDDY	0x10	Automatically selects the previous control in the z-order as the UpDown control's buddy control.
UDS_ARROWKEYS	0x20	Allows the user to press the Up or Down arrow keys on the keyboard to increase or decrease the UpDown control's position.
		Named option "Horz". Causes the

UDS_HORZ	0x40	control's arrows to point left and right instead of up and down.
UDS_NOTHOUSANDS	0x80	Does not insert a thousands separator between every three decimal digits in the buddy control.
UDS_HOTTRACK	0x100	Causes the control to exhibit "hot tracking" behavior. That is, it highlights the control's buttons as the mouse passes over them. The flag is ignored on Windows XP when the desktop theme overrides it.
(hex)		<p>The number format displayed inside the buddy control may be changed from decimal to hexadecimal by following this example:</p> <pre data-bbox="828 1018 1352 1297">Gui +LastFound SendMessage, 1133, 16, 0, msctls_updown321 ; 1133 is UDM_SETBASE</pre> <p>However, this affects only the buddy control, not the UpDown's reported position.</p>

Picture styles

default		None
forced		SS_ICON for icons and cursors. SS_BITMAP for other image types.
SS_REALSIZECONTROL	0x40	[Windows XP or later] Adjusts the bitmap to fit the size of the control.
SS_CENTERIMAGE	0x200	Centers the bitmap in the control. If the bitmap is too large, it will be clipped. For text controls If the control contains a single line of text, the text is centered vertically within the available height of the control Microsoft Windows XP: This style bit no longer results in unused portions of the control being filled with the color of the top left pixel of the bitmap or icon. Unused portions of the control will remain the background color.

Button, Checkbox, Radio, and GroupBox styles

default		<p>BS_MULTILINE (except for GroupBox, and except for buttons, checkboxes, and radios that have no explicitly set width or height, nor any CR/LF characters in their text)</p> <p>WS_TABSTOP (except for GroupBox) -- however, radio buttons other than the first of each radio group lack WS_TABSTOP by default. In addition, radio buttons have BS_NOTIFY so that double clicks can be detected.</p>
forced		<p>Button: BS_PUSHBUTTON or BS_DEFPUSHBUTTON</p> <p>Radio: BS_AUTORADIOBUTTON</p> <p>Checkbox: BS_AUTOCHECKBOX or BS_AUTO3STATE</p> <p>GroupBox: BS_GROUPBOX</p>
BS_LEFT	0x100	+/-Left. Left-aligns the text.
BS_PUSHLIKE	0x1000	Makes a checkbox or radio button look and act like a push button. The button looks raised when it isn't pushed or checked, and sunken when it is pushed or checked.
BS_RIGHT	0x200	+/-Right. Right-aligns the text.
BS_RIGHTBUTTON	0x20	+Right (i.e. +Right includes both BS_RIGHT and BS_RIGHTBUTTON , but -Right removes only BS_RIGHT, not BS_RIGHTBUTTON). Positions a checkbox square or radio button circle

		on the right side of the control's available width instead of the left.
BS_BOTTOM	0x800	Places the text at the bottom of the control's available height.
BS_CENTER	0x300	+/-Center. Centers the text horizontally within the control's available width.
BS_DEFPUSHBUTTON	0x1	+/-Default. Creates a push button with a heavy black border. If the button is in a dialog box, the user can select the button by pressing the Enter  even when the button does not have the input focus. This style is useful for enabling the user to quickly select the most likely option.
BS_MULTILINE	0x2000	+/-Wrap. Wraps the text to multiple lines if the text is too long to fit on a single line in the control's available width. This also allows linefeed (<code>\n</code>) to start new lines of text.
BS_TOP	0x400	Places text at the top of the control's available height.
BS_VCENTER	0xC00	Vertically centers text in the control's available height.
BS_FLAT	0x8000	Specifies that the button is two-dimensional; it does not use the default shading to create a 3-D effect.

DropDownList and ComboBox styles

default		<p>WS_TABSTOP (+/-Tabstop)</p> <p>DropDownList: WS_VSCROLL</p> <p>ComboBox: WS_VSCROLL, CBS_AUTOHSCROLL</p>
forced		<p>DropDownList: CBS_DROPDOWNLIST</p> <p>ComboBox: Either CBS_DROPDOWN or CBS_SIMPLE</p>
CBS_AUTOHSCROLL	0x40	<p>+/-Limit. Automatically scrolls the text in an edit control to the right when the user types a character at the end of the line. If this style is not set, only text that fits within the rectangular boundary is enabled.</p>
CBS_DISABLENOSCROLL	0x800	<p>Shows a disabled vertical scroll bar in the list box when the box does not contain enough items to scroll. Without this style, the scroll bar is hidden when the list box does not contain enough items.</p>
CBS_LOWERCASE	0x4000	<p>+/-Lowercase. Converts to lowercase any uppercase characters that are typed into the edit control of a combo box.</p>
CBS_NOINTEGRALHEIGHT	0x400	<p>Specifies that the combo box will be exactly the size specified by the application when it created the combo box. Usually, Windows CE sizes a combo box</p>

		so that it does not display partial items.
CBS_OEMCONVERT	0x80	Converts text typed in the combo box edit control from the Windows CE character set to the OEM character set and then back to the Windows CE set. This style is most useful for combo boxes that contain file names. It applies only to combo boxes created with the CBS_DROPDOWN style.
CBS_SIMPLE	0x1	+/-Simple (ComboBox only). Displays the list box at all times. The current selection in the list box is displayed in the edit control.
CBS_SORT	0x100	+/-Sort. Sorts the items in the drop-list alphabetically.
CBS_UPPERCASE	0x2000	+/-Uppercase. Converts to uppercase any lowercase characters that are typed into the edit control of a ComboBox.

ListBox styles

default		WS_TABSTOP , LBS_USETABSTOPS , WS_VSCROLL , and WS_EX_CLIENTEDGE (extended style E0x200).
forced		LBS_NOTIFY (supports detection of double-clicks)
LBS_DISABLENOSCROLL	0x1000	Shows a disabled vertical scroll bar for the list box when the box does not contain enough items to scroll. If you do not specify this style, the scroll bar is hidden when the list box does not contain enough items.
LBS_NOINTEGRALHEIGHT	0x100	Specifies that the list box will be exactly the size specified by the application when it created the list box.
LBS_EXTENDEDSEL	0x800	+/-Multi. Allows multiple selections via control-click and shift-click.
LBS_MULTIPLESEL	0x8	A simplified version of multi-select in which control-click and shift-click are not necessary because normal left clicks serve to extend the selection or de-select a selected item.
LBS_NOSEL	0x4000	+/-ReadOnly. Specifies that the user can view list box strings but cannot select them.
LBS_SORT	0x2	+/-Sort. Sorts the items in the list

		box alphabetically.
LBS_USETABSTOPS	0x80	Enables a ListBox to recognize and expand tab characters when drawing its strings. The default tab positions are 32 dialog box units. A dialog box unit is equal to one-fourth of the current dialog box base-width unit.

ListView styles

default		WS_TABSTOP, LVS_REPORT, LVS_SHOWSELALWAYS, LVS_EX_FULLROWSELECT, LVS_EX_HEADERDRAGDROP, WS_EX_CLIENTEDGE (E0x200)
forced		None
LVS_ALIGNLEFT	0x800	Items are left-aligned in icon and small icon view.
LVS_ALIGNTOP	0	Items are aligned with the top of the list-view control in icon and small icon view. This is the default.
LVS_AUTOARRANGE	0x100	Icons are automatically kept arranged in icon and small icon view.
LVS_EDITLABELS	0x200	+/-ReadOnly. Specifying -ReadOnly (or +0x200) allows the user to edit the first field of each row in place.
LVS_ICON	0	+Icon. Specifies large-icon view.
LVS_LIST	0x3	+List. Specifies list view.
LVS_NOCOLUMNHEADER	0x4000	+/-Hdr. Avoids displaying column headers in report view.
LVS_NOLABELWRAP	0x80	Item text is displayed on a single line in icon view. By default, item text may wrap in icon view.
		Scrolling is disabled. All items must be within the client area.

LVS_NOSCROLL	0x2000	This style is not compatible with the LVS_LIST or LVS_REPORT styles.
LVS_NOSORTHEADER	0x8000	+/-NoSortHdr. Column headers do not work like buttons. This style can be used if clicking a column header in report view does not carry out an action, such as sorting.
LVS_OWNERDATA	0x1000	This style specifies a virtual list-view control (not directly supported by AutoHotkey).
LVS_OWNERDRAWFIXED	0x400	The owner window can paint items in report view in response to WM_DRAWITEM messages (not directly supported by AutoHotkey).
LVS_REPORT	0x1	+Report. Specifies report view.
LVS_SHAREIMAGELISTS	0x40	The image list will not be deleted when the control is destroyed. This style enables the use of the same image lists with multiple list-view controls.
LVS_SHOWSELALWAYS	0x8	The selection, if any, is always shown, even if the control does not have keyboard focus.
LVS_SINGLESEL	0x4	+/-Multi. Only one item at a time can be selected. By default, multiple items can be selected.
LVS_SMALLICON	0x2	+IconSmall. Specifies small-icon view.
LVS_SORTASCENDING	0x10	+/-Sort. Rows are sorted in ascending order based on the contents of the first field.

LVS_SORTDESCENDING	0x20	+/-SortDesc. Same as above but in descending order.
--------------------	------	---

Extended ListView styles require the **LV prefix** when used with the Gui commands. Some extended styles introduced in Windows XP or later versions are not listed here. For a full list, see [MSDN: Extended List-View Styles](#).

LVS_EX_BORDERSELECT	LV0x8000	When an item is selected the border color of the item changes rather than the item being highlighted (might be non-functional in recent operating systems).
LVS_EX_CHECKBOXES	LV0x4	<p>+/-Checked. Displays a checkbox with each item. When set with this style, the control creates a state image list with two images using DrawFrameControl. image 1 is the unchecked state image 2 is the checked state. Setting the state image to 0 removes the check box alt</p> <p>Windows XP or later: Checkboxes are visible and functional in all list-view modes except the 'None' view mode introduced in Windows XP. Clicking a checkbox in 'None' view mode only selects the item, the state does not change.</p>
		Windows XP or later: Pairing with LVS_EX_DOUBLEBUFFER, which removes double-buffering, which r

LVS_EX_DOUBLEBUFFER	LV0x10000	flicker. This extended style enables alpha-blended mouse selection on systems where supported.
LVS_EX_FLATSB	LV0x100	Enables flat scroll bars in view.
LVS_EX_FULLROWSELECT	LV0x20	When a row is selected, all fields are highlighted. This is available only in conjunction with the LVS_REPORT style.
LVS_EX_GRIDLINES	LV0x1	+/-Grid. Displays gridline rows and columns. This is available only in conjunction with the LVS_REPORT style.
LVS_EX_HEADERDRAGDROP	LV0x10	Enables drag-and-drop reordering of columns in a list-view control. This style is only available for view controls that use the LVS_REPORT style.
LVS_EX_INFOTIP	LV0x400	When a list-view control uses the LVS_EX_INFOTIP style, LVN_GETINFOTIP notification message is sent to the parent window before displaying item's ToolTip.
LVS_EX_LABELTIP	LV0x4000	If a partially hidden label in list-view mode lacks ToolTip, the list-view control will use the label. If this style is not set, the list-view control will unhide hidden labels only for the icon mode. Requires Windows XP or later, or the DLLs distributed with Internet Explorer 5.0.

		Note: On some versions of Windows, this style might work properly if the GUI is set to be always-on-top.
LVS_EX_MULTIWORAREAS	LV0x2000	If the list-view control has LVS_AUTOARRANGE style, the control will not autoarrange icons until one or more work areas are defined (see LVM_SETWORKAREAS). If LVS_AUTOARRANGE is effective, this style must be set before any work areas are defined and any items have been added to the control.
LVS_EX_ONECLICKACTIVATE	LV0x40	The list-view control sends LVN_ITEMACTIVATE notification message to the window when the user clicks an item. This style also enables hot tracking in the list-view control. Hot tracking means that when the cursor moves over an item, the item is highlighted but not selected.
LVS_EX_REGIONAL	LV0x200	Sets the list-view window to include only the item icon and text using SetWindowRgn to create a region area that is not part of an icon area excluded from the window. This style is only available for list-view controls that use the LVS_ICON style.
LVS_EX_SIMPLESELECT	LV0x100000	In icon view, moves the start image of the control to the right of the large icon region. In views other than icon view, there is no change. When the user

		the state by using the spacebar to cycle over selected items, or the arrow keys to move the focus to the next item with the focus. Requires Windows XP or later.
LVS_EX_SUBITEMIMAGES	LV0x2	Allows images to be displayed in list-view fields beyond the first. This style is available only in conjunction with the LVS_REPORT style.
LVS_EX_TRACKSELECT	LV0x8	Enables hot-track selection in list-view control. Hot track selection means that an item is automatically selected when the mouse cursor remains over the item for a certain period of time. This period can be changed from the Control Panel system setting with a registry value LVM_SETHOVERTIME. This style applies to all styles used in list-view control. You can disable whether hot-track selection is enabled by calling SystemParametersInfo.
LVS_EX_TWOCLICKACTIVATE	LV0x80	The list-view control sends LVN_ITEMACTIVATE notification message to the parent window when the user double-clicks an item. This style enables hot tracking in the list-view control. Hot tracking means that when the cursor moves over an item, it is highlighted but not selected.
LVS_EX_UNDERLINECOLD	LV0x1000	Causes those non-hot items to be activated to be displayed with underlined text. This style requires that

		LVS_EX_TWOCCLICKAC be set also.
LVS_EX_UNDERLINEHOT	LV0x800	Causes those hot items that are activated to be displayed with underlined text. This style requires that LVS_EX_ONECLICKAC or LVS_EX_TWOCCLICKAC also be set.

TreeView styles

default		WS_TABSTOP , TVS_SHOWSELALWAYS , TVS_HASLINES , TVS_LINESATROOT , TVS_HASBUTTONS , WS_EX_CLIENTEDGE (E0x200)
forced		None
TVS_CHECKBOXES	0x100	+/-Checked. Displays a checkbox next to each item.
TVS_DISABLEDRAHDROP	0x10	Prevents the tree-view control from sending TVN_BEGINDRAG notification messages.
TVS_EDITLABELS	0x8	+/-ReadOnly. Allows the user to edit the names of tree-view items.
TVS_FULLROWSELECT	0x1000	Enables full-row selection in the tree view. The entire row of the selected item is highlighted, and clicking anywhere on an item's row causes it to be selected. This style cannot be used in conjunction with the TVS_HASLINES style.
TVS_HASBUTTONS	0x1	+/-Buttons. Displays plus (+) and minus (-) buttons next to parent items. The user clicks the buttons to expand or collapse a parent item's list of child items. To include buttons with items at the root of the tree view, TVS_LINESATROOT must also

		be specified.
TVS_HASLINES	0x2	+/-Lines. Uses lines to show the hierarchy of items.
TVS_INFOTIP	0x800	Obtains ToolTip information by sending the TVN_GETINFOTIP notification.
TVS_LINESATROOT	0x4	+/-Lines. Uses lines to link items at the root of the tree-view control. This value is ignored if TVS_HASLINES is not also specified.
TVS_NOHSCROLL	0x8000	+/-HScroll. Disables horizontal scrolling in the control. The control will not display any horizontal scroll bars.
TVS_NONEVENHEIGHT	0x4000	Sets the height of the items to an odd height with the TVM_SETITEMHEIGHT message. By default, the height of items must be an even value.
TVS_NOSCROLL	0x2000	Disables both horizontal and vertical scrolling in the control. The control will not display any scroll bars.
TVS_NOTOOLTIPS	0x80	Disables ToolTips.
TVS_RTLREADING	0x40	Causes text to be displayed from right-to-left (RTL). Usually, windows display text left-to-right (LTR).
TVS_SHOWSELALWAYS	0x20	Causes a selected item to remain selected when the tree-view control loses focus.
		Causes the item being selected to

TVS_SINGLEEXPAND	0x400	expand and the item being unselected to collapse upon selection in the tree-view. If the user holds down the Ctrl  while selecting an item, the item being unselected will not be collapsed.
TVS_TRACKSELECT	0x200	Enables hot tracking of the mouse in a tree-view control.

DateTime styles

default		DTS_SHORTDATECENTURYFORMAT and WS_TABSTOP (+/-Tab)
forced		None
DTS_UPDOWN	0x1	Provides an up-down control of the control to modify date which replaces the of the date month calendar that would be available.
DTS_SHOWNONE	0x2	Displays a checkbox inside the control that users can uncheck to make the control have no date/time style. Whenever the control has no style, Gui Submit and GuiControl retrieve a blank value (empty).
DTS_SHORTDATEFORMAT	0x0	Displays the date in short format. In some locales, it looks like 6/1/05. On older operating systems, the year might be displayed. The default is DTS_SHORTDATECENTURYFORMAT and not DTS_SHORTDATEFORMAT .
DTS_LONGDATEFORMAT	0x4	Format option "LongDate". Displays the date in long format. In some locales, it looks like Wednesday, June 1, 2005.
DTS_SHORTDATECENTURYFORMAT	0xC	Format option blank/omit the date in short format with the year. In some locales, it looks like 6/1/2005. If the system's version of Comctl32.dll is older than 5.0, this style is not supported and the default is DTS_SHORTDATEFORMAT .

		automatically substituted.
DTS_TIMEFORMAT	0x9	Format option "Time". Displays time, which in some locales is 5:31:42 PM.
DTS_APPCANPARSE	0x10	Not yet supported. Allows to parse user input and take next action. It enables users to enter client area of the control with the  . The control sends DTN_USERSTRING notification messages when users are finished.
DTS_RIGHTALIGN	0x20	+/-Right. The calendar will drop down on the right side of the control instead of the left.
(colors inside the drop-down calendar)		<p>The colors of the day numbers inside the drop-down calendar obey that set by the Gui Font command or the c (Color) option. To change the colors of other parts of the calendar, follow this example:</p> <pre> Gui +LastFound SendMessage, 0x1006, 4, 0xFFAA99, SysDateTimePick321 0x1006 is DTM_SETMCCOLOR. 4 is MCSC_MONTHBK (background color). The color must be specified in BGR vs. RGB format (red and blue components swapped).</pre>

MonthCal styles

default		WS_TABSTOP (only on [v1.1.22+], and only on Windows Vista and later)
forced		None
MCS_DAYSTATE	0x1	Makes the control send MCN_GETDAYSTATE notifications to request information about which days should be displayed in bold. [Not yet supported]
MCS_MULTISELECT	0x2	Named option "Multi". Allows the user to select a range of dates rather than being limited to a single date. By default, the maximum range is 366 days, which can be changed by sending the MCM_SETMAXSELCOUNT message to the control. For example: <pre>Gui +LastFound SendMessage, 0x1004, 7, 0, SysMonthCal321 ; 7 days. 0x1004 is MCM_SETMAXSELCOUNT.</pre>
MCS_WEEKNUMBERS	0x4	Displays week numbers (1-52) to the left of each row of days. Week 1 is defined as the first week that contains at least four days.
MCS_NOTODAYCIRCLE	0x8	Prevents the circling of today's date within the control.
MCS_NOTODAY	0x10	Prevents the display of today's date at the bottom of the control.

(colors)

The colors of the day numbers inside the calendar obey that set by the [Gui Font](#) command or the [c \(Color\)](#) option. To change the colors of other parts of the calendar, follow this example:

```
Gui +LastFound
SendMessage, 0x100A,
5, 0xFFAA99,
SysMonthCal321 ;
0x100A is
MCM_SETCOLOR. 5 is
MCSC_TITLETEXT
(color of title
text). The color
must be specified in
BGR vs. RGB format
(red and blue
components swapped).
```

Slider styles

default		WS_TABSTOP (+/-Tabstop)
forced		None
TBS_VERT	0x2	+/-Vertical. The control is oriented vertically.
TBS_LEFT	0x4	+/-Left. The control displays tick marks at the top of the control (or to its left if TBS_VERT is present). Same as TBS_TOP
TBS_TOP	0x4	same as TBS_LEFT .
TBS_BOTH	0x8	+/-Center. The control displays tick marks on both sides of the control. This will be both top and bottom when used with TBS_HORZ or both left and right if used with TBS_VERT .
TBS_AUTOTICKS	0x1	The control has a tick mark for each increment in its range of values. Use +/-TickInterval to have more flexibility.
		<p>The control displays a selection range only. The tick marks at the starting and ending positions of a selection range are displayed as triangles (instead of vertical dashes), and the selection range is highlighted (highlighting might require that the theme be removed via <code>Gui - theme</code>).</p> <p>To set the selection range, follow this</p>

TBS_ENABLESELRANGE	0x20	<p>example, which sets the starting position to 55 and the ending position to 66:</p> <pre>SendMessage, 1035, 1, 55, msctls_trackbar321, WinTitle SendMessage, 1036, 1, 66, msctls_trackbar321, WinTitle</pre>
TBS_FIXEDLENGTH	0x40	+/-Thick. Allows the thumb's size to be changed.
TBS_NOTHUMB	0x80	The control does not display the moveable bar.
TBS_NOTICKS	0x10	+/-NoTicks. The control does not display any tick marks.
TBS_TOOLTIPS	0x100	+/-ToolTip. The control supports ToolTips. When a control is created using this style, it automatically creates a default ToolTip control that displays the slider's current position. You can change where the ToolTips are displayed by using the TBM_SETTIPSIDE message.
TBS_REVERSED	0x200	Unfortunately, this style has no effect on the actual behavior of the control, so there is probably no point in using it (instead, use +Invert in the control's options to reverse it). Depending on OS version, this style might require Internet Explorer 5.0 or greater.

TBS_DOWNISLEFT	0x400	Unfortunately, this style has no effect on the actual behavior of the control, so there is probably no point in using it. Depending on OS version, this style might require Internet Explorer 5.01 or greater.
----------------	-------	--

Progress styles

default		PBS_SMOOTH
forced		None
PBS_SMOOTH	0x1	+/-Smooth. The progress bar displays progress status in a smooth scrolling bar instead of the default segmented bar. When this style is present, the control automatically reverts to the Classic Theme appearance on Windows XP or later.
PBS_VERTICAL	0x4	+/-Vertical. The progress bar displays progress status vertically, from bottom to top.
PBS_MARQUEE	0x8	<p>[Requires Windows XP or later] The progress bar moves like a marquee; that is, each change to its position causes the bar to slide further along its available length until it wraps around to the other side. A bar with this style has no defined position. Each attempt to change its position will instead slide the bar by one increment.</p> <p>This style is typically used to indicate an ongoing operation whose completion time is unknown.</p>

Tab styles

default		WS_TABSTOP and TCS_MULTILINE
forced		WS_CLIPSIBLINGS TCS_OWNERDRAWFIXED : forced on or off as required by the control's background color and/or text color.
TCS_SCROLLOPPOSITE	0x1	Unneeded tabs scroll to the opposite side of the control when a tab is selected.
TCS_BOTTOM	0x2	+/-Bottom. Tabs appear at the bottom of the control instead of the top.
TCS_RIGHT	0x2	Tabs appear vertically on the right side of controls that use the TCS_VERTICAL style.
TCS_MULTISELECT	0x4	Multiple tabs can be selected by holding down Ctrl  when clicking. This style must be used with the TCS_BUTTONS style.
TCS_FLATBUTTONS	0x8	Selected tabs appear as being indented into the background while other tabs appear as being on the same plane as the background. This style only affects tab controls with the TCS_BUTTONS style.
		Icons are aligned with the left edge of each fixed-width tab.

TCS_FORCEICONLEFT	0x10	This style can only be used with the TCS_FIXEDWIDTH style.
TCS_FORCELABELLEFT	0x20	Labels are aligned with the left edge of each fixed-width tab; that is, the label is displayed immediately to the right of the icon instead of being centered. This style can only be used with the TCS_FIXEDWIDTH style, and it implies the TCS_FORCEICONLEFT style.
TCS_HOTTRACK	0x40	Items under the pointer are automatically highlighted
TCS_VERTICAL	0x80	+/-Left or +/-Right. Tabs appear at the left side of the control, with tab text displayed vertically. This style is valid only when used with the TCS_MULTILINE style. To make tabs appear on the right side of the control, also use the TCS_RIGHT style. This style will not correctly display the tabs if a custom background color or text color is in effect. To workaround this, specify -Background and/or cDefault in the tab control's options.

TCS_BUTTONS	0x100	+/-Buttons. Tabs appear as buttons, and no border is drawn around the display area.
TCS_SINGLELINE	0	+/-Wrap. Only one row of tabs is displayed. The user can scroll to see more tabs, if necessary. This style is the default.
TCS_MULTILINE	0x200	+/-Wrap. Multiple rows of tabs are displayed, if necessary, so all tabs are visible at once.
TCS_RIGHTJUSTIFY	0	This is the default. The width of each tab is increased, if necessary, so that each row of tabs fills the entire width of the tab control. This window style is ignored unless the TCS_MULTILINE style is also specified.
TCS_FIXEDWIDTH	0x400	All tabs are the same width. This style cannot be combined with the TCS_RIGHTJUSTIFY style.
TCS_RAGGEDRIGHT	0x800	Rows of tabs will not be stretched to fill the entire width of the control. This style is the default.
TCS_FOCUSONBUTTONDOWN	0x1000	The tab control receives the input focus when clicked.
TCS_OWNERDRAWFIXED	0x2000	The parent window is

		responsible for drawing tabs.
TCS_TOOLTIPS	0x4000	The tab control has a tooltip control associated with it.
TCS_FOCUSNEVER	0x8000	The tab control does not receive the input focus when clicked.

StatusBar styles

default		SBARS_TOOLTIPS and SBARS_SIZEGRIP (the latter only if the window is resizable).
forced		None
SBARS_TOOLTIPS	0x800	<p>Displays a tooltip when the mouse hovers over a part of the status bar that: 1) has too much text to be fully displayed; or 2) has an icon but no text.</p> <p>The text of the ToolTip can be set via:</p> <pre>Gui +LastFound SendMessage, 0x410, 0, "Text to display", msctls_statusbar321</pre> <p>The bold 0 above is the zero-based part number. To use a part other than the first, specify 1 for second, 2 for the third, etc. NOTE: The ToolTip might never appear on certain OS versions.</p>
SBARS_SIZEGRIP	0x100	Includes a sizing grip at the right end of the status bar. A sizing grip is similar to a sizing border; it is a rectangular area that the user can click and drag to resize the parent window.

ToolTip

Creates an always-on-top window anywhere on the screen.

```
ToolTip [, Text, X, Y, WhichToolTip]
```

Parameters

Text

If blank or omitted, the existing tooltip (if any) 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`.

If *Text* is long, it can be broken up into several shorter lines by means of a [continuation section](#), which might improve readability and maintainability.

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. X and Y can be [expressions](#).

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). This parameter can be an [expression](#).

Remarks

If the X & Y coordinates would cause the tooltip to run off screen, it is repositioned to be entirely visible.

The tooltip is displayed until one of the following occurs:

- The script terminates.
- The ToolTip command is executed again with a blank *Text* parameter.
- The user clicks on the tooltip (this behavior may vary depending on operating system version).

A GUI window may be made the owner of a tooltip by means of [Gui+OwnDialogs](#). Such a tooltip is automatically destroyed when its owner is destroyed.

Related

[CoordMode](#), [TrayTip](#), [GUI](#), [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. On Windows 10, a toast notification may be shown instead.

```
TrayTip [, Title, Text, Seconds, Options]
```

Parameters

Title

The title of the window. Only the first 73 characters will be displayed.

If *Title* is blank, the title line will be entirely omitted from the window, making it vertically shorter.

Warning: The window will not be shown if the *Text* parameter is omitted, even if a *Title* is specified.

Text

The message to display. Only the first 265 characters will be displayed.

If this parameter is omitted or blank, any TrayTip balloon window currently displayed will be removed. However, to hide a Windows 10 toast notification it may be necessary to [temporarily remove the tray icon](#).

Carriage return (`\r`) or linefeed (`\n`) may be used to create multiple lines of text. For example: `Line1\nLine2`.

If *Text* is long, it can be broken up into several shorter lines by means of a [continuation section](#), which might improve readability and maintainability.

Seconds

Note: This parameter has no effect on Windows Vista and later.

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. This parameter can be an [expression](#).

The actual timeout may vary from the one specified. Microsoft explains, "if the user does not appear to be using the computer, the system does not count this time towards the timeout." (Technical details [here](#)). Therefore, 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

The *Options* parameter can be a combination (sum) of zero or more of the following values:

Function	Decimal Value	Hex Value
Info icon	1	0x1
Warning icon	2	0x2
Error icon	3	0x3
Windows XP and later: Do not play the notification sound.	16	0x10
Windows Vista and later: Use the large version of the icon.	32	0x20

If omitted, it defaults to 0, which is no icon. The icon is also not shown by the balloon window if it lacks a *Title* (this does not apply to Windows 10 toast notifications).

This parameter can be an [expression](#).

Remarks

Windows 10 replaces all balloon windows with toast notifications by default (this can be overridden via group policy). Calling TrayTip multiple times will usually cause multiple notifications to be placed in a "queue" instead of each notification replacing the last. To hide a notification, temporarily removing the tray icon may be effective. For example:

```
TrayTip #1, This is TrayTip #1
Sleep 3000 ; Let it display for 3 seconds.
HideTrayTip()
TrayTip #2, This is the second notification.
Sleep 3000

; Copy this function into your script to use it.
HideTrayTip() {
    TrayTip ; Attempt to hide it the normal
way.
    if SubStr(A_OSVersion,1,3) = "10." {
        Menu Tray, NoIcon
        Sleep 200 ; It may be necessary to
adjust this sleep.
        Menu Tray, Icon
    }
}
```

TrayTip has no effect if the script lacks a tray icon (via `#NoTrayIcon` or `Menu, Tray, NoIcon`). TrayTip also has no effect if the following REG_DWORD value exists and has been set to 0:

```
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced >> EnableBalloonTips
```

On a related note, there is a tooltip displayed whenever the user hovers the mouse over the script's tray icon. The contents of this tooltip can be changed via: `Menu, Tray, Tip, My New Text`.

Related

[ToolTip](#), [SetTimer](#), [Menu](#), [SplashTextOn](#), [MsgBox](#), [InputBox](#), [FileSelectFile](#), [FileSelectFolder](#)

Examples

```
TrayTip, My Title, 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, HideTrayTip, -5000  
  
HideTrayTip() { ; NOTE: For Windows 10, replace  
this function with the one defined above.  
    TrayTip  
}
```

```
; To have a TrayTip permanently displayed, use a  
timer to refresh it periodically.  
; NOTE: This probably won't work well on Windows  
10 for reasons described above.  
#Persistent  
SetTimer, RefreshTrayTip, 1000  
Gosub, RefreshTrayTip ; Call it once to get it  
started right away.  
return  
  
RefreshTrayTip:  
TrayTip, Refreshed TrayTip, This is a more  
permanent TrayTip., , 16  
return
```


Math Functions

Quick Reference:

- General Math:
 - Abs - Absolute value
 - Ceil - Rounding up
 - Exp - Exponential
 - Floor - Rounding down
 - Log - Decimal logarithm
 - Ln - Natural logarithm
 - Max - Highest value
 - Min - Lowest value
 - Mod - Remainder after division (Modulo)
 - Round - Rounding
 - Sqrt - Square root
- Trigonometry:
 - Sin - Sine
 - Cos - Cosine
 - Tan - Tangent
 - ASin - Arcsine
 - ACos - Arccosine
 - ATan - Arctangent
- Error-handling

General Math

Abs Absolute value

```
Value := Abs(Number)
```

Returns the absolute value of *Number*.

The return value is the same type as *Number* (integer or floating point).

```
MsgBox, % Abs(-1.2) ; Returns 1.2
```

Ceil Rounding up

```
Value := Ceil(Number)
```

Returns *Number* rounded up to the nearest integer (without any .00 suffix).

```
MsgBox, % Ceil(1.2) ; Returns 2  
MsgBox, % Ceil(-1.2) ; Returns -1
```

Exp Exponential

```
Value := Exp(N)
```

Returns e (which is approximately 2.71828182845905) raised to the N th power.

N may be negative and may contain a decimal point. To raise numbers other than e to a power, use the **** operator**.

```
MsgBox, % Exp(1.2) ; Returns 3.320117
```

Floor Rounding down

```
Value := Floor(Number)
```

Returns *Number* rounded down to the nearest integer (without any .00 suffix).

```
MsgBox, % Floor(1.2) ; Returns 1  
MsgBox, % Floor(-1.2) ; Returns -2
```

Log Decimal logarithm

```
Value := Log(Number)
```

Returns the logarithm (base 10) of *Number*.

The result is formatted as **floating point**. If *Number* is negative, an empty string is returned.

```
MsgBox, % Log(1.2) ; Returns 0.079181
```

Ln Natural logarithm

```
Value := Ln(Number)
```

Returns the natural logarithm (base e) of *Number*.

The result is formatted as **floating point**. If *Number* is negative, an empty string is returned.

```
MsgBox, % Ln(1.2) ; Returns 0.182322
```

Max Highest value [v1.1.27+]

```
Value := Max(Number1 [, Number2, ...])
```

Returns the highest value of one or more numbers.

If one of the input values is non-numeric, an empty string is returned.

```
MsgBox, % Max(2.11, -2, 0) ; Returns 2.11
```

You can also specify a [variadic parameter](#) to compare multiple values within an array. For example:

```
array := [1, 2, 3, 4]  
MsgBox, % Max(array*) ; Returns 4
```

Min Lowest value [v1.1.27+]

```
Value := Min(Number1 [, Number2, ...])
```

Returns the lowest value of one or more numbers.

If one of the input values is non-numeric, an empty string is returned.

```
MsgBox, % Min(2.11, -2, 0) ; Returns -2
```

You can also specify a [variadic parameter](#) to compare multiple values within an array. For example:

```
array := [1, 2, 3, 4]  
MsgBox, % Min(array*) ; Returns 1
```

Mod Remainder after division (Modulo)

```
Value := Mod(Dividend, Divisor)
```

Returns the remainder when *Dividend* is divided by *Divisor*.

The sign of the result is always the same as the sign of the first parameter. If either input is a floating point number, the result is also a floating point number. If the second parameter is zero, the function yields a blank result (empty string).

```
MsgBox, % Mod(7.5, 2) ; Returns 1.5 (2 x 3 + 1.5)
```

Round Rounding

```
Value := Round(Number [, N])
```

Returns *Number* rounded to *N* decimal places.

If *N* is omitted or 0, *Number* is rounded to the nearest integer:

```
MsgBox, % Round(3.14) ; Returns 3
```

If *N* is positive number, *Number* is rounded to *N* decimal places:

```
MsgBox, % Round(3.14, 1) ; Returns 3.1
```

If *N* is negative, *Number* is rounded by *N* digits to the left of the decimal point:

```
MsgBox, % Round(345, -1) ; Returns 350  
MsgBox, % Round(345, -2) ; Returns 300
```

Unlike [Transform Round](#), the result has no .000 suffix whenever *N* is omitted or less than 1. [v1.0.44.01+]: A value of *N* greater than zero displays exactly *N* decimal places rather than obeying [SetFormat](#). To avoid this, perform another math operation on Round()'s return value; for example:

```
Round(3.333, 1)+0 .
```

Sqrt Square root

```
Value := Sqrt(Number)
```

Returns the square root of *Number*.

The result is formatted as **floating point**. If *Number* is negative, the function yields a blank result (empty string).

```
MsgBox, % Sqrt(16) ; Returns 4
```

Trigonometry

Note: 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 π (approximately 3.141592653589793) is 4 times the arctangent of 1.

Sin Sine

```
Value := Sin(Number)
```

Returns the trigonometric sine of *Number*.

Number must be expressed in radians.

```
MsgBox, % Sin(1.2) ; Returns 0.932039
```

Cos Cosine

```
Value := Cos(Number)
```

Returns the trigonometric cosine of *Number*.

Number must be expressed in radians.

```
MsgBox, % Cos(1.2) ; Returns 0.362358
```

Tan Tangent

```
Value := Tan(Number)
```

Returns the trigonometric tangent of *Number*.

Number must be expressed in radians.

```
MsgBox, % Tan(1.2) ; Returns 2.572152
```

ASin Arcsine

```
Value := ASin(Number)
```

Returns the arcsine (the number whose sine is *Number*) in radians.

If *Number* is less than -1 or greater than 1, the function yields a blank result (empty string).

```
MsgBox, % ASin(0.2) ; Returns 0.201358
```

ACos Arccosine

```
Value := ACos(Number)
```

Returns the arccosine (the number whose cosine is *Number*) in radians.

If *Number* is less than -1 or greater than 1, the function yields a blank result (empty string).

```
MsgBox, % ACos(0.2) ; Returns 1.369438
```

ATan Arctangent

```
Value := ATan(Number)
```

Returns the arctangent (the number whose tangent is *Number*) in radians.

```
MsgBox, % ATan(1.2) ; Returns 0.876058
```

Error-Handling

Invalid operations such as divide by zero generally yield a blank result (empty string).

[Abs](#), [Max](#), [Min](#) and [Mod](#) return an empty string if any of their incoming parameters are non-numeric. Most math functions do not perform strict type-checking, so may treat non-numeric values as zero or another number. For example, `Round("1.0foo")` produces 1. However, this is expected to change in [AutoHotkey v2](#).

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](#) upon which to operate.

Value

▮ Any integer, floating point number, or [expression](#).

TimeUnits

If present, this parameter directs the command to add *Value* to *Var*, treating *Var* as a date-time stamp in the [YYYYMMDDHH24MISS](#) format and treating *Value* as the integer or floating point number of units to add (specify a negative number to perform subtraction). *TimeUnits* can be either Seconds, Minutes, Hours, or Days (or just the first letter of each of these).

If *Var* is an empty variable, the current time will be used in its place. If *Var* contains an invalid timestamp or a year prior to 1601, or if *Value* is non-numeric, *Var* will be made blank to indicate the problem.

The built-in variable **A_Now** contains the current local time in [YYYYMMDDHH24MISS](#) format.

To calculate the amount of time between two timestamps, use [EnvSub](#).

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 used *internally* in an expression and except when using the *TimeUnits* parameter).

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

[EnvSub](#), [EnvMult](#), [EnvDiv](#), [SetFormat](#), [Expressions](#), [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.

Deprecated: This command is not recommended for use in new scripts. Use `Var := Var / Value` or `Var /= Value` instead.

`EnvDiv, Var, Value`

Parameters

Var

The name of the [variable](#) upon which to operate.

Value

Any integer, floating point number, or [expression](#).

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); otherwise it makes the variable blank.

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 used *internally* in an expression such as `Var := X /= Y`).

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](#), [Expressions](#), [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.

Deprecated: This command is not recommended for use in new scripts. Use `Var := Var * Value` or `Var *= Value` instead.

EnvMult, Var, Value

Parameters

Var

▮ The name of the [variable](#) upon which to operate.

Value

▮ Any integer, floating point number, or [expression](#).

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 (except when used *internally* in an expression such as `Var := X *= Y`).

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](#), [Expressions](#), [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](#) upon which to operate.

Value

Any integer, floating point number, or [expression](#) (expressions are not supported when *TimeUnits* is present).

TimeUnits

If present, this parameter directs the command to subtract *Value* from *Var* as though both of them are date-time stamps in the [YYYYMMDDHH24MISS](#) format. *TimeUnits* can be either Seconds, Minutes, Hours, or Days (or just the first letter of each of these). If *Value* is blank, the current time will be used in its place. Similarly, if *Var* is an empty variable, the current time will be used in its place.

The result is always rounded *down* to the nearest integer. For example, if the actual difference between two timestamps is 1.999 days, it will be reported as 1 day. If higher precision is needed, specify Seconds for *TimeUnits* and divide the result by 60.0, 3600.0, or 86400.0.

If either *Var* or *Value* is an invalid timestamp or contains a year prior to 1601, *Var* will be made blank to indicate the problem.

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

[YYYYMMDDHH24MISS](#) format.

To precisely determine the elapsed time between two events, use the [A_TickCount](#) method because it provides millisecond precision.

To add or subtract a certain number of seconds, minutes, hours, or days from a timestamp, use [EnvAdd](#) (subtraction is achieved by adding a negative number).

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 used *internally* in an expression and except when using the *TimeUnits* parameter).

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](#), [Expressions](#), [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.
```


Random

Generates a pseudo-random number.

```
Random, OutputVar [, Min, Max]  
Random, , NewSeed
```

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, which can be negative, floating point, or an [expression](#). 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, which can be negative, floating point, or an [expression](#). If omitted, the largest number will be 2147483647 (which is also the largest allowed integer value -- but floating point numbers have no restrictions).

NewSeed

This mode reseeds the random number generator with *NewSeed* (which can be an [expression](#)). This affects all subsequently generated random numbers. *NewSeed* should be an integer between 0 and 4294967295 (0xFFFFFFFF). Reseeding can improve the quality/security of generated random numbers, especially when *NewSeed* is a genuine random number rather than one of lesser quality such as a pseudo-random number. Generally, reseeding does not need to be done more than once.

If reseeding is never done by the script, the seed starts off as the low-

order 32-bits of the 64-bit value that is the number of 100-nanosecond intervals since January 1, 1601. This value travels from 0 to 4294967295 every ~7.2 minutes.

Remarks

This command yields 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.

All numbers within the specified range have approximately the same probability of being generated (however, see "known limitations" below).

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.

Known limitations for floating point: 1) only about 4,294,967,296 distinct numbers can be generated for any particular range, so all other numbers in the range will never be generated; 2) occasionally a result can be slightly greater than the specified *Max* (this is caused in part by the imprecision inherent in floating point numbers).

Related

[SetFormat](#)

Examples

```
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 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.

When you use this, send an email to: matumoto@math.keio.ac.jp with an appropriate reference to your work. It would be nice to CC: rjwagner@writeme.com and Cokus@math.washington.edu when you write.

This above has been already been done for AutoHotkey, but if you use the Random command in a publicly distributed application, consider sending an e-mail to the above people to thank them.

SetFormat

Sets the format of integers and floating point numbers generated by math operations.

Deprecated: This command is not recommended for use in new scripts. Use the [Format](#) function instead.

SetFormat, **NumberType**, **Format**

Parameters

NumberType

Must be either IntegerFast, FloatFast, Integer, or Float (the two fast modes require [v1.0.48+]; see [remarks](#)).

Format

For *NumberType* Integer or IntegerFast, specify H or HEX for hexadecimal, or D for decimal. Hexadecimal numbers all start with the prefix 0x (e.g. 0xFF). [AHK_L 42+]: Hexadecimal integers are formatted with digits A-F in lowercase when this parameter is **h** and uppercase when it is **H**.

For *NumberType* Float or FloatFast, specify **TotalWidth.DecimalPlaces** (e.g. **0.6**). [v1.0.46.11+]: The letter "e" may be appended to produce scientific notation; e.g. **0.6e** or **0.6E** (using uppercase produces an uppercase E in each number instead of lowercase). Note: In AutoHotkey 1.x, scientific notation must include a decimal point; e.g. **1.0e1** is valid but not **1e1**.

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 [remarks](#)) 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, floating point results are displayed as integers rather than a floating point number. The starting default is 6.

Padding: If *TotalWidth* is high enough to cause padding, spaces will be added on the left side; that is, each number 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 (e.g. `06.2`).

Fast vs. Slow Mode

[v1.0.48+]: IntegerFast may be used instead of Integer, and FloatFast may be used instead of Float.

- **Advantages:** The fast mode preserves the ability of variables to [cache](#) integers and floating point numbers, which substantially accelerates numerically-intensive operations. (By contrast, the slow mode forces all numeric results to be immediately converted and stored as strings.)
- **Disadvantages:** When storing a number in a variable, the fast mode delays the effects of SetFormat until the script actually needs a text/string version of that variable (e.g. to display in a [MsgBox](#)). Since a different SetFormat may be in effect at that time (e.g. more or fewer decimal places), this can lead to unexpected results. To make the current fast format take effect immediately, use an operation like `HexValue .= ""`, which appends an empty string to the number currently stored in *HexValue*.

If the slow mode "Integer" or "Float" is used anywhere in the script, even if that SetFormat line is never executed, the [caching](#) of integers or floating point numbers (respectively) is disabled the moment the script launches.

Floating Point Format

[v1.0.48+]: Floating point variables have about 15 digits of precision internally unless `SetFormat Float` (i.e. the slow mode) is present anywhere in the script. In that case, the stored precision of floating point numbers is determined by [DecimalPlaces](#) (like it was in pre-1.0.48 versions). In other words, once a floating point result is stored in a variable, the extra precision is lost and cannot be reclaimed without redoing the calculation with something like

`SetFormat, Float, 0.15`. To avoid this loss of precision, avoid using `SetFormat Float` anywhere in the script, or use `SetFormat FloatFast` instead.

Regardless of whether slow or fast mode is in effect, floating point results and variables are rounded off to *DecimalPlaces* whenever they are displayed or converted to a string of text (e.g. `MsgBox` or `FileAppend`). To see the full precision, use something like `SetFormat, FloatFast, 0.15`.

To convert a floating point number to an integer, use `Var:=Round(Var)`, `Var:=Floor(Var)`, or `Var:=Ceil(Var)`. To convert an integer to a floating point number, add 0.0 to it (e.g. `Var+=0.0`) or use something like `MyFloat:=Round(MyInteger, 1)`.

The built-in variable **A_FormatFloat** contains the current floating point format (e.g. `0.6`).

Integer Format

Integer results are normally displayed as decimal, not hexadecimal. To switch to hexadecimal, use `SetFormat, IntegerFast, Hex`. This may also be used to convert an integer from decimal to hexadecimal (or vice versa) as shown in the example at the very bottom of this page.

Literal integers specified in a script may be written as either hexadecimal or decimal. Hexadecimal integers all start with the prefix `0x` (e.g. `0xA9`). They may 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`.

AutoHotkey supports 64-bit signed integers, which range from -9223372036854775808 (-0x8000000000000000) to 9223372036854775807 (0x7FFFFFFFFFFFFFFF).

The built-in variable **A_FormatInteger** contains the current integer format (H or D).

General Remarks

If `SetFormat` is not used in a script, integers default to decimal format, and floating point numbers default to `TotalWidth.DecimalPlaces = 0.6`. Every newly launched [thread](#) (such as a [hotkey](#), [custom menu item](#), or [timed subroutine](#)) starts off fresh with these defaults; but the defaults may be changed by using `SetFormat` in the auto-execute section (top part of the script).

An old-style assignment like `x=%y%` omits any leading or trailing spaces (i.e. padding). To avoid this, use [AutoTrim](#) or the [colon-equals operator](#) (e.g. `x:=y`).

You can determine whether a variable contains a numeric value by using `"if var is number/integer/float"`

To pad an integer with zeros or spaces without having to use floating point math on it, follow this example:

```
Var := "          " . Var ; The quotes
contain 10 spaces. To pad with zeros,
substitute zeros for the spaces.
StringRight, Var, Var, 10 ; This pads the
number in Var with enough spaces to make its
total width 10 characters.
Var := SubStr("          " . Var, -9) ; A
single-line alternative to the above two lines.
```

Related

[Format\(\)](#), [Caching](#), [Expression assignment \(:=\)](#), [EnvAdd](#), [EnvSub](#), [EnvMult](#), [EnvDiv](#), [AutoTrim](#), [if var is type](#)

Examples

```
Var = 11.333333
SetFormat, float, 6.2
Var -= 1 ; Sets Var to be 10.33 with one leading
space because the total width is 6.
```

```
SetFormat, float, 0.2
Var += 1 ; Sets Var to be 11.33 with no leading
spaces.
SetFormat, float, 06.0
Var += 0 ; Sets Var to be 000011

; Convert a decimal integer to hexadecimal:
SetFormat, IntegerFast, hex
Var += 0 ; Sets Var (which previously contained
11) to be 0xb.
Var .= "" ; Necessary due to the "fast" mode.
SetFormat, IntegerFast, d
```

Transform

Performs miscellaneous math functions, bitwise operations, and tasks such as ASCII/Unicode conversion.

```
Transform, OutputVar, Cmd, Value1 [, Value2]
```

Parameters

OutputVar

The name of the variable in which to store the result of *Cmd*. [SetFormat](#) determines whether integers are stored as hexadecimal or decimal.

Cmd, Value1/2

See list below.

Cmd, Value1, Value2

The *Cmd*, *Value1* and *Value2* parameters are dependent upon each other and their usage is described below.

Unicode [, String]: *(This command is not available in Unicode versions of AutoHotkey.)* Retrieves or stores Unicode text on the clipboard. Note: The entire clipboard may be saved and restored by means of [ClipboardAll](#), which allows "Transform Unicode" to operate without losing the original contents of the clipboard.

There are two modes of operation as illustrated in the following examples:

```
Transform, OutputVar, Unicode ; Retrieves the
clipboard's Unicode text as a UTF-8 string.
Transform, Clipboard, Unicode, %MyUTF_String% ;
Places Unicode text onto the clipboard.
```

In the second example above, a literal UTF-8 string may be optionally used in place of %MyUTF_String%.

Use a hotkey such as the following to determine the UTF-8 string that corresponds to a given Unicode string:

```
^!u:: ; Control+Alt+U hotkey.
MsgBox Copy some Unicode text onto the
clipboard, then return to this window and press
OK to continue.
Transform, ClipUTF, Unicode
Clipboard = Transform, Clipboard, Unicode,
%ClipUTF%`r`n
MsgBox 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%
return
```

Note: The `Send {U+nnnn}` command is an alternate way to produce Unicode characters.

Asc, String: Retrieves the character code (a number between 1 and 255, or 1 and 65535 if Unicode is supported) for the first character in *String*. If *String* is empty, *OutputVar* will also be made empty. For example: `Transform, OutputVar, Asc, %VarContainingString%`. Corresponding function: `Asc(String)`. To allow for Unicode supplementary characters, use `Ord(String)` instead.

Chr, Value1: Retrieves the single character corresponding to the character code indicated by *Value1*. If *Value1* is not between 1 and 255 inclusive (or 1 and 65535 if Unicode is supported), *OutputVar* will be made blank to indicate the problem. For example: `Transform, OutputVar, Chr, 130`. Corresponding function: `Chr(Number)`. Unlike `Chr()`, `Transform` does not support Unicode supplementary characters (character codes 0x10000 to

0x10FFFF).

Deref, String: 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%`. Within a [function](#), each variable in *String* always resolves to a local variable unless there is no such variable, in which case it resolves to a global variable (or blank if none).

HTML, String [, Flags]:

For ANSI executables: 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). The *Flags* parameter is ignored.

For Unicode executables: In addition of the functionality above, *Flags* can be zero or a combination (sum) of the following values. If omitted, it defaults to 1.

- 1: Converts certain characters to named expressions. e.g. € is converted to `€`
- 2: Converts certain characters to numbered expressions. e.g. € is converted to `€`

Only non-ASCII characters are affected. If *Flags* = 3, numbered expressions are used only where a named expression is not available. The following characters are always converted: `<>"&` and `\n` (line feed).

FromCodePage / ToCodePage: Deprecated. Use [StrPut\(\)](#) / [StrGet\(\)](#) instead.

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`. Corresponding function: [Mod\(Dividend, Divisor\)](#).

Pow, Base, Exponent: Retrieves *Base* raised to the power of *Exponent*. Both *Base* and *Exponent* may contain a decimal point. If *Exponent* is negative, *OutputVar* will be formatted as a floating point number even if *Base* and *Exponent* are both integers. A negative *Base* combined with a fractional *Exponent* such as 1.5 is not supported; it will cause *OutputVar* to be made blank. See also: [** operator](#).

Exp, N: Retrieves e (which is approximately 2.71828182845905) raised to the *N*th power. *N* may be negative and may contain a decimal point. Corresponding function: [Exp\(N\)](#).

Sqrt, Value1: Retrieves the square root of *Value1*. If *Value1* is negative, *OutputVar* will be made blank. Corresponding function: [Sqrt\(Number\)](#).

Log, Value1: Retrieves the logarithm (base 10) of *Value1*. If *Value1* is negative, *OutputVar* will be made blank. Corresponding function: [Log\(Number\)](#).

Ln, Value1: Retrieves the natural logarithm (base e) of *Value1*. If *Value1* is negative, *OutputVar* will be made blank. Corresponding function: [Ln\(Number\)](#).

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, and -3 rounds to the hundreds place. Note: Round does not remove trailing zeros when rounding decimal places. For example, 12.333 rounded to one decimal place would become 12.300000. This behavior can be altered by using something like [SetFormat, Float, 0.1](#) prior to the operation (in fact, [SetFormat](#) might eliminate the need to use Round in the first place). Corresponding function: [Round\(Number \[, N\]\)](#).

Ceil, Value1: Retrieves *Value1* rounded up to the nearest integer. Corresponding function: [Ceil\(Number\)](#).

Floor, Value1: Retrieves *Value1* rounded down to the nearest integer. Corresponding function: [Floor\(Number\)](#).

Abs, Value1: Retrieves the absolute value of *Value1*, which is computed by removing the leading minus sign (dash) from *Value1* if it has one. Corresponding function: [Abs\(Number\)](#).

Sin, Value1: Retrieves the trigonometric sine of *Value1*. *Value1* must be expressed in radians. Corresponding function: [Sin\(Number\)](#).

Cos, Value1: Retrieves the trigonometric cosine of *Value1*. *Value1* must be expressed in radians. Corresponding function: [Cos\(Number\)](#).

Tan, Value1: Retrieves the trigonometric tangent of *Value1*. *Value1* must be expressed in radians. Corresponding function: [Tan\(Number\)](#).

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. Corresponding function: [ASin\(Number\)](#).

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. Corresponding function: [ACos\(Number\)](#).

ATan, Value1: Retrieves the arctangent (the number whose tangent is *Value1*) in radians. Corresponding function: [ATan\(Number\)](#).

Note: Each of the following bitwise operations has a more concise [bitwise operator](#) for use in expressions.

BitNot, Value1: Stores the bit-inverted version of *Value1* in *OutputVar* (if *Value1* is floating point, it is truncated to an integer prior to the calculation). 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* (floating point values are truncated to integers prior to the calculation). 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* (floating point values are truncated to integers prior to the calculation). 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* (floating point values are truncated to integers prior to the calculation). 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" (floating point values are truncated to integers prior to the calculation). 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 (floating point values are truncated to integers prior to the calculation). In the following example, the result is 2: `Transform, OutputVar, BitShiftRight, 17, 3`.

Remarks

Sub-commands that accept numeric parameters can also use [expressions](#) for those parameters.

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](#), [Expressions](#), [EnvMult](#), [EnvDiv](#), [StringLower](#), [if var is type](#)

Example

```
Transform, OutputVar, Asc, A ; Get the ASCII code  
of the letter A.
```

#HotkeyInterval

Along with [#MaxHotkeysPerInterval](#), specifies the rate of [hotkey](#) activations beyond which a warning dialog will be displayed.

```
#HotkeyInterval Milliseconds
```

Parameters

Milliseconds

- ▮ The length of the interval in milliseconds.

Remarks

If this directive is unspecified in the script, it will behave as though set to 2000.

For details and remarks, see [#MaxHotkeysPerInterval](#).

Related

[#MaxHotkeysPerInterval](#)

Example

```
#HotkeyInterval 2000 ; This is the default value  
(milliseconds).  
#MaxHotkeysPerInterval 200
```

#HotkeyModifierTimeout

Affects the behavior of [hotkey](#) modifiers:    

#HotkeyModifierTimeout Milliseconds

Parameters

Milliseconds

The length of the interval in milliseconds. The value can be -1 so that it never times out (modifier keys are always pushed back down after the Send), or 0 so that it always times out (modifier keys are never pushed back down).

Remarks

This directive **need not** be used when:

- Hotkeys send their keystrokes via the [SendInput](#) or [SendPlay](#) methods. This is because those methods postpone the user's physical pressing and releasing of keys until after the Send completes.
- 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 (    is physically holding down and doesn't need to use the timeout.

To illustrate the effect of this directive, consider this example: `^!a::Send, abc`.

When the [Send](#) command executes, the first thing it does is release   the characters get sent properly. After sending all the keys, the command doesn't know whether it can safely push back down 

`Alt` [redacted] (to check 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 reduce or eliminate the need for this directive with one of the following:

- Install the keyboard hook by adding the line `#InstallKeybdHook` anywhere in the script.
- Use the `SendInput` or `SendPlay` methods rather than the traditional `SendEvent` method.
- When using the traditional `SendEvent` method, reduce `SetKeyDelay` to 0 or -1, which should help because it sends the keystrokes more quickly.

If this directive is unspecified in a script, it behaves as though set to 50.

Related

[GetKeyState](#)

Example

```
#HotkeyModifierTimeout 100
```

#Hotstring

Changes [hotstring](#) options or ending characters.

```
#Hotstring NoMouse
#Hotstring EndChars NewChars
#Hotstring NewOptions
```

Parameters

NoMouse

Prevents mouse clicks from resetting the hotstring recognizer as described [here](#). As a side-effect, this also prevents the [mouse hook](#) from being required by hotstrings (though it will still be installed if the script requires it for other purposes, such as mouse hotkeys). The presence of `#Hotstring NoMouse` anywhere in the script affects all hotstrings, not just those physically beneath it.

EndChars NewChars

Specify the word EndChars followed a single space and then the new ending characters. For example:

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

Since the new ending characters are in effect globally for the entire script -- regardless of where the EndChars directive 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 only one other character, or no other

characters).

NewOptions

Specify new options as described in [Hotstring Options](#). For example:

```
#Hotstring r s k0 c0.
```

Unlike *EndChars* 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:

```
::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.
```

Related

[Hotstrings](#)

[v1.1.28+]: The [Hotstring](#) function can be used to change hotstring options while the script is running.

#If [AHK_L 4+]

Creates context-sensitive [hotkeys](#) and [hotstrings](#). Such hotkeys perform a different action (or none at all) depending on the result of an expression.

```
#If [Expression]
```

Parameters

Expression

Any valid [expression](#).

Basic Operation

Any valid expression may be used to define the context in which a hotkey should be active. For example:

```
#If WinActive("ahk_class Notepad") or  
WinActive(MyWindowTitle)  
#Space::MsgBox You pressed Win+Spacebar in  
Notepad or %MyWindowTitle%.
```

Like the `#IfWin` directives, `#If` is positional: it affects all hotkeys and hotstrings physically beneath it in the script. `#If` and `#IfWin` are also mutually exclusive; that is, only the most recent `#If` or `#IfWin` will be in effect.

To turn off context sensitivity, specify `#If` or any `#IfWin` directive but omit all the parameters. For example:

```
#If
```

General Remarks

When the key, mouse or joystick button combination which forms a hotkey is pressed, the #If expression is evaluated to determine if the hotkey should activate.

Note: Scripts should not assume that the expression is only evaluated when the key is pressed (see below).

The expression may also be evaluated whenever the program needs to know whether the hotkey is active. For example, the #If expression for a custom combination like `a & b : :` might be evaluated when the prefix key (`a` in this example) is pressed, to determine whether it should act as a custom modifier key.

Note: Use of #If in an unresponsive script may cause input lag or break hotkeys (see below).

There are several more caveats to the #If directive:

- Keyboard or mouse input is typically buffered (delayed) until expression evaluation completes or [times out](#).
- Expression evaluation can only be performed by the script's main thread (at the OS level, not a [quasi-thread](#)), not directly by the keyboard/mouse hook. If the script is busy or unresponsive, such as if a FileCopy is in progress, expression evaluation is delayed and may time out.
- If the [system-defined timeout](#) is reached, the system may stop notifying the script of keyboard or mouse input (see #IfTimeout for details).
- Sending keystrokes or mouse clicks while the expression is being evaluated (such as from a function which it calls) may cause complications and should be avoided.

[AHK_L 53+]: [A_ThisHotkey](#) and [A_TimeSinceThisHotkey](#) are set based on the hotkey for which the current #If expression is being evaluated.

[v1.0.95.00+]: [A_PriorHotkey](#) and [A_TimeSincePriorHotkey](#) temporarily contain the previous values of the corresponding "This" variables.

Related

Most behavioural properties of the [#IfWin](#) directives also apply to #If.

`#IfTimeout` may be used to override the default timeout value.

Examples

```
; Example 1: Adjust volume by scrolling the mouse  
wheel over the taskbar.
```

```
#If MouseIsOver("ahk_class Shell_TrayWnd")  
WheelUp::Send {Volume_Up}  
WheelDown::Send {Volume_Down}
```

```
MouseIsOver(WinTitle) {  
    MouseGetPos,,, Win  
    return WinExist(WinTitle . " ahk_id " . Win)  
}
```

```
; Example 2: Simple word-delete shortcuts for all  
Edit controls.
```

```
#If ActiveControlIsOfClass("Edit")  
^BS::Send ^+{Left}{Del}  
^Del::Send ^+{Right}{Del}
```

```
ActiveControlIsOfClass(Class) {  
    ControlGetFocus, FocusedControl, A  
    ControlGet, FocusedControlHwnd, Hwnd,,  
    %FocusedControl%, A  
    WinGetClass, FocusedControlClass, ahk_id  
    %FocusedControlHwnd%  
    return (FocusedControlClass=Class)  
}
```

```
; Example 3: Context-insensitive hotkey.
```

```
#If  
Esc::ExitApp
```

```
; Example 4: Dynamic hotkeys. Requires Example 1.
```

```
NumpadAdd::
```

```
Hotkey, If, MouseIsOver("ahk_class Shell_TrayWnd")
if (doubleup := !doubleup)
    Hotkey, WheelUp, DoubleUp
else
    Hotkey, WheelUp, WheelUp
return

DoubleUp:
Send {Volume_Up 2}
return
```

#IfTimeout [AHK_L 8+]

Sets the maximum time that may be spent evaluating a single `#If` expression.

```
#IfTimeout Timeout
```

Parameters

Timeout

▮ The timeout value to apply globally, in milliseconds.

Remarks

A timeout is implemented to prevent long-running expressions from stalling keyboard input processing. If the timeout value is exceeded, the expression continues to evaluate, but the keyboard hook continues as if the expression had already returned false.

If this directive is unspecified in the script, it will behave as though set to 1000.

Note that the system implements its own timeout, defined by the DWORD value *LowLevelHooksTimeout* in the following registry key:

HKEY_CURRENT_USER\Control Panel\Desktop

If the system timeout value is exceeded, the system may stop calling the script's keyboard hook, thereby preventing hook hotkeys from working until the hook is re-registered or the script is [reloaded](#). The hook can *usually* be re-registered by [suspending](#) and un-suspending all hotkeys.

Microsoft's documentation is unclear about the details of this timeout, but research indicates the following for Windows 7 and later: If *LowLevelHooksTimeout* is not defined, the default timeout is 300ms. The hook may time out up to 10 times, but is silently removed if it times out an 11th time.

If a given hotkey has multiple `#If` variants, the timeout might be applied to each

variant independently, making it more likely that the system timeout will be exceeded. This may be changed in a future update.

Related

[#If](#)

Example

```
#IfTimeout 10 ; Set the timeout to 10 ms.
```

#IfWin[Not]Active / #IfWin[Not]Exist

Creates context-sensitive [hotkeys](#) and [hotstrings](#). Such hotkeys perform a different action (or none at all) depending on the type of window that is active or exists.

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

Parameters

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

Title matching behaviour is determined by [SetTitleMatchMode](#) as set in the [auto-execute section](#).

As with most other directives, variables are not supported. Although [ahk_pid](#) and [ahk_id](#) can be used with a hard-coded process or window ID, it is more common for #IfWin to use them indirectly via [GroupAdd](#) or [Hotkey IfWin](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). Hidden text elements are detected if [DetectHiddenText](#) has been turned on in the auto-execute section (top part of the script).

ExcludeTitle

ExcludeText

Although these are **not** supported, they can be used indirectly by specifying `ahk_group MyGroup` for `WinTitle` (where `MyGroup` is a group created via [GroupAdd](#), which supports `ExcludeTitle/Text`).

Basic Operation

The `#IfWin` directives make it easy to create context-sensitive [hotkeys](#) and [hotstrings](#). For example:

```
#IfWinActive ahk_class Notepad
#space::MsgBox You pressed Win+Spacebar in
Notepad.
```

The `#IfWin` directives are positional: they affect all hotkeys and hotstrings physically beneath them in the script. They are also mutually exclusive; that is, only the most recent one will be in effect.

To turn off context sensitivity, specify any `#IfWin` directive but omit all of its parameters. For example:

```
#IfWinActive
```

When `#IfWin` is turned off (or never used in a script), all [hotkeys](#) and [hotstrings](#) are enabled for all windows (unless disabled via [Suspend](#) or the [Hotkey command](#)).

When a mouse or keyboard hotkey is disabled via `#IfWin`, it performs its native function; that is, it passes through to the active window as though there is no such hotkey. There is one exception: Joystick hotkeys: although `#IfWin` works, it never prevents other programs from seeing the press of a button.

`#IfWin` can also be used to alter the behavior of an ordinary key like `Enter` or `Space` useful when a particular window ignores that key or performs some action you find undesirable. For example:

```
#IfWinActive Reminders ahk_class #32770 ; The
```

```
"reminders" window in Outlook.  
Enter::Send !o ; Have an "Enter" keystroke open  
the selected reminder rather than snoozing it.  
#IfWinActive
```

Variant (Duplicate) Hotkeys

A particular [hotkey](#) or [hotstring](#) can be defined more than once in the script if each definition has different `#IfWin` criteria. These are known as *hotkey variants*. For example:

```
#IfWinActive ahk_class Notepad  
^!c::MsgBox You pressed Control+Alt+C in  
Notepad.  
#IfWinActive ahk_class WordPadClass  
^!c::MsgBox You pressed Control+Alt+C in  
WordPad.  
#IfWinActive  
^!c::MsgBox You pressed Control+Alt+C in a  
window other than Notepad/WordPad.
```

If more than one variant is eligible to fire, only the one closest to the top of the script will fire. The exception to this is the global variant (the one with no `#IfWin` criteria): It always has the lowest precedence; therefore, it will fire only if no other variant is eligible (this exception does not apply to [hotstrings](#)).

When creating duplicate hotkeys, the order of [modifier symbols](#) such as `^!+#` does not matter. For example, `^!C` is the same as `!^C`. However, keys must be spelled consistently. For example, *Esc* is not the same as *Escape* for this purpose (though the case does not matter). Also, any hotkey with a [wildcard prefix \(*\)](#) is entirely separate from a non-wildcard one; for example, `*F1` and `F1` would each have their own set of variants.

To have the same hotkey subroutine executed by more than one variant, the easiest way is to create a stack of identical hotkeys, each with a different `#IfWin` directive above it. For example:

```
#IfWinActive ahk_class Notepad
#Z::
#IfWinActive ahk_class WordPadClass
#Z::
MsgBox You pressed Win+Z in either Notepad or
WordPad.
return
```

Alternatively, a [window group](#) can be used via `#IfWinActive ahk_group MyGroup`.

To create hotkey variants dynamically (while the script is running), see "[Hotkey IfWin](#)".

General Remarks

#IfWin also restores prefix keys to their native function when appropriate (a [prefix key](#) is the  in a hotkey such as "a & b"). This occurs whenever there are no enabled hotkeys for a given prefix.

When Gosub or Goto is used to jump to a hotkey or hotstring label, it jumps to the variant closest to the top of the script.

When a hotkey is currently disabled via #IfWin, its key or mouse button will appear with a "#" character in [KeyHistory's](#) "Type" column. This can help debug a script.

Variable references such as %Var% are not currently supported. Therefore, percent signs must be [escaped](#) via `% to allow future support for them. Similarly, literal commas must be escaped (via `,) to allow additional parameters to be added in the future. If you need to work around this limitation, use [GroupAdd](#) and [ahk_group](#).

A label to which the [Hotkey command](#) has assigned a hotkey is not directly affected by #IfWin. Instead, the use of #IfWin closest to the bottom of the script (if any) will be in effect for all hotkeys created by the Hotkey command (unless "[Hotkey IfWin](#)" has been used to change that).

[Alt-tab hotkeys](#) are not affected by #IfWin: they are in effect for all windows.

The [Last Found Window](#) is set by #IfWinActive/Exist (though not by #IfWinNotActive/NotExist). For example:

```
#IfWinExist ahk_class Notepad
#n::WinActivate ; Activates the window found by
#IfWin.
```

The [escape sequences](#) `s and `t may be used if leading or trailing spaces/tabs are needed in one of #IfWin's parameters.

For performance reasons, #IfWin does not continuously monitor the activation or existence of the specified windows. Instead, it checks for a matching window only when you type a hotkey or hotstring. If a matching window is not present, your keystroke or mouse click is allowed to pass through to the active window unaltered.

Window titles and text are case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on in the auto-execute section (top part of the script).

Related

[#If expression](#), [Hotkey command](#), [Hotkeys](#), [Hotstrings](#), [Suspend](#), [IfWinActive](#), [IfWinExist](#), [SetTitleMatchMode](#), [DetectHiddenWindows](#)

Examples

```
#IfWinActive ahk_class Notepad
^!a::MsgBox You pressed Ctrl-Alt-A while Notepad
is active. ; This hotkey will have no effect if
pressed in other windows (and it will "pass
through").
#c::MsgBox You pressed Win-C while Notepad is
active.
::btw::This replacement text for "btw" will occur
```

only in Notepad.

```
#IfWinActive
```

```
#c::MsgBox You pressed Win-C in a window other  
than Notepad.
```

#InputLevel [v1.1.06+]

Controls which artificial keyboard and mouse events are ignored by hotkeys and hotstrings.

```
#InputLevel [Level]
```

Parameters

Level

An integer between 0 and 100. If omitted, it defaults to 0.

General Remarks

For an explanation of how `SendLevel` and `#InputLevel` are used, see [SendLevel](#).

This directive is positional: it affects all hotkeys and hotstrings between it and the next `#InputLevel` directive. If not specified by an `#InputLevel` directive, hotkeys and hotstrings default to level 0.

[v1.1.23+]: A hotkey's input level can also be set using the `Hotkey` command. For example: `Hotkey, #z, my_hotkey_sub, I1`

The input level of a hotkey or non-auto-replace hotstring is also used as the default [send level](#) for any keystrokes or button clicks generated by that hotkey or hotstring. Since a keyboard or mouse [remapping](#) is actually a pair of hotkeys, this allows `#InputLevel` to be used to allow remappings to trigger other hotkeys.

AutoHotkey versions older than [v1.1.06] behave as though `#InputLevel 0` and `SendLevel 0` are in effect.

Related

[SendLevel](#), [Hotkeys](#), [Hotstrings](#)

Examples

```
#InputLevel 1
Numpad0::LButton
#InputLevel 0
; This hotkey can be triggered by both Numpad0 and
LButton:
~LButton::MsgBox Clicked
```

#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.

As an oversimplified example, the hotkey `^c::Send ^c` would produce an infinite loop of keystrokes. To avoid this, add the [\\$ prefix](#) to the hotkey definition (e.g. `$^c::`) so that the hotkey cannot be triggered by the Send command.

If this directive is unspecified in the script, it will behave as though set to 70.

Related

[#HotkeyInterval](#)

Example

```
#MaxHotkeysPerInterval 200
```

#MaxThreads

Sets the maximum number of simultaneous [threads](#).

#MaxThreads Value

Parameters

Value

The maximum total number of [threads](#) that can exist simultaneously. Specifying a number higher than 255 is the same as specifying 255 (in versions prior to 1.0.48, the limit was 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. It would also prevent [timers](#) from running whenever another [thread](#) is sleeping or waiting.

Up to two of the following types of [threads](#) may be created even when #MaxThreads has been reached: A [hotkey](#), [hotstring](#), [OnClipboardChange](#), or [GUI event](#) if the first line of its subroutine is [ExitApp](#), [Pause](#), [Edit](#), [Reload](#), [KeyHistory](#), [ListLines](#), [ListVars](#), or [ListHotkeys](#). Also, the [OnExit](#) subroutine will always launch regardless of how many threads exist.

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

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 (such as [Thread Interrupt](#) and [Critical](#)). 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 the first hotkey label you wish to have affected by it.

Related

[#MaxThreads](#), [#MaxThreadsPerHotkey](#), [Critical](#), [Thread \(command\)](#), [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](#) or [hotstring](#).

#MaxThreadsPerHotkey Value

Parameters

Value

The maximum number of [threads](#) that can be launched for a given hotkey/hotstring subroutine (limit 255).

Remarks

This setting is used to control how many "instances" of a given [hotkey](#) or [hotstring](#) 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](#), [Critical](#), [Threads](#), [Hotkey](#),
[#MaxHotkeysPerInterval](#), [#HotkeyInterval](#), [ListHotkeys](#)

Example

```
#MaxThreadsPerHotkey 3
```

#MenuMaskKey [AHK_L 38+]

Changes which key is used to mask   events.

#MenuMaskKey KeyName

Parameters

KeyName

A [key name](#) or [VKnn](#) sequence which specifies a non-zero virtual keycode. Scan codes are used only in [v1.1.28+], which also supports [SCnnn](#) and [VKnnSCnnn](#).

Remarks

The mask key is sent automatically to prevent the Start menu or the active window's menu bar from activating at unexpected times.

The default mask key is . A directive can be used to change the mask key to a key with fewer side effects.

Good candidates are virtual key codes which generally have no effect, such as vk07, which Microsoft documents as "undefined", or vkFF, which is reserved to mean "no mapping" (a key which has no function). [v1.1.28+]: Both the VK and SC can be specified, and are not required to match an existing key. Specifying `vk00sc000` will disable all automatic masking. Some values, such as zero VK with non-zero SC, may fail to suppress the Start menu.

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.

Hotkeys: If a hotkey is implemented using the keyboard hook or mouse hook, the final keypress may be invisible to the active window and the system. If the system was to detect *only* a   down and keyup with no intervening keypress, it would usually activate a menu. To prevent this, the

keyboard or mouse hook may automatically send the mask key.

[v1.1.27+]: Pressing a hook hotkey causes the next `Alt Win` to be masked if all of the following conditions are met:

- The hotkey is suppressed (it lacks the [tilde modifier](#)).
- `Alt Win` is physically down when the hotkey is pressed.
- The modifier is physically down or the hotkey requires the modifier to activate. For example, `~#a::` in combination with `AppsKey::RWin` causes masking when `Menu A` is pressed, but `Menu Win` is able to open the Start Menu.
- `Alt Win` is masked if `Ctrl Win` is down when the hotkey was pressed, since `Ctrl Alt` does not activate the menu bar.
- `Win` is masked if the most recent `Win` was modified with `Ctrl Shift Alt`. The Start Menu does not normally activate in those cases. However, key-repeat occurs even for the `Win` if it was the last key physically pressed, so it can be hard to predict *when* the most recent `Win` was.
- Either the keyboard hook is not installed (i.e. for a mouse hotkey), or there have been no other (unsuppressed) keydown or keyup events since the last `Alt Win`. Note that key-repeat occurs even for modifier keys and even after sending other keys, but only for the last physically pressed key.

Mouse hotkeys may send the mask key immediately if the keyboard hook is not installed.

Hotkeys with the [tilde modifier](#) are not intended to block the native function of the key, so in [v1.1.27+] they do not cause masking. Hotkeys like `~#a::` still suppress the menu, since the system detects that `Win` is used in combination with another key. However, mouse hotkeys and the `Win` themselves (`~LWin::` and `~RWin::`) do not suppress the Start Menu.

The Start Menu (or the active window's menu bar) can be suppressed by sending any keystroke. The following example disables the ability for the left `Win`

key to activate the Start Menu, while still allowing its use as a modifier:

```
~LWin::Send {Blind}{vk07}
```

Send: [Send](#), [ControlSend](#) and related often release modifier keys as part of their normal operation. For example, the hotkey `<#a::SendRaw %Address%` usually must release the left `Win` key prior to sending the contents of *Address*, and press the left `Win` key down afterward (so that other `Win` combinations continue working). The mask key may be sent in such cases to prevent a `Win` `Alt` combination from activating a menu.

Related

See [this thread](#) for background information.

Examples

```
#MenuMaskKey vk07 ; vk07 is unassigned.
#UseHook
#Space::
!Space::
    KeyWait LWin
    KeyWait RWin
    KeyWait Alt
    KeyHistory
return
```

#UseHook

Forces the use of the hook to implement all or some keyboard [hotkeys](#).

```
#UseHook [On|Off]
```

Parameters

On|Off

#UseHook without one of the following words after it is equivalent to #UseHook On .

On: The [keyboard hook](#) will be used to implement all keyboard hotkeys between here and the next #USEHOOK OFF (if any).

Off: Hotkeys will be implemented using the default method (RegisterHotkey() if possible; otherwise, the keyboard hook).

Remarks

Normally, the windows API function RegisterHotkey() is used to implement a keyboard hotkey whenever possible. However, the responsiveness of hotkeys might be better under some conditions if the [keyboard hook](#) is used instead.

Turning this directive ON is equivalent to using the [\\$ prefix](#) in the definition of each affected hotkey.

As with all # directives -- which are processed only once when the script is launched -- #USEHOOK should not be positioned in the script as though it were a command (that is, it is not necessary to have it contained within a subroutine). Instead, position it immediately before the first hotkey label you wish to have affected by it.

By default, hotkeys that use the [keyboard hook](#) cannot be triggered by means of the [Send](#) command. Similarly, mouse hotkeys cannot be triggered by commands such as [Click](#) because all mouse hotkeys use the [mouse hook](#). One workaround

is to use [Gosub](#) to jump directly to the hotkey's subroutine. For example:
`Gosub #LButton`.

[v1.1.06+]: [#InputLevel](#) and [SendLevel](#) provide additional control over which hotkeys and hotstrings are triggered by the Send command.

If this directive does not appear in the script at all, it will behave as though set to OFF.

Related

[#InstallKeybdHook](#), [#InstallMouseHook](#), [ListHotkeys](#), [#InputLevel](#)

Example

```
#UseHook ; Force the use of the hook for hotkeys
after this point.
#x::MsgBox, This hotkey will be implemented with
the hook.
#y::MsgBox, And this one too.
#UseHook off
#z::MsgBox, But not this one.
```

Hotkey

Creates, modifies, enables, or disables a hotkey while the script is running.

```
Hotkey, KeyName [, Label, Options]
Hotkey, IfWinActive/Exist [, WinTitle, WinText]
Hotkey, If [, Expression]
Hotkey, If, % FunctionObject
```

Parameters

KeyName

Name of the hotkey's activation key, including any [modifier symbols](#). For example, specify `#C` for the  key.

If *KeyName* already exists as a hotkey, that hotkey will be updated with the values of the command's other parameters.

KeyName can also be the name of an existing hotkey label (i.e. a double-colon label), which will cause that hotkey to be updated with the values of the command's other parameters.

When specifying an *existing* hotkey, *KeyName* is not case sensitive. However, the names of keys must be spelled the same as in the existing hotkey (e.g. Esc is not the same as Escape for this purpose). Also, the order of [modifier symbols](#) such as `^!+#` does not matter. [GetKeyName](#) can be used to retrieve the standard spelling of a key name.

When a hotkey is first created -- either by the Hotkey command or a [double-colon label](#) in the script -- its key name and the ordering of its modifier symbols becomes the permanent name of that hotkey as reflected by [A_ThisHotkey](#). This name is shared by all [variants](#) of the hotkey, and does not change even if the Hotkey command later accesses the hotkey with a different symbol ordering.

[v1.1.15+]: If the hotkey variant already exists, its behavior is updated according to whether *KeyName* includes or excludes the [tilde \(~\) prefix](#). However, prior to [v1.1.19], the hotkey was not updated if *Label* was omitted.

[v1.1.19+]: The [use hook \(\\$\) prefix](#) can be added to existing hotkeys. This prefix affects all variants of the hotkey and cannot be removed. Prior to [v1.1.19], the prefix was ignored when modifying an existing hotkey variant.

Label

The name of the [label](#) whose contents will be executed (as a new [thread](#)) when the hotkey is pressed. Both normal labels and [hotkey/hotstring](#) labels can be used. The trailing colon(s) should not be included. If *Label* is dynamic (e.g. `%VarContainingLabelName%`), [IsLabel\(VarContainingLabelName\)](#) may be called beforehand to verify that the label exists.

[v1.1.20+]: If not a valid label name, this parameter can be the name of a function, or a single variable reference containing a [function object](#). For example, `Hotkey %FuncObj%, On` or `Hotkey % FuncObj, On`. Other expressions which return objects are currently unsupported. When the hotkey executes, the function is called without parameters. Hotkeys can also be [defined as functions](#) without the Hotkey command.

This parameter can be left blank if *KeyName* already exists as a hotkey, in which case its label will not be changed. This is useful to change only the hotkey's *Options*.

Note: If the label or function is specified but the hotkey is disabled from a previous use of this command, the hotkey will remain disabled. To prevent this, include the word ON in *Options*.

This parameter can also be one of the following special values:

On: The hotkey becomes enabled. No action is taken if the hotkey is already On.

Off: The hotkey becomes disabled. No action is taken if the hotkey is already Off.

Toggle: The hotkey is set to the opposite state (enabled or disabled).

AltTab (and others): These are special Alt-Tab hotkey actions that are described [here](#).

Caution: Defining a label named On, Off, Toggle or AltTab (or any variation recognized by this command) may cause inconsistent behavior. It is strongly recommended that these values not be used as label names.

Options

A string of zero or more of the following letters with optional spaces in between. For example: `UseErrorLevel B0`.

UseErrorLevel: If the command encounters a problem, this option skips the warning dialog, sets [ErrorLevel](#) to one of the codes from the table [below](#), then allows the [current thread](#) to continue.

On: Enables the hotkey if it is currently disabled.

Off: Disables the hotkey if it is currently enabled. This is typically used to create a hotkey in an initially-disabled state.

B or **B0:** Specify the letter B to buffer the hotkey as described in [#MaxThreadsBuffer](#). Specify B0 (B with the number 0) to disable this type of buffering.

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.

Tn: Specify the letter T followed by a the number of threads to allow for this hotkey as described in [#MaxThreadsPerHotkey](#). For example: `T5`.

In (InputLevel) [v1.1.23+]: Specify the letter I (or i) followed by the hotkey's [input level](#). For example: `I1`.

If any of the 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 is about to be created by this command -- the options will default to those most recently in effect. For example, the instance of [#MaxThreadsBuffer](#) that occurs closest to the bottom of the script will be used. If [#MaxThreadsBuffer](#) does not appear in the script, its default setting (OFF in this case) will be used. This behavior also applies to [#IfWin](#): the bottommost occurrence applies to newly created hotkeys unless "[Hotkey IfWin](#)" has executed since the script started.

IfWinActive

IfWinExist

IfWinNotActive

IfWinNotExist

If, Expression

If, % FunctionObject

These sub-commands make all subsequently-created hotkeys context sensitive. See [below](#) for details.

WinTitle

WinText

Within these parameters, any variable reference such as `%var%` becomes permanent the moment the command finishes. In other words, subsequent changes to the contents of the variable are not seen by existing `IfWin` hotkeys.

Like [#IfWinActive/Exist](#), `WinTitle` and `WinText` use the default settings for [SetTitleMatchMode](#) and [DetectHiddenWindows](#) as set in the [auto-execute](#) section. See [#IfWinActive/Exist](#) for details.

If, Expression

[AHK_L 4+]: Associates subsequently-created hotkeys with a given `#If` expression. *Expression* must be an expression which has been used with the [#If directive](#) elsewhere in the script. Although this command is unable to create new expressions, it can create new hotkeys using an existing expression. See [#If example 4](#).

Note: The Hotkey command uses the string that you pass to it, not

the original source code. Commas and deref chars (percent signs) are interpreted *before* the command is called, so may need to be escaped if they are part of the original expression. [Escape sequences](#) are resolved when the script loads, so only the resulting characters are considered; for example, `Hotkey, If, x = "`t"` and `Hotkey, If, % "x = "" A_Tab ""` both correspond to `#If x = "`t"`.

Known limitation: If *Expression* contains an `and / or` operator, it is not recognized as an existing expression. As a workaround, use the equivalent `&& / ||` operator in both the original `#If` expression and the one passed to the `Hotkey` command.

If, % `FunctionObject`

[v1.1.25+]: Associates subsequently-created hotkeys with a given [function object](#). Such hotkeys will only execute if calling the given function object yields a non-zero number. This is like `Hotkey, If, Expression`, except that each hotkey can have many [variants](#) (one per object). *FunctionObject* must be a single variable (not an expression) containing an object with a *call* method. The function or *call* method can accept one parameter, the [name](#) of the hotkey.

Once passed to the `Hotkey` command, the object will never be deleted (but memory will be reclaimed by the OS when the process exits).

The ["three-key combination" example](#) below uses this sub-command.

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[ErrorLevel](#) is changed only when: 1) the first parameter is `IfWin[Not]Active/Exist`, in which case it is set to 1 if there was a problem or 0 otherwise; or 2) the word `UseErrorLevel` is present in the *Options* parameter.

[v1.1.25+]: If the first parameter is "If", an exception is thrown if the second parameter is invalid or a memory allocation fails. `ErrorLevel` is not set in those

cases, but is still set to 0 on success.

Error	Description
1	The <i>Label</i> parameter specifies a nonexistent label name.
2	The <i>KeyName</i> parameter specifies one or more keys that are either not recognized or not supported by the current keyboard layout/language.
3	Unsupported prefix key. For example, using the mouse wheel as a prefix in a hotkey such as <code>WheelDown & Enter</code> is not supported.
4	The <i>KeyName</i> parameter is not suitable for use with the AltTab or ShiftAltTab actions. A combination of two keys is required. For example: <code>RControl & RShift::AltTab</code> .
5	The command attempted to modify a nonexistent hotkey.
6	The command attempted to modify a nonexistent variant of an existing hotkey. To solve this, use Hotkey IfWin to set the criteria to match those of the hotkey to be modified.
98	Creating this hotkey would exceed the 1000-hotkey-per-script limit (however, each hotkey can have an unlimited number of variants , and there is no limit to the number of hotstrings).
99	Out of memory. This is very rare and usually happens only when the operating system has become unstable.

Tip: The `UseErrorLevel` option can be used to test for the existence of a hotkey variant. For example:

```
Hotkey, ^!p,, UseErrorLevel
if ErrorLevel in 5,6
    MsgBox The hotkey does not exist or it has
    no variant for the current IfWin criteria.
```

Remarks

The [current IfWin setting](#) determines the [variant](#) of a hotkey upon which the Hotkey command will operate.

If the goal is to disable selected hotkeys or hotstrings automatically based on the type of window that is active, `Hotkey, ^!c, Off` is usually less convenient than using [#IfWinActive/Exist](#) (or their dynamic counterparts "Hotkey IfWinActive/Exist" [below](#)).

Creating hotkeys via [double-colon labels](#) performs better than using the Hotkey command because the hotkeys can all be enabled as a batch when the script starts (rather than one by one). 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 checking the built-in variable [A_ThisHotkey](#).

If the script is [suspended](#), newly added/enabled hotkeys will also be suspended until the suspension is turned off (unless they are 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.

Although the Hotkey command cannot directly enable or disable hotkeys in scripts other than its own, in most cases it can [override](#) them by creating or enabling the same hotkeys. Whether this works depends on a combination of factors: 1) Whether the hotkey to be overridden is a [hook hotkey](#) in the other script (non-hook hotkeys can always be overridden); 2) The fact that the most recently started script's hotkeys generally take precedence over those in other scripts (therefore, if the script intending to override was started most recently, its override should always succeed); 3) Whether the enabling or creating of this hotkey will newly activate the [keyboard](#) or [mouse](#) hook (if so, the override will always succeed).

Once a script has at least one hotkey, it becomes persistent, meaning that `ExitApp` rather than `Exit` should be used to terminate it. Hotkey scripts are also automatically `#SingleInstance` unless `#SingleInstance Off` has been specified.

Remarks About *Hotkey, If*

The "Hotkey If" commands allow context-sensitive `hotkeys` to be created and modified while the script is running (by contrast, the `#If` and `#IfWinActive/Exist` directives are positional and take effect before the script begins executing). For example:

```
Hotkey, IfWinActive, ahk_class Notepad
Hotkey, ^!e, MyLabel ; Creates a hotkey that
works only in Notepad.
```

Using "Hotkey If" puts context sensitivity into effect for all subsequently created or modified `hotkeys`. In addition, each If sub-command is mutually exclusive; that is, only the most recent one will be in effect.

To turn off context sensitivity (that is, to make subsequently-created hotkeys work in all windows), specify any If sub-command but omit the parameters. For example: `Hotkey, If` or `Hotkey, IfWinActive`.

If "Hotkey If" is never used by a script, the bottommost use of any `#If` or `#IfWin` directive (if any) will be in effect for the Hotkey command.

When a mouse or keyboard hotkey is disabled via an If sub-command or directive, it performs its native function; that is, it passes through to the active window as though there is no such hotkey. However, joystick hotkeys always pass through, whether they are disabled or not.

Variant (Duplicate) Hotkeys

A particular hotkey can be created more than once if each definition has different IfWin criteria. These are known as *hotkey variants*. For example:

```
Hotkey, IfWinActive, ahk_class Notepad
Hotkey, ^!c, MyLabelForNotepad
Hotkey, IfWinActive, ahk_class WordPadClass
Hotkey, ^!c, MyLabelForWordPad
Hotkey, IfWinActive
Hotkey, ^!c, MyLabelForAllOtherWindows
```

If more than one variant of a hotkey is eligible to fire, only the one created earliest will fire. The exception to this is the global variant (the one with no IfWin criteria): It always has the lowest precedence, and thus will fire only if no other variant is eligible.

When creating duplicate hotkeys, the order of [modifier symbols](#) such as ^!+# does not matter. For example, `^!c` is the same as `!^c`. However, keys must be spelled consistently. For example, *Esc* is not the same as *Escape* for this purpose (though the case does not matter). Finally, any hotkey with a [wildcard prefix \(*\)](#) is entirely separate from a non-wildcard one; for example, `*F1` and `F1` would each have their own set of variants.

For more information about IfWin hotkeys, see [#IfWin's General Remarks](#).

Related

[Hotkey Symbols](#), [#IfWinActive/Exist](#), [#MaxThreadsBuffer](#), [#MaxThreadsPerHotkey](#), [Suspend](#), [IsLabel\(\)](#), [Threads](#), [Thread](#), [Critical](#), [Gosub](#), [Return](#), [Menu](#), [SetTimer](#)

Examples

```
Hotkey, ^!z, MyLabel
return

MyLabel:
MsgBox You pressed %A_ThisHotkey%.
return
```

```
; Other examples:  
Hotkey, RCtrl & RShift, AltTab ; Makes RCtrl &  
RShift operate like Alt-Tab.  
Hotkey, #c, On ; Re-enables the Win-C hotkey.  
Hotkey, $+#c, Off ; Disables the Shift-Win-C  
hotkey.  
Hotkey, ^!a, , T5 ; Changes the hotkey to allow 5  
threads.
```

```
Hotkey, IfWinActive, ahk_class Notepad  
Hotkey, ^!c, MyLabelForNotepad ; Creates Ctrl-  
Alt-C as a hotkey that works only in Notepad.
```

```
; This GUI allows you to register primitive three-  
key combination hotkeys:
```

```
Gui Add, Text, xm, Prefix key:  
Gui Add, Edit, yp x100 w100 vPrefix, Space  
Gui Add, Text, xm, Suffix hotkey:  
Gui Add, Edit, yp x100 w100 vSuffix, f & j  
Gui Add, Button, Default, Register  
Gui Show  
return  
  
ButtonRegister() {  
    global  
    Gui Submit, NoHide  
    local fn  
    fn := Func("HotkeyShouldFire").Bind(Prefix)  
    Hotkey If, % fn  
    Hotkey % Suffix, FireHotkey  
}
```

```
HotkeyShouldFire(prefix, thisHotkey) {  
    return GetKeyState(prefix)  
}
```

```
FireHotkey() {  
    MsgBox %A_ThisHotkey%  
}
```

```
GuiClose:  
GuiEscape:  
ExitApp
```

Hotstring [v1.1.28+]

Creates, modifies, enables, or disables a [hotstring](#) while the script is running.

```
Hotstring(String [, Replacement, OnOffToggle])
Hotstring(NewOptions)
OldValue := Hotstring("EndChars" [, NewValue])
OldValue := Hotstring("MouseReset" [, NewValue])
Hotstring("Reset")
```

Parameters

String

The hotstring's trigger string, preceded by [the usual colons](#) and [option characters](#). For example, `::btw` or `::*:]d`.

String may be matched to an existing hotstring by considering [case-sensitivity \(C\)](#), [word-sensitivity \(?\)](#), activation criteria (as set by [#If](#), [#IfWin](#) or [Hotkey, If](#)) and the trigger string. For example, `::btw` and `::BTW` match unless the case-sensitive mode was enabled as a default, while `::C:btw` and `::C:BTW` never match. The `C` and `?` options may be included in *String* or set as defaults by the [#Hotstring](#) directive or a previous call to [this function](#).

If the hotstring already exists, any options specified in *String* are put into effect, while all other options are left as is. However, since hotstrings with `C` or `?` are considered distinct from other hotstrings, it is not possible to add or remove these options. Instead, turn off the existing hotstring and create a new one.

When a hotstring is first created -- either by the `Hotstring` function or a [double-colon label](#) in the script -- its trigger string and sequence of option characters becomes the permanent name of that hotstring as reflected by [A_ThisHotkey](#). This name does not change even if the `Hotstring` function

later accesses the hotstring with different option characters.

If the **X (execute) option** is present in *String* (not just set as a default), the *Replacement* parameter is interpreted as a label or function name instead of replacement text. This has no effect if *Replacement* is an object.

Replacement

The replacement string, or a **label**, **function** or **function object** to call (as a new **thread**) when the hotstring triggers.

By default, all strings are treated as replacement text. To specify a label or function by name, include the **X (execute) option** in *String*. Both normal labels and **hotkey/hotstring** labels can be used, and the trailing colon(s) should not be included. If a function and a label exist with the same name, the label takes precedence. To use the function instead, pass a **function reference**.

This parameter can be left blank if the hotstring already exists, in which case its replacement will not be changed. This is useful to change only the hotstring's options, or to turn it on or off.

Note: If this parameter is specified but the hotstring is disabled from a previous use of this function, the hotstring will remain disabled. To prevent this, include the word **"On"** in *OnOffToggle*.

OnOffToggle

One of the following strings (enclosed in quote marks if used literally):

On: Enables the hotstring.

Off: Disables the hotstring.

Toggle: Sets the hotstring to the opposite state (enabled or disabled).

Errors

This function throws an exception if the parameters are invalid or a memory allocation fails. It does not affect **ErrorLevel**.

An exception is also thrown if *Replacement* is omitted and *String* is valid but does not match an existing hotstring. This can be utilized to test for the existence of a hotstring. For example:

```
try
    Hotstring("::btw")
catch
    MsgBox The hotstring does not exist or it
    has no variant for the current IfWin criteria.
```

Remarks

The [current IfWin setting](#) determines the [variant](#) of a hotstring upon which the Hotstring function will operate.

A given label or function can be the target of more than one hotstring. If it is known that a label or function was called by a hotstring, you can determine which hotstring by checking the built-in variable [A_ThisHotkey](#).

If the script is [suspended](#), newly added/enabled hotstrings will also be suspended until the suspension is turned off (unless they are 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 function.

This function cannot directly enable or disable hotstrings in scripts other than its own.

Once a script has at least one hotstring, it becomes persistent, meaning that [ExitApp](#) rather than Exit should be used to terminate it. Hotstring scripts are also automatically [#SingleInstance](#) unless `#SingleInstance Off` has been specified.

Variant (Duplicate) Hotstrings

A particular hotstring can be created more than once if each definition has different IfWin criteria, [case-sensitivity](#) (`C` vs. `C0 / C1`), or [word-sensitivity](#)

(?). These are known as *hotstring variants*. For example:

```
Hotkey, IfWinActive, ahk_group CarForums
Hotstring("::btw", "behind the wheel")
Hotkey, IfWinActive, Inter-Office Chat
Hotstring("::btw", "back to work")
Hotkey, IfWinActive
Hotstring("::btw", "by the way")
```

If more than one variant of a hotstring is eligible to fire, only the one created earliest will fire.

For more information about IfWin, see [#IfWin's General Remarks](#).

EndChars

```
OldValue := Hotstring("EndChars" [, NewValue])
```

Retrieves or modifies the set of characters used as [ending characters](#) by the hotstring recognizer. For example:

```
prev_chars := Hotstring("EndChars", "-()[  
{ } ' : ; \" / \ , . ? ! ` `n `t")  
MsgBox The previous value was: %prev_chars%
```

[#Hotstring EndChars](#) also affects this setting.

It is currently not possible to specify a different set of end characters for each hotstring.

MouseReset

```
OldValue := Hotstring("MouseReset" [, NewValue])
```

Retrieves or modifies the global setting which controls whether mouse clicks reset the hotstring recognizer, as described [here](#). *NewValue* should be 1 (true) to enable mouse click detection and resetting of the hotstring recognizer, or 0 (false) to disable it. The return value is the setting which was in effect before the function was called.

The [mouse](#) hook may be installed or removed if justified by the changes made by this function.

[#Hotstring NoMouse](#) also affects this setting, and is equivalent to specifying `false` for *NewValue*.

Reset [v1.1.28.01+]

```
Hotstring("Reset")
```

Immediately resets the hotstring recognizer. In other words, the script will begin waiting for an entirely new hotstring, eliminating from consideration anything you previously typed.

Setting Default Options

```
Hotstring(NewOptions)
```

To set new default options for subsequently created hotstrings, pass the options to the Hotstring function without any leading or trailing colon. For example: `Hotstring("T")`.

Turning on [case-sensitivity \(C\)](#) or [word-sensitivity \(?\)](#) also affects which existing hotstrings will be found by any subsequent calls to the Hotstring function. For example, `Hotstring(":T:btw")` will find `::BTW` by default, but not if `Hotstring("C")` or [#Hotstring C](#) is in effect. This can be undone or overridden by passing a mutually-exclusive option; for example, `C0` and `C1` override `C`.

Related

[Hotstrings](#), [#IfWinActive/Exist](#), [#MaxThreadsPerHotkey](#), [Suspend](#), [Threads](#), [Thread](#), [Critical](#)

Examples

Example #1: Hotstring Helper

The following script might be useful if you are a heavy user of hotstrings. It's based on [the script created by Andreas Borutta](#). By pressing  (or another hotkey of your choice), the currently selected text can be turned into a hotstring. For example, if you have "by the way" selected in a word processor, pressing  will prompt you for its abbreviation (e.g. btw) and then add the new hotstring to the script. The hotstring will be activated without reloading the script.

```
#h:: ; Win+H hotkey
; Get the text currently selected. The clipboard
is used instead of
; "ControlGet Selected" because it works in a
greater variety of editors
; (namely word processors). Save the current
clipboard contents to be
; restored later. Although this handles only
plain text, it seems better
; than nothing:
ClipboardOld := Clipboard
Clipboard := "" ; Must start off blank for
detection to work.
Send ^c
ClipWait 1
if ErrorLevel ; ClipWait timed out.
    return
; Replace CRLF and/or LF with `n for use in a
"send-raw" hotstring:
```

```
; The same is done for any other characters that
might otherwise
```

```
; be a problem in raw mode:
```

```
ClipContent := StrReplace(Clipboard, "`",
"``") ; Do this replacement first to avoid
interfering with the others below.
```

```
ClipContent := StrReplace(ClipContent, "`r`n",
"``r") ; Using `r works better than `n in MS
Word, etc.
```

```
ClipContent := StrReplace(ClipContent, "`n",
"``r")
```

```
ClipContent := StrReplace(ClipContent, "`t",
"``t")
```

```
ClipContent := StrReplace(ClipContent, "`;",
"``;")
```

```
Clipboard := ClipboardOld ; Restore previous
contents of clipboard.
```

```
ShowInputBox(":T:`::" ClipContent)
```

```
return
```

```
ShowInputBox(DefaultValue)
```

```
{
```

```
    ; This will move the InputBox's caret to a
more friendly position:
```

```
    SetTimer, MoveCaret, 10
```

```
    ; Show the InputBox, providing the default
hotstring:
```

```
    InputBox, UserInput, New Hotstring,
    (
```

```
    Type your abbreviation at the indicated
insertion point. You can also edit the
replacement text if you wish.
```

```
    Example entry: :R:btw`::by the way
```

```
    ),,,,,,, %DefaultValue%
```

```
    if ErrorLevel ; The user pressed Cancel.
```

```
    return
```

```

        if RegExMatch(UserInput, "0)(?P<Label>:.*?:
(?P<Abbreviation>.*)?::(?P<Replacement>.*)",
Hotstring)
    {
        if !Hotstring.Abbreviation
            MsgText := "You didn't provide an
abbreviation"
        else if !Hotstring.Replacement
            MsgText := "You didn't provide a
replacement"
        else
        {
            Hotstring(Hotstring.Label,
Hotstring.Replacement) ; Enable the hotstring
now.
            FileAppend, `n%UserInput%,
%A_ScriptFullPath% ; Save the hotstring for
later use.
        }
    }
    else
        MsgText := "The hotstring appears to be
improperly formatted"

    if MsgText
    {
        MsgBox, 4,, %MsgText%. Would you like to
try again?
        IfMsgBox, Yes
            ShowInputBox(DefaultValue)
    }
    return

MoveCaret:
WinWait, New Hotstring
; Otherwise, move the InputBox's insertion

```

point to where the user will type the abbreviation.

```
    Send {Home}{Right 3}  
    SetTimer,, Off  
    return  
}
```

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

This command is equivalent to selecting the View->Hotkeys menu item in the main window.

If a hotkey has been disabled via the [Hotkey](#) command, it will be listed as OFF or PART ("PART" means that only some of the hotkey's [variants](#) are disabled).

[v1.1.16+]: If any of a hotkey's variants have a non-zero [#InputLevel](#), the level (or minimum and maximum levels) are displayed.

If any of a hotkey's subroutines are currently running, the total number of threads is displayed for that hotkey.

Finally, the type of hotkey is also displayed, which is one of the following:

reg: The hotkey is implemented via the operating system's RegisterHotkey() function.

reg(no): Same as above except that this hotkey is inactive (due to being unsupported, disabled, or [suspended](#)).

k-hook: The hotkey is implemented via the [keyboard hook](#).

m-hook: The hotkey is implemented via the [mouse hook](#).

2-hooks: The hotkey requires both the hooks mentioned above.

joypoll: The hotkey is implemented by polling the joystick at regular intervals.

Related

[#InstallKeybdHook](#), [#InstallMouseHook](#), [#UseHook](#), [KeyHistory](#), [ListLines](#), [ListVars](#), [#MaxThreadsPerHotkey](#), [#MaxHotkeysPerInterval](#)

Example

ListHotkeys

#InstallKeybdHook

Forces the unconditional installation of the keyboard hook.

```
#InstallKeybdHook
```

Remarks

The keyboard hook monitors keystrokes for the purpose of activating [hotstrings](#) and any keyboard [hotkeys](#) not supported by RegisterHotkey (which is a function built into the operating system). It also supports a few other features such as the [Input](#) command.

AutoHotkey does not install the keyboard and mouse hooks unconditionally because together they consume at least 500 KB of memory. Therefore, the keyboard hook is normally installed only when the script contains one of the following: 1) [hotstrings](#); 2) one or more [hotkeys](#) that require the keyboard hook (most do not); 3) [SetCaps/Scroll/NumLock AlwaysOn/AlwaysOff](#); 4) the [Input](#) command, for which the hook is installed upon first actual use.

By contrast, the `#InstallKeybdHook` directive will unconditionally install the keyboard hook, which might be useful to allow [KeyHistory](#) to display the last 20 keystrokes (for debugging purposes), or to avoid the need for [#HotkeyModifierTimeout](#).

You can determine whether a script is using the hook via the [KeyHistory](#) command or menu item. You can determine which hotkeys are using the hook via the [ListHotkeys](#) command or menu item.

This directive also makes a script [persistent](#), meaning that [ExitApp](#) should be used to terminate it.

Related

[#InstallMouseHook](#), [#UseHook](#), [Hotkey](#), [Input](#), [#Persistent](#), [KeyHistory](#), [Hotstrings](#), [GetKeyState](#), [KeyWait](#)

Example

```
#InstallKeybdHook
```

#InstallMouseHook

Forces the unconditional installation of the mouse hook.

```
#InstallMouseHook
```

Remarks

The mouse hook monitors mouse clicks for the purpose of activating mouse [hotkeys](#) and [facilitating hotstrings](#).

AutoHotkey does not install the keyboard and mouse hooks unconditionally because together they consume at least 500 KB of memory (but if the keyboard hook is installed, installing the mouse hook only requires about 50 KB of additional memory; and vice versa). Therefore, the mouse hook is normally installed only when the script contains one or more mouse [hotkeys](#). It is also installed for [hotstrings](#), but that can be disabled via [#Hotstring NoMouse](#).

By contrast, the `#InstallMouseHook` directive will unconditionally install the mouse hook, which might be useful to allow [KeyHistory](#) to monitor mouse clicks.

You can determine whether a script is using the hook via the [KeyHistory](#) command or menu item. You can determine which hotkeys are using the hook via the [ListHotkeys](#) command or menu item.

This directive also makes a script [persistent](#), meaning that [ExitApp](#) should be used to terminate it.

Related

[#InstallKeybdHook](#), [#UseHook](#), [Hotkey](#), [#Persistent](#), [KeyHistory](#), [GetKeyState](#), [KeyWait](#)

Example

#InstallMouseHook

#KeyHistory

Sets the maximum number of keyboard and mouse events displayed by the [KeyHistory](#) window. You can set it to 0 to disable key history.

```
#KeyHistory MaxEvents
```

Parameters

MaxEvents

The maximum number of keyboard and mouse events displayed by the [KeyHistory](#) window (default 40, limit 500). Specify 0 to disable key history entirely.

Remarks

Because this setting is put into effect before the script begins running, it is only necessary to specify it once (anywhere in the script).

Because each keystroke or mouse click consists of a down-event and an up-event, [KeyHistory](#) displays only half as many "complete events" as specified here. For example, if the script contains `#KeyHistory 50`, up to 25 keystrokes and mouse clicks will be displayed.

Related

[KeyHistory](#), [#NoTrayIcon](#)

Example

```
#KeyHistory 0      ; Disable key history.  
#KeyHistory 100   ; Store up to 100 events.
```

BlockInput

Disables or enables the user's ability to interact with the computer via keyboard and mouse.

BlockInput, Mode

Parameters

Mode

Mode 1: 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.

Mode 2: This mode operates independently of the other two. For example, `BlockInput On` will continue to block input until `BlockInput Off` is used, even if one of the below is also in effect.

Send: The user's keyboard and mouse input is ignored while a `Send` or `SendRaw` is in progress (the traditional `SendEvent mode` only). This prevents the user's keystrokes from disrupting the flow of simulated keystrokes. When the `Send` finishes, input is re-enabled (unless still blocked by a previous use of `BlockInput On`).

Mouse: The user's keyboard and mouse input is ignored while a `Click`, `MouseMove`, `MouseClick`, or `MouseClickDrag` is in progress (the traditional `SendEvent mode` only). This prevents the user's mouse movements and clicks from disrupting the simulated mouse events. When the mouse command finishes, input is re-enabled (unless still blocked by a previous use of `BlockInput On`).

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.

Mode 3 [requires v1.0.43.11+]: This mode operates independently of the other two. For example, if `BlockInput On` and `BlockInput MouseMove` are both in effect, mouse movement will be blocked until both are turned off.

MouseMove: The mouse cursor will not move in response to the user's physical movement of the mouse (DirectInput applications are a possible exception). When a script first uses this command, the [mouse hook](#) is installed (if it is not already). In addition, the script becomes [persistent](#), meaning that `ExitApp` should be used to terminate it. The mouse hook will stay installed until the next use of the `Suspend` or `Hotkey` command, at which time it is removed if not required by any hotkeys or hotstrings (see [#Hotstring NoMouse](#)).

MouseMoveOff: Allows the user to move the mouse cursor.

Remarks

Note: Mode 1 and 2 might have no effect if UAC is enabled or the script has not been run as administrator. For more information, refer to the [FAQ](#).

In preference to `BlockInput`, it is often better to use `SendMode Input` or `SendMode Play` so that keystrokes and mouse clicks become uninterruptible. This is because unlike `BlockInput`, those modes do not discard what the user types during the send; instead, those keystrokes are buffered and sent afterward. Avoiding `BlockInput` also avoids the need to work around sticking keys as described in the next paragraph.

If `BlockInput` becomes active while the user is holding down keys, it might cause those keys to become "stuck down". This can be avoided by waiting for the keys to be released prior to turning `BlockInput` on, as in this example:

```
^!p::
```

```
KeyWait Control ; Wait for the key to be
released. Use one KeyWait for each of the
hotkey's modifiers.
KeyWait Alt
BlockInput On
; ... send keystrokes and mouse clicks ...
BlockInput Off
return
```

Input blocking is automatically and momentarily disabled whenever an  event is sent (then re-enabled afterward).

When BlockInput is in effect, user input is blocked but AutoHotkey can simulate keystrokes and mouse clicks. However, pressing    re-enable input due to a Windows API feature.

Certain types of [hook hotkeys](#) can still be triggered when BlockInput is on. Examples include `MButton` (mouse hook) and `LWin & Space` (keyboard hook with explicit prefix rather than modifiers `$#`).

Input is automatically re-enabled when the script closes.

Related

[SendMode](#), [Send](#), [Click](#), [MouseMove](#), [MouseClick](#), [MouseClickDrag](#)

Example

```
BlockInput, On
Run, notepad
WinWaitActive, Untitled - Notepad
Send, {F5} ; pastes time and date
BlockInput, Off
```

Click [v1.0.43+]

Clicks a mouse button at the specified coordinates. It can also hold down a mouse button, turn the mouse wheel, or move the mouse.

Here are examples of common usages (all commas are optional):

Click (by itself)	Clicks the left mouse button once at the mouse cursor's current position.
Click, 44, 55	Clicks the left mouse button once at coordinates 44, 55 (based on CoordMode).
Click, right, 44, 55	Same as above but clicks the right mouse button.
Click, 2	Clicks the left mouse button twice at the cursor's current position (i.e. double-click).
Click, down	Presses the left mouse button down and holds it.
Click, up, right	Releases the right mouse button.
Click, %x%, %y%	Since click does not support expressions , variables should be enclosed in percent signs.

Zero or more of the following items can follow the word *Click*. Separate each item from the next with at least one space, tab, and/or comma. The items can appear in any order except *ClickCount*, which must occur somewhere to the right of the coordinates (if coordinates are present).

X, Y: The x/y coordinates to which the mouse cursor is moved prior to clicking. Coordinates are relative to the active window unless [CoordMode](#) was used to change that. If omitted, the cursor's current position is used.

Button Name: Left (default), Right, Middle (or just the first letter of each of these); or the fourth or fifth mouse button (X1 or X2).

Note: Unlike [MouseClicked](#), the left and right buttons behave consistently

across all systems, even if the user has swapped the buttons via the system's control panel.

Mouse Wheel: Specify `WheelUp` or `WU` to turn the wheel upward (away from you); specify `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). For *ClickCount* (below), specify the number of notches to turn the wheel. However, some applications do not obey a *ClickCount* higher than 1 for the mouse wheel. For them, use a [Loop](#) such as the following:

```
Loop 5
    Click, WheelUp
```

ClickCount: The number of times to click the mouse (examples: `Click, 2`, `Click, 100, 200, 2`). If omitted, the button is clicked once. If coordinates are specified, *ClickCount* must appear after them. Specify zero (0) to move the mouse without clicking (for example: `Click, 100, 200, 0`).

Down or Up: These words are normally omitted, in which case each click consists of a down-event followed by an up-event. Otherwise, specify *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.

Relative: The word *Rel* or *Relative* treats the specified X and Y 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

Click is generally preferred over [MouseClicked](#) because it automatically compensates if the user has swapped the left and right mouse buttons via the system's control panel.

Click uses the sending method set by [SendMode](#). To override this mode for a particular click, use a specific Send command 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](#), *Click* does not automatically release the modifier keys (

example, if [Control](#) 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](#), [MouseClicked](#), [MouseClickedDrag](#), [MouseMove](#), [ControlClick](#), [BlockInput](#)

Examples

Click ; Click left mouse button at mouse cursor's current position.

Click, 100, 200 ; Click left mouse button at specified coordinates.

Click, 100, 200, 0 ; Move the mouse without clicking.

Click, 100, 200, right ; Click the right mouse button.

Click, 2 ; Perform a double-click.

Click, down ; Presses down the left mouse button and holds it.

Click, up, right ; Releases the right mouse button.

ControlClick

Sends a mouse button or mouse wheel event to a control.

```
ControlClick [, Control-or-Pos, WinTitle, WinText,  
WhichButton, ClickCount, Options, ExcludeTitle,  
ExcludeText]
```

Parameters

Control-or-Pos

If this parameter is blank, the target window's topmost control will be clicked (or the target window itself if it has no controls). Otherwise, one of the two modes below will be used.

Mode 1 (Position): Specify the 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. For 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 mode 1, the X and Y option letters of the *Options* parameter are ignored.

Mode 2 (ClassNN or Text): Specify 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](#).

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`.

To operate upon a control's HWND (window handle), leave this parameter blank and specify `ahk_id %ControlHwnd%` for the *WinTitle* parameter (this also works on hidden controls even when [DetectHiddenWindows](#) is Off). The HWND of a control is typically retrieved via [ControlGet Hwnd](#), [MouseGetPos](#), or [DllCall](#).

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). Hidden text elements are detected if [DetectHiddenText](#) is ON.

WhichButton

The button to click: LEFT, RIGHT, MIDDLE (or just the first letter of each of these). If omitted or blank, the LEFT button will be used.

X1 (XButton1, the 4th mouse button) and X2 (XButton2, the 5th mouse button) are also supported.

WheelUp (or WU) and WheelDown (or WD) are also supported. In this case, *ClickCount* is the number of notches to turn the wheel.

Windows Vista or later [v1.0.48+]: WheelLeft (or WL) and WheelRight (or WR) are also supported (they have no effect on older operating systems). In this case, *ClickCount* is the number of notches to turn the wheel.

ClickCount

The number of clicks to send, which can be an [expression](#). If omitted or blank, 1 click is sent.

Options

A series of zero or more of the following option letters. For example: `d x50 y25`.

NA [v1.0.45+]: May improve reliability. See [reliability](#) below.

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.

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).

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.

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.

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.

Use decimal (not hexadecimal) numbers for the **X** and **Y** options.

ExcludeTitle

Windows whose titles include this value will not be considered.

ExcludeText

Windows whose text include this value will not be considered.

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

Reliability

To improve reliability -- especially during times when the user is physically moving the mouse during the `ControlClick` -- one or both of the following may help:

- 1) Use `SetControlDelay -1` prior to `ControlClick`. This avoids holding the mouse button down during the click, which in turn reduces interference from the user's physical movement of the mouse.
- 2) Specify the string `NA` anywhere in the sixth parameter (*Options*) as shown below:

```
SetControlDelay -1  
ControlClick, Toolbar321, WinTitle,,,, NA
```

`NA` avoids marking the target window as active and avoids merging its input processing with that of the script, which may prevent physical movement of the mouse from interfering (but usually only when the target window is not active). However, this method might not work for all types of windows and controls.

Remarks

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
```

Window titles and text are case sensitive. Hidden windows are not detected unless `DetectHiddenWindows` has been turned on.

Related

SetControlDelay, Control, ControlGet, ControlGetText, ControlMove, ControlGetPos, ControlFocus, ControlSetText, ControlSend, Click

Examples

```
ControlClick, OK, Some Window Title ; Clicks the
OK button
ControlClick, x55 y77, WinTitle ; Clicks at a set
of coordinates. Note the lack of a comma between X
and Y.

; The following method may improve reliability and
reduce side effects:
SetControlDelay -1
ControlClick, Toolbar321, WinTitle,,,, NA x192 y10
; Clicks in NA mode at coordinates that are
relative to a named control.
```

ControlSend[Raw]

Sends simulated keystrokes to a window or control.

```
ControlSend [, Control, Keys, WinTitle, WinText,  
ExcludeTitle, ExcludeText]
```

ControlSendRaw: Same parameters as above.

Parameters

Control

Can be either ClassNN (the classname and instance number of the control) or the control's text, both of which can be determined via Window Spy. When using text, the matching behavior is determined by [SetTitleMatchMode](#). If this parameter is blank or omitted, the target window's topmost control will be used. If this parameter is `ahk_parent`, the keystrokes will be sent directly to the target window instead of one of its controls (see [Automating Winamp](#) for an example).

To operate upon a control's HWND (window handle), leave the *Control* parameter blank and specify `ahk_id %ControlHWND%` for the *WinTitle* parameter (this also works on hidden controls even when [DetectHiddenWindows](#) is Off). The HWND of a control is typically retrieved via [ControlGet Hwnd](#), [MouseGetPos](#), or [DllCall](#).

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. Use [ControlClick](#) for that.

WinTitle

A window title or other criteria identifying the target window. See

| [WinTitle](#).

WinText

| If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

Remarks

ControlSendRaw sends the keystrokes in the *Keys* parameter without translating {Enter} to   || Control   for details, see [Raw mode](#). It is also valid to use {Raw} or {Text} with ControlSend. [v1.1.27+]: [Text mode](#) may be more reliable for sending text.

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 (   ) 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.

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` [redacted] being sent (since `Alt` [redacted]es the active window's menu bar). 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 method above also allows the sending of modifier keystrokes (`Control` [redacted] / `Alt` `Shift` `Win` [redacted]) the workstation is locked (protected by logon prompt).

[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` [redacted]g properly.

The value of [SetKeyDelay](#) determines the speed at which keys are sent. If the target window does not receive the keystrokes reliably, try increasing the press duration via the second parameter of [SetKeyDelay](#) as in these examples:

```
SetKeyDelay, 10, 10  
SetKeyDelay, 0, 10  
SetKeyDelay, -1, 0
```

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
```

ControlSend is generally not capable of manipulating a window's menu bar. To work around this, use [WinMenuSelectItem](#). If that is not possible due to the nature of the menu bar, you could try to discover the message that corresponds to the desired menu item by following the [SendMessage Tutorial](#).

Window titles and text are case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[SetKeyDelay](#), [Escape sequences \(e.g. `%\)](#), [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.
```

CoordMode

Sets coordinate mode for various commands to be relative to either the active window or the screen.

```
CoordMode, ToolTip|Pixel|Mouse|Caret|Menu [,  
Screen|Window|Client]
```

Parameters

Param1

ToolTip: Affects [ToolTip](#).

Pixel: Affects [PixelGetColor](#), [PixelSearch](#), and [ImageSearch](#).

Mouse: Affects [MouseGetPos](#), [Click](#), and [MouseMove/Click/Drag](#).

Caret: Affects the built-in variables [A_CaretX](#) and [A_CaretY](#).

Menu: Affects the [Menu Show](#) command when coordinates are specified for it.

Param2

If *Param2* is omitted, it defaults to `Screen`.

Screen: Coordinates are relative to the desktop (entire screen).

Relative: Coordinates are relative to the active window.

Window [v1.1.05+]: Synonymous with *Relative* and recommended for clarity.

Client [v1.1.05+]: Coordinates are relative to the active window's client area, which excludes the window's title bar, menu (if it has a standard one) and borders. Client coordinates are less dependent on OS version and theme.

Remarks

If this command is not used, all commands except those documented otherwise (e.g. [WinMove](#) and [InputBox](#)) use coordinates that are relative to the active window.

Every newly launched [thread](#) (such as a [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).

[v1.1.23+]: The built-in [A_CoordMode variables](#) contain the current settings.

Related

[Click](#), [MouseMove](#), [MouseClicked](#), [MouseClickedDrag](#), [MouseGetPos](#), [PixelGetColor](#), [PixelSearch](#), [ToolTip](#), [Menu](#)

Example

```
CoordMode, ToolTip, Screen ; Place ToolTips at
absolute screen coordinates:
CoordMode, ToolTip ; Same effect as the above
because "screen" is the default.
```

GetKeyName / GetKeyVK / GetKeySC

[v1.1.01+]

Retrieves the name/text, virtual key code or scan code of a key.

```
String := GetKeyName(Key)
Number := GetKeyVK(Key)
Number := GetKeySC(Key)
```

Parameters

Key

A VK or SC code, such as "vkA2" or "sc01D", a combination of both, or a [key name](#). For example, both `GetKeyName("vk1B")` and `GetKeyName("Esc")` return `Escape`, while `GetKeyVK("Esc")` returns `27`. Note that VK and SC codes must be in hexadecimal. To convert a decimal number to the appropriate format, use `Format("vk{:x}", vk_code)` or `Format("sc{:x}", sc_code)`.

Return Value

These functions return a name, virtual key code or scan code of *Key*.

Related

[GetKeyState](#), [Key List](#), [Format](#)

Examples

```
; Show information for a specific key.
key := "LWin" ; Any key can be used here.
```

```
name := GetKeyName(key)
vk    := GetKeyVK(key)
sc    := GetKeySC(key)
```

```
MsgBox, %
```

```
Format("Name: `t{}`nVK: `t{:X}`nSC: `t{:X}", name,
vk, sc)
```

GetKeyState() / GetKeyState

Checks if a keyboard key or mouse/joystick button is down or up. Also retrieves joystick status.

GetKeyState()

```
KeyIsDown := GetKeyState(KeyName [, Mode])
```

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.

Examples: B, 5, LWin, RControl, Alt, Enter, Escape, LButton, MButton, Joy1.

Alternatively, an explicit virtual key code such as vkFF may be specified. This is useful in the rare case where a key has no name. The virtual key code of such a key can be determined by following the steps at the bottom of the [key list page](#).

Known limitation: This function cannot differentiate between two keys which share the same virtual key code, such as Left and NumpadLeft.

Mode

This parameter is ignored when retrieving joystick status.

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: Retrieve the physical state (i.e. whether the user is physically holding it down). 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 (as long as it was pressed down while the script was running). You can determine if your script is using the hooks via the [KeyHistory](#) command or menu item. You can

force the hooks to be installed by adding the [#InstallKeybdHook](#) and/or [#InstallMouseHook](#) directives to the script.

T: Retrieve the toggle state (only valid for keys that can be toggled such as `CapsLock`, `NumLock`, `ScrollLock`, `Insert`, etc.) retrieved value of 1 (true) means the key is "on", while 0 (false) means it's "off".

Return Value

This function returns 1 if the key is down (or toggled on) or 0 if it is up (or toggled off). For the joystick's special keys such as axes and POV switch, [other values are retrieved](#).

If *KeyName* is invalid or the state of the key could not be determined, an empty string is returned.

GetKeyState

Deprecated: This command is not recommended for use in new scripts. Use the [GetKeyState](#) function instead.

```
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. For the joystick's special keys such as axes and POV switch, [other values are retrieved](#).

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.

Examples: B, 5, LWin, RControl, Alt, Enter, Escape, LButton, MButton, Joy1.

Alternatively, an explicit virtual key code such as vkFF may be specified. This is useful in the rare case where a key has no name. The virtual key code of such a key can be determined by following the steps at the bottom of the [key list page](#).

Known limitation: This command cannot differentiate between two keys which share the same virtual key code, such as Left and NumpadLeft.

Mode

This parameter is ignored when retrieving joystick status.

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: Retrieve the physical state (i.e. whether the user is physically holding it down). 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 (as long as it was pressed down while the script was running). You can determine if your script is using the hooks via the [KeyHistory](#) command or menu item. You can force the hooks to be installed by adding the [#InstallKeybdHook](#) and/or [#InstallMouseHook](#) directives to the script.

T: Retrieve the toggle state (only valid for keys that can be toggled such as `CapsLock`, `NumLock`, `ScrollLock`, `Insert`, etc.). A retrieved value of D means the key is "on", while U means it's "off".

Joystick's special keys

When *KeyName* is a joystick axis such as JoyX, the retrieved value will be 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).

When *KeyName* is JoyPOV, the retrieved value will be between 0 and 35900. The following approximate POV values are used by many joysticks:

- -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

Remarks

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.

Systems with unusual keyboard drivers might be slow to update the state of their keys, especially the toggle-state of keys like `CapsLock`  checks the state of such a key immediately after it changed may use [Sleep](#) beforehand to give the system time to update the key state.

For examples of using GetKeyState with a joystick, see the [joystick remapping page](#) and the [Joystick-To-Mouse script](#).

Related

[KeyWait](#), [Key List](#), [Joystick remapping](#), [KeyHistory](#), [#InstallKeybdHook](#), [#InstallMouseHook](#)

Examples

Example #1: Command vs. function

Although the first code block uses the command and the second one uses the function, these two blocks are functionally identical:

```
GetKeyState, state, RButton ; Right mouse
button.
GetKeyState, state, Joy2 ; The second button of
the first joystick.

GetKeyState, state, Shift
if state = D
    MsgBox At least one Shift key is down.
else
    MsgBox Neither Shift key is down.

GetKeyState, state, CapsLock, T ; D if CapsLock
is ON or U otherwise.
```

```
state := GetKeyState("RButton") ; Right mouse
button.
state := GetKeyState("Joy2") ; The second
button of the first joystick.

if GetKeyState("Shift")
    MsgBox At least one Shift key is down.
else
    MsgBox Neither Shift key is down.

state := GetKeyState("CapsLock", "T") ; True if
CapsLock is ON, false otherwise.
```

Example #2: Remapping

(This example is only for illustration because it would be easier to use the [built-in remapping feature](#).) In the following hotkey, 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::
MouseClicked, left,,, 1, 0, D ; Hold down the
left mouse button.
Loop
{
    Sleep, 10
    if !GetKeyState("NumpadAdd", "P") ; The key
has been released, so break out of the loop.
        break
    ; ... insert here any other actions you want
repeated.
}
MouseClicked, left,,, 1, 0, U ; Release the mouse
button.
return
```

Example #3

Make joystick button behavior depend on joystick axis position:

```
joy2::
JoyX := GetKeyState("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
```

Other examples

See the [joystick remapping page](#) and the [Joystick-To-Mouse script](#) for other examples.

KeyHistory

Displays script info and a history of the most recent keystrokes and mouse clicks.

KeyHistory

Remarks

This command is equivalent to selecting the "View->Key history" menu item in the main window.

To disable key history, specify the following line anywhere in the script:

```
#KeyHistory 0
```

`#KeyHistory` can also be used to change the maximum number of events that will be displayed.

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).

The virtual key (VK) of the wheel events (WheelDown, WheelUp, WheelLeft, and WheelRight) are placeholder values that do not have any meaning outside of AutoHotkey. Also, the scan code for wheel events is actually the number of notches by which the wheel was turned (typically 1).

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. You can find out if your script uses either hook via "View->Key History" in the script's main window (accessible via "Open" in the tray icon). You can force the hooks to be installed by adding either or both of the

following lines to the script:

```
#InstallKeybdHook  
#InstallMouseHook
```

Related

[#KeyHistory](#), [#InstallKeybdHook](#), [#InstallMouseHook](#), [ListHotkeys](#), [ListLines](#), [ListVars](#), [GetKeyState](#), [KeyWait](#), [A_PriorKey](#)

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 of 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 option is ignored for joystick buttons.

T: Timeout (e.g. `T3`). The number of seconds to wait before timing out

and setting [ErrorLevel](#) to 1. If the key or button achieves the specified state, the command will not wait for the timeout to expire. Instead, it will immediately set [ErrorLevel](#) to 0 and the script will continue executing.

The timeout 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

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.

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

To wait for two or more keys to be released, use `KeyWait` consecutively. For example:

```
KeyWait Control ; Wait for both Control and Alt  
to be released.  
KeyWait Alt
```

To wait for any one key among a set of keys to be pressed down, see the examples section of the [Input](#) command.

Related

[GetKeyState](#), [Key List](#), [Input](#), [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
(this is only for illustration because it
; would be easier to use the built-in remapping
feature):
; 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" or  
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.

OutputVar does not store keystrokes per se. Instead, it stores characters produced by keystrokes according to the active window's keyboard layout/language. Consequently, keystrokes that do not produce characters (such as `PageUp` `Home`) stored (though they can be recognized via the *EndKeys* parameter below).

Note: The `Escape` key is translated to the ASCII ESC character, `Chr(27)`. This can be avoided by including `{ESC}` in *EndKeys*, in which case pressing `Escape` will terminate `Input`.

Whitespace characters such as `Tab` `Enter` stored literally. `Enter` is stored as `linefeed (^n)`.

Options

A string of zero or more of the following letters (in any order, with optional spaces in between):

B: Backspace 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 deletes the last character rather than the one behind the caret (insertion point).

C: Case sensitive. Normally, *MatchList* is not case sensitive (in versions prior to 1.0.43.03, only the letters A-Z are recognized as having varying case, not letters like ü/Û).

I: Ignore input generated by any AutoHotkey script, such as the [SendEvent](#) command. However, the [SendInput](#) and [SendPlay](#) methods are always ignored, regardless of this setting.

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 *MatchList* 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: Modified keystrokes such as Control + A 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 + ←` 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 blocked (hidden from the system). 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. Use this option to find a match more often by searching the entire length of the input text.

E [v1.1.20+]: Handle single-character end keys by character code instead of by keycode. This provides more consistent results if the active window's keyboard layout is different to the script's keyboard layout. It also prevents key combinations which don't actually produce the given end characters from ending input; for example, if `@` is an end key, on the US layout `Shift + 2` trigger it but `Ctrl + Shift + 2` will not (if the E option is used). If the **C** option is also used, the end character is case-sensitive.

EndKeys

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`. Examples: `EndKey:.`, `EndKey:Escape`.

The `EndKey` list uses a format similar to the `Send` command. For example, specifying `{Enter}. {ESC}` would cause either `Enter` or `Escape` to terminate the Input. To use the braces themselves as end keys, specify `{{}}` and/or `{}}`.

To use `Control` `Alt` `Shift` keys, specify the left and/or right version of the key, not the neutral version. For example, specify `{LControl}{RControl}` rather than `{Control}`.

[v1.0.14+]: Although modified keys such as `Alt` `C` are not supported, certain characters that require the `Shift` held down -- namely punctuation marks such as `? ! : @ & { }` -- are supported.

[v1.1.20+]: Other characters are supported with the `E` option described above. When the `E` and `M` options are both used, `Control` `A` through `Control` `Z` supported by including the corresponding ASCII control characters in *EndKeys*.

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 of the [key list page](#).

MatchList

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`). The entirety of what the user types must exactly match one of the phrases for a match to occur (unless the `*` option is present). In addition, **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

variables contains a large sub-list of match phrases.

ErrorLevel

[v1.1.04+]: This command is able to throw an exception if called with no parameters and there is no Input in progress. For more information, see [Runtime Errors](#).

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	<p>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. <code>EndKey:Enter</code> , <code>EndKey:Escape</code> , etc.</p> <p>Note that <i>Name</i> is the "normalized" name of the key regardless of how it was written in <i>EndKeys</i>. For example, <code>{ESC}</code> and <code>{vk1B}</code> both produce <code>EndKey:Escape</code> . GetKeyName() can be used to retrieve the normalized name.</p> <p>If the E option was used, <i>Name</i> is the actual character which was typed (if applicable). Otherwise, the key name is determined according to the script's active keyboard layout.</p> <p>Prior to [v1.1.20], if the VK code of the end key was in the range 0x41 (A) through 0x5A (Z), ErrorLevel usually</p>

contained the corresponding ASCII character even if that wasn't correct for the current keyboard layout. In [v1.1.20] and later, the correct character is used. If a character in the range A through Z is used, it is upper-case for backward-compatibility; otherwise it is usually lower-case.

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.

While an Input is in progress, new [threads](#) such as [custom menu items](#) and [timed subroutines](#) can still be created. Similarly, keyboard [hotkeys](#) are still in effect if the Input is [visible](#). If the Input is not visible, only [hook hotkeys](#) can be triggered.

When a script first uses this command, the [keyboard hook](#) is installed (if it is not already). In addition, the script becomes [persistent](#), meaning that [ExitApp](#) should be used to terminate it. The keyboard hook will stay installed until the next use of the [Suspend](#) or [Hotkey](#) command, at which time it is removed if not required by any hotkeys or hotstrings.

If you use multiple languages or keyboard layouts, Input uses the keyboard layout of the active window rather than the script's (regardless of whether the Input is [visible](#)). However, in versions prior to 1.0.44.03, the script's own layout is used.

Although not as flexible, [hotstrings](#) are generally easier to use than the Input command.

Related

[KeyWait](#), [Hotstrings](#), [InputBox](#), [#InstallKeybdHook](#), [Threads](#), [if var in/contains MatchList](#)

Examples

```
; Wait for the user to press any key. Keys that
produce no visible character -- such as
; the modifier keys, function keys, and arrow keys
-- are listed as end keys so that they
; will be detected too.
```

```
Input, SingleKey, L1, {LControl}{RControl}{LAlt}
{RAlt}{LShift}{RShift}{LWin}{RWin}{AppsKey}{F1}
{F2}{F3}{F4}{F5}{F6}{F7}{F8}{F9}{F10}{F11}{F12}
{Left}{Right}{Up}{Down}{Home}{End}{PgUp}{PgDn}
{Del}{Ins}{BS}{CapsLock}{NumLock}{PrintScreen}
{Pause}
```

```
; This is a working hotkey example. Since the
hotkey has the tilde (~)
; prefix, its own keystroke will pass through to
the active window.
```

```
; 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, f1, 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
If InStr(ErrorLevel, "EndKey:")
{
    MsgBox, You entered "%UserInput%" and
terminated the input with %ErrorLevel%.
    return
}
; Otherwise, a match was found.
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, https://autohotkey.com
return
```

MouseClicked

Clicks or holds down a mouse button, or turns the mouse wheel. NOTE: The [Click command](#) is generally more flexible and easier to use.

```
MouseClicked [, WhichButton, X, Y, ClickCount, Speed, D|U, R]
```

Parameters

WhichButton

The button to click: Left (default), Right, Middle (or just the first letter of each of these); or the fourth or fifth mouse button (X1 or X2). For example: `MouseClicked, X1`. This parameter may be omitted, in which case it defaults to Left.

Rotate the mouse wheel: Specify WheelUp or WU to turn the wheel upward (away from you); specify WheelDown or WD to turn the wheel downward (toward you). [v1.0.48+]: Specify WheelLeft (or WL) or WheelRight (or WR) to push the wheel left or right, respectively (but these have no effect on operating systems older than Windows Vista). *ClickCount* is the number of notches to turn the wheel.

To compensate automatically for cases where the user has swapped the left and right mouse buttons via the system's control panel, use the [Click command](#) instead.

X, Y

The x/y coordinates to which the mouse cursor is moved prior to clicking, which can be [expressions](#). Coordinates are relative to the active window unless [CoordMode](#) was used to change that. If omitted, the cursor's current position is used.

ClickCount

The number of times to click the mouse, which can be an [expression](#). If

omitted, the button is clicked once.

Speed

The speed to move the mouse in the range 0 (fastest) to 100 (slowest), which can be an [expression](#).

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.

Speed is ignored for [SendInput/Play modes](#); they move the mouse instantaneously (though [SetMouseDelay](#) has a mode that applies to [SendPlay](#)). To visually move the mouse more slowly -- such as a script that performs a demonstration for an audience -- use [SendEvent {Click 100, 200}](#) or [SendMode Event](#) (optionally in conjunction with [BlockInput](#)).

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

This command uses the sending method set by [SendMode](#).

The [Click command](#) is recommended over [MouseClicked](#) because:

1. It automatically compensates when the left and right mouse buttons are

- swapped via the control panel.
2. It is generally easier to use.

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}  
MouseClicked, left, 55, 233  
Send, {Control up}
```

```
; Example #2:  
Send, {Shift down}  
MouseClicked, left, 55, 233  
Send, {Shift up}
```

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. The *speed* parameter or [SetDefaultMouseSpeed](#) can be used to reduce the speed (in the default [SendEvent mode](#) only).

Some applications do not obey a *ClickCount* higher than 1 for the mouse wheel. For them, use a [Loop](#) such as the following:

```
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. However, this is generally not needed for the [SendInput/Play](#) 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

[CoordMode](#), [SendMode](#), [SetDefaultMouseSpeed](#), [SetMouseDelay](#), [Click](#), [MouseClickedDrag](#), [MouseGetPos](#), [MouseMove](#), [ControlClick](#), [BlockInput](#)

Examples

```
; Double click at the current mouse pos:
MouseClicked, left
MouseClicked, left

; Same as above:
MouseClicked, left, , , 2

; Move to specified coordinates then right-click
once:
MouseClicked, right, 200, 300

; Here are two hotkeys that simulate the turning
of the mouse wheel:
#up::MouseClicked, WheelUp, , , 2 ; Turn it by two
notches.
#down::MouseClicked, WheelDown, , , 2
```

MouseClickedDrag

Clicks and holds the specified mouse button, moves the mouse to the destination coordinates, then releases the button.

```
MouseClickedDrag, WhichButton, X1, Y1, X2, Y2 [, Speed, R]
```

Parameters

WhichButton

The button to click: Left, Right, Middle (or just the first letter of each of these). Specify X1 for the fourth button and X2 for the fifth. For example:
`MouseClickedDrag, X1,`

To compensate automatically for cases where the user has swapped the left and right mouse buttons via the system's control panel, use the [Click command](#) instead.

X1, Y1

The x/y coordinates of the drag's starting position, which can be [expressions](#) (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), which can be [expressions](#). 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), which can be an [expression](#).

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.

Speed is ignored for [SendInput/Play modes](#); they move the mouse instantaneously (though [SetMouseDelay](#) has a mode that applies to [SendPlay](#)). To visually move the mouse more slowly -- such as a script that performs a demonstration for an audience -- use [SendEvent {Click 100, 200}](#) or [SendMode Event](#) (optionally in conjunction with [BlockInput](#)).

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

This command uses the sending method set by [SendMode](#).

Dragging can also be done via the various [Send](#) commands, which is more flexible because the mode can be specified via the command name. For example:

```
SendEvent {Click 6, 52, down}{click 45, 52, up}
```

Another advantage of the method above is that unlike [MouseClickDrag](#), it automatically compensates when the user has swapped the left and right mouse buttons via the system's control panel.

The [SendPlay mode](#) is able to successfully generate mouse events in a broader variety of games than the other modes. However, dragging via SendPlay might not work in RichEdit controls (and possibly others) such as those of WordPad and Metapad.

Some applications and games may have trouble tracking the mouse if it moves too quickly. The *speed* parameter or [SetDefaultMouseSpeed](#) can be used to reduce the speed (in the default [SendEvent mode](#) only).

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/Play](#) 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](#)). 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](#), [SendMode](#), [SetDefaultMouseSpeed](#), [SetMouseDelay](#), [Click](#), [MouseClicked](#), [MouseGetPos](#), [MouseMove](#), [BlockInput](#)

Example

```
MouseClickedDrag, left, 0, 200, 600, 400
```

```
; The following example opens MS Paint and draws a  
little house:
```

```
Run, mspaint.exe
```

```
WinWaitActive, ahk_class MSPaintApp,, 2
```

```
if ErrorLevel
```

```
    return
```

```
MouseClickedDrag, L, 150, 250, 150, 150
```

```
MouseClickedDrag, L, 150, 150, 200, 100
```

```
MouseClickedDrag, L, 200, 100, 250, 150
```

MouseClickedDrag, L, 250, 150, 150, 150
MouseClickedDrag, L, 150, 150, 250, 250
MouseClickedDrag, L, 250, 250, 250, 150
MouseClickedDrag, L, 250, 150, 150, 250
MouseClickedDrag, L, 150, 250, 250, 250

MouseGetPos

Retrieves the current position of the mouse cursor, and optionally which window and control it is hovering over.

```
MouseGetPos [, OutputVarX, OutputVarY, OutputVarWin,  
OutputVarControl, 1|2|3]
```

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.

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 (ClassNN) 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 distributed with [v1.0.14+] (but not necessarily older versions 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.

If omitted, it defaults to 0. Otherwise, specify one of the following digits:

1: Uses a simpler method to determine *OutputVarControl*. This method correctly retrieves the active/topmost child window of an Multiple Document Interface (MDI) application such as SysEdit or TextPad. However, it is less accurate for other purposes such as detecting controls inside a GroupBox control.

2 [v1.0.43.06+]: Stores the [control's HWND](#) in *OutputVarControl* rather than the control's ClassNN.

3 [v1.0.43.06+]: A combination of 1 and 2 above.

Remarks

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

Related

[CoordMode](#), [WinGet](#), [SetDefaultMouseSpeed](#), [Click](#)

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, which can be [expressions](#). 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), which can be an [expression](#).

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.

Speed is ignored for [SendInput/Play modes](#); they move the mouse instantaneously (though [SetMouseDelay](#) has a mode that applies to [SendPlay](#)). To visually move the mouse more slowly -- such as a script that performs a demonstration for an audience -- use [SendEvent {Click 100, 200}](#) or [SendMode Event](#) (optionally in conjunction with [BlockInput](#)).

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

This command uses the sending method set by [SendMode](#).

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. The *speed* parameter or [SetDefaultMouseSpeed](#) can be used to reduce the speed (in the default [SendEvent mode](#) only).

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/Play](#) modes because they automatically postpone the user's physical mouse activity until afterward.

There is an automatic delay after every movement of the mouse (except for [SendInput mode](#)). Use [SetMouseDelay](#) to change the length of the delay.

The following is an alternate way to move the mouse cursor that may work better in certain multi-monitor configurations:

```
DllCall("SetCursorPos", int, 100, int, 400) ;  
The first number is the X-coordinate and the  
second is the Y (relative to the screen).
```

On a related note, the mouse cursor can be temporarily hidden via the [hide-cursor example](#).

Related

[CoordMode](#), [SendMode](#), [SetDefaultMouseSpeed](#), [SetMouseDelay](#), [Click](#), [MouseClicked](#), [MouseClickedDrag](#), [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
```

Send[Raw|Input|Play|Event]: Send Keys & Clicks

Sends simulated keystrokes and mouse clicks to the [active](#) window.

Send Keys

SendRaw Keys

SendInput Keys

SendPlay Keys

SendEvent Keys

Parameters

Keys

The sequence of keys to send. As with other commands, the comma in front of the first parameter is optional.

Raw mode - `SendRaw` or `{Raw}`: The characters `^+!#{}` are interpreted literally rather than translating `{Enter}` to `Enter` `^C` `Control` `C`. To use raw mode with `SendInput`, `SendPlay`, `SendEvent`, or `ControlSend`, write `{Raw}` as the first item in the string; for example: `SendInput {Raw}abc`.

Raw mode does not affect the interpretation of [escape sequences](#), [variable references](#) and [expressions](#). For example, `SendRaw, ``100`%` sends the string ``100%`. When using `ControlSend`, it is also necessary to escape literal commas (`` , ``).

Text mode - `{Text}` [v1.1.27+]: Similar to Raw mode, except that no attempt is made to translate characters (other than ``r`, ``n`, ``t` and ``b`) to keycodes; instead, the [fallback method](#) is used for all of the remaining characters. For `SendEvent`, `SendInput` and `ControlSend`, this improves reliability because the characters are much less dependent on correct modifier state. Text mode can be

combined with Blind mode to avoid releasing any modifier keys: `Send {Blind}{Text}your text`. However, some applications require that the modifier keys be released.

``n`, ``r` and ``r`n` are all translated to a single `Enter`, unlike normal mode and Raw mode, which translate ``r`n` to two `Enter` keystrokes. ``t` is translated to `Tab` to `Backspace` other characters are sent without translation.

Normal mode: When not in Raw mode or Text mode, the following symbols have special meaning: `!+^#{}`

The modifiers `!+^#` affect only the very next key. To send the corresponding modifier key on its own, enclose the key name in braces. To just press (hold down) or release the key, follow the key name with the word "down" or "up" as shown below.

Symbol	Key	Press	Release	Examples
!	{Alt}	{Alt down}	{Alt up}	Send <code>!a</code> presses <code>Alt</code> <code>A</code>
+	{Shift}	{Shift down}	{Shift up}	Send <code>+abC</code> sends the text "AbC" Send <code>!+a</code> presses <code>Alt</code> <code>Shift</code> <code>A</code>
^	{Ctrl}	{Ctrl down}	{Ctrl up}	Send <code>^{Home}</code> presses <code>Control</code> <code>Home</code>
#	{LWin} {RWin}	{LWin down} {RWin down}	{LWin up} {RWin up}	Send <code>#e</code> holds down the <code>Win</code> and then presses the <code>E</code>
Symbol	Meaning			

Braces are used to enclose [key names and other options](#), and to send special characters literally. For example, `{Tab}` is the

`Tab` and `{!}` is a literal exclamation mark.

{ }

[v1.1.27+]: Enclosing a plain ASCII letter (a-z or A-Z) in braces forces it to be sent as the corresponding virtual keycode, even if the character does not exist on the current keyboard layout. In other words, `Send a` produces the letter "a" while `Send {a}` may or may not produce "a", depending on the keyboard layout. For details, see the [remarks below](#).

Note: As capital letters are produced by sending the `Shift` key, `A` produces a different effect in some programs than `a`. For example, `!A` presses `Alt + Shift + A` and `!a` presses `Alt + A`. If in doubt, use lowercase.

Blind mode - `{Blind}`: Blind mode 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 Blind mode. It has the following effects:

- Blind mode avoids releasing `Alt`, `Control`, `Shift`, `Win` 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`. `^space::Send {Blind}{Ctrl up}` allows `Ctrl` logically up even though it is physically down.
- [SetStoreCapsLockMode](#) is ignored; that is, the state of `CapsLock` 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** [redacted] keys (LWin/RWin); 2) menu bar activation during **Alt** [redacted] keys. However, Blind mode does not prevent masking performed by the keyboard hook following activation of a hook hotkey.

- Send does not wait for the **Win** [redacted] to be released even if the text contains an **L** [redacted] stroke. This would normally be done to prevent Send from triggering the system "lock workstation" hotkey (**Win** [redacted] **L** [redacted] see [Hotkeys](#) for details).

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) **Control** [redacted] **B** [redacted] you type

Control [redacted] **A** [redacted]

{Blind} is not supported by `SendRaw` or `ControlSendRaw`; use **{Blind}{Raw}** instead.

Blind mode is not completely supported by `SendPlay`, especially when dealing with the modifier keys (**Control** [redacted] **Alt** [redacted] **Shift** [redacted] **Win** [redacted])

SendInput and **SendPlay** [v1.0.43+]: `SendInput` and `SendPlay` use the same syntax as `Send` but are generally faster and more reliable. In addition, they buffer any physical keyboard or mouse activity during the send, which prevents the user's keystrokes from being interspersed with those being sent. `SendMode` can be used to make `Send` synonymous with `SendInput` or `SendPlay`. For more details about each mode, see [SendInput](#) and [SendPlay](#) below.

SendEvent [v1.0.43+]: `SendEvent` sends keystrokes using the same method as the pre-1.0.43 `Send` command. The rate at which keystrokes are sent is determined by `SetKeyDelay`. By default, `Send` is synonymous `SendEvent`; but it can be made a synonym for `SendInput` or `SendPlay` via `SendMode`.

Key Names: The following table lists the special keys that can be sent (each key name must be enclosed in braces):

--

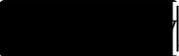
{F1} - {F24}	Function keys. For example: {F12} is the  key.
{!}	!
{#}	#
{+}	+
{^}	^
{{}	{
{}}	}
{Enter}	 the main keyboard
{Escape} or {Esc}	
{Space}	 ly needed for spaces that appear either at the beginning or the end of the string to be sent -- ones in the middle can be literal spaces)
{Tab}	
{Backspace} or {BS}	
{Delete} or {Del}	
{Insert} or {Ins}	
{Up}	 (up arrow) key on main keyboard
{Down}	 (down arrow) key on main keyboard
{Left}	 (left arrow) key on main keyboard
{Right}	 (right arrow) key on main keyboard
{Home}	 main keyboard
{End}	

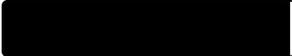
	End [redacted] main keyboard
{PgUp}	PageUp [redacted] in keyboard
{PgDn}	PageDown [redacted] keyboard
{CapsLock}	CapsLock [redacted] SetCapsLockState is more reliable on Win 2k/XP). Sending {CapsLock} might require SetStoreCapsLockMode Off beforehand.
{ScrollLock}	ScrollLock [redacted] SetScrollLockState)
{NumLock}	NumLock [redacted] SetNumLockState)
{Control} or {Ctrl}	Control [redacted] (info: sends the neutral virtual key but the left scan code)
{LControl} or {LCtrl}	Left Control [redacted] (technical info: sends the left virtual key rather than the neutral one)
{RControl} or {RCtrl}	Right Control [redacted]
{Control down} or {Ctrl down}	Holds the Control [redacted] until {Ctrl up} is sent. To hold down the left or right key instead, use {RCtrl down} and {RCtrl up}.
{Alt}	Alt [redacted] (technical info: sends the neutral virtual key but the left scan code)
{LAlt}	Left Alt [redacted] (technical info: sends the left virtual key rather than the neutral one)
{RAlt}	Right Alt [redacted] or AltGr [redacted] (depending on keyboard layout)

{Alt down}	Holds the Alt [redacted] down until {Alt up} is sent. To hold down the left or right key instead, use {RAlt down} and {RAlt up}.
{Shift}	Shift [redacted] info: sends the neutral virtual key but the left scan code)
{LShift}	Left Shift [redacted] (technical info: sends the left virtual key rather than the neutral one)
{RShift}	Right Shift [redacted]
{Shift down}	Holds the Shift [redacted] until {Shift up} is sent. To hold down the left or right key instead, use {RShift down} and {RShift up}.
{LWin}	Left Win [redacted]
{RWin}	Right Win [redacted]
{LWin down}	Holds the left Win [redacted] down until {LWin up} is sent
{RWin down}	Holds the right Win [redacted] down until {RWin up} is sent
{AppsKey}	Menu [redacted] (triggers the right-click or context menu)
{Sleep}	Sleep [redacted]
	Sends an Alt [redacted] keypad combination, which can be used to generate special characters that don't exist on the keyboard. To generate ASCII characters,

<p>{ASC nnnnn}</p>	<p>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}.</p> <p>Unicode characters may be generated by specifying a number between 256 and 65535 (without a leading zero). However, this is not supported by all applications. For alternatives, see the section below.</p>
<p>{U+nnnn}</p>	<p>[AHK_L 24+]: Sends a Unicode character where <i>nnnn</i> is the hexadecimal value of the character excluding the 0x prefix. This typically isn't needed in Unicode versions of AutoHotkey, where Send and ControlSend automatically support Unicode text.</p> <p>SendInput() or WM_CHAR is used to send the character and the current Send mode has no effect. Characters sent this way usually do not trigger shortcut keys or hotkeys.</p>
<p>{vkXX} {scYYY} {vkXXscYYY}</p>	<p>Sends a keystroke that has virtual key XX and scan code YYY. For example: <code>Send {vkFFsc159}</code>. If the sc or vk portion is omitted, the most appropriate value is sent in its place.</p> <p>The values for XX and YYY are hexadecimal and can usually be determined from the main window's View->Key history menu item. See also: Special Keys</p> <div style="background-color: #fff9c4; padding: 10px; border: 1px solid #ccc;"> <p>Warning: Combining vk and sc in this manner is valid only with Send. Prior to [v1.1.27], hotkeys permitted but ignored any non-hexadecimal characters following XX.</p> </div>

{Numpad0} - {Numpad9}	Numpad digit keys (as seen when  ON). For example: {Numpad5} is 
{NumpadDot}	 (Numpad period) (as seen when  ON).
{NumpadEnter}	 keypad
{NumpadMult}	 (Numpad multiplication)
{NumpadDiv}	 (Numpad division)
{NumpadAdd}	 (Numpad addition)
{NumpadSub}	 (Numpad subtraction)
{NumpadDel}	 (this key and the following Numpad keys are used when  is OFF)
{NumpadIns}	 keypad
{NumpadClear}	Clear key on keypad (usually the '5' key when 
{NumpadUp}	 (up arrow) key on keypad
{NumpadDown}	 (down arrow) key on keypad
{NumpadLeft}	 (left arrow) on keypad
{NumpadRight}	 (right arrow) key on keypad
{NumpadHome}	 keypad
{NumpadEnd}	 keypad

{NumpadPgUp}	PageUp  pad
{NumpadPgDn}	PageDown 
{Browser_Back}	Select the browser "back" button
{Browser_Forward}	Select the browser "forward" button
{Browser_Refresh}	Select the browser "refresh" button
{Browser_Stop}	Select the browser "stop" button
{Browser_Search}	Select the browser "search" button
{Browser_Favorites}	Select the browser "favorites" button
{Browser_Home}	Launch the browser and go to the home page
{Volume_Mute}	Mute/unmute the master volume. Usually equivalent to <code>SoundSet, +1, , mute</code> .
{Volume_Down}	Reduce the master volume. Usually equivalent to <code>SoundSet -5</code> .
{Volume_Up}	Increase the master volume. Usually equivalent to <code>SoundSet +5</code> .
{Media_Next}	Select next track in media player
{Media_Prev}	Select previous track in media player
{Media_Stop}	Stop media player
{Media_Play_Pause}	Play/pause media player
{Launch_Mail}	Launch the email application
{Launch_Media}	Launch media player
{Launch_App1}	Launch user app1
{Launch_App2}	Launch user app2

{PrintScreen}	PrintScreen 
{CtrlBreak}	Ctrl  Break 
{Pause}	Pause 
{Click [Options]} [v1.0.43+]	Sends a mouse click using the same options available in the Click command . For example, <code>{Click}</code> would click the left mouse button once at the mouse cursor's current position, and <code>{Click 100, 200}</code> would click at coordinates 100, 200 (based on CoordMode). To move the mouse without clicking, specify 0 after the coordinates; for example: <code>{Click 100, 200, 0}</code> . The delay between mouse clicks is determined by SetMouseDelay (not SetKeyDelay).
{WheelDown}, {WheelUp}, {WheelLeft}, {WheelRight}, {LButton}, {RButton}, {MButton}, {XButton1}, {XButton2}	Sends a mouse button event at the cursor's current position (to have control over position and other options, use Click above). The delay between mouse clicks is determined by SetMouseDelay . WheelLeft/Right require [v1.0.48+], but have no effect on operating systems older than Windows Vista.
{Blind}	Enables Blind mode , which gives the script more control by disabling a number of things that are normally done automatically to make things generally work as expected. The string <code>{Blind}</code> must occur at the beginning of the string.
{Raw} [v1.0.43+]	Enables Raw mode , which causes the following characters to be interpreted literally: <code>^+!#{}</code> . Although the string <code>{Raw}</code> need not occur at the beginning of the string, once specified, it stays in effect for the remainder of the string.

{Text}

[v1.1.27+]

Enables *Text mode*, which sends a stream of characters rather than keystrokes. Like Raw mode, Text mode causes the following characters to be interpreted literally: `^+!#{}`. Although the string `{Text}` need not occur at the beginning of the string, once specified, it stays in effect for the remainder of the string.

Repeating or Holding Down a Key

To repeat a keystroke: Enclose in braces the name of the key followed by the number of times to repeat it. For example:

```
Send {DEL 4} ; Presses the Delete key 4 times.
Send {S 30} ; Sends 30 uppercase S characters.
Send +{TAB 4} ; Presses Shift-Tab 4 times.
```

To hold down or release a key: Enclose in braces the name of the key followed by the word **Down** or **Up**. For example:

```
Send {b down}{b up}
Send {TAB down}{TAB up}
Send {Up down} ; Press down the up-arrow key.
Sleep 1000 ; Keep it down for one second.
Send {Up up} ; Release the up-arrow key.
```

When a key is held down via the method above, it does not begin auto-repeating like it would if you were physically holding it down (this is because auto-repeat is a driver/hardware feature). However, a [Loop](#) can be used to simulate auto-repeat. The following example sends 20 tab keystrokes:

```
Loop 20
{
    Send {Tab down} ; Auto-repeat consists of
consecutive down-events (with no up-events).
    Sleep 30 ; The number of milliseconds
between keystrokes (or use SetKeyDelay).
}
Send {Tab up} ; Release the key.
```

By default, Send will not automatically release a modifier key (**Control** 

`Shift` `Alt` `Win` modifier key was "pressed down" by sending it. For example, `Send a` may behave similar to `Send {Blind}{Ctrl up}a{Ctrl down}` if the user is physically holding `Ctrl` `Send` `{Ctrl Down}` followed by `Send a` will produce a `Control` `A` keystroke. *DownTemp* and *DownR* can be used to override this behavior. *DownTemp* and *DownR* have the same effect as *Down* except for the modifier keys (`Control` `Shift` `Alt` `Win`).

DownTemp tells subsequent sends that the key is not permanently down, and may be released whenever a keystroke calls for it. For example, `Send {Control DownTemp}` followed later by `Send a` would produce a normal `A` stroke, not a `Control` `A` stroke. Any use of `Send` may potentially release the modifier permanently, so *DownTemp* is not ideal for [remapping](#) modifier keys.

[v1.1.27+]: **DownR** (where "R" stands for [remapping](#), which is its main use) tells subsequent sends that if the key is automatically released, it should be pressed down again when send is finished. For example, `Send {Control DownR}` followed later by `Send a` would produce a normal `A` stroke, not a `Control` `A` stroke, but will leave the `Control` pressed state for use with keyboard shortcuts. In other words, *DownR* has an effect similar to physically pressing the key.

If a character does not correspond to a virtual key on the current keyboard layout, it cannot be "pressed" or "released". For example, `Send {µ up}` has no effect on most layouts, and `Send {µ down}` is equivalent to `Send µ`.

General Remarks

Characters vs. keys: By default, characters are sent by first translating them to keystrokes. If this translation is not possible (that is, if the current keyboard layout does not contain a key or key combination which produces that character), the character is sent by one of following fallback methods:

- SendEvent and SendInput use `SendInput()` with the `KEYEVENTF_UNICODE` flag. [v1.1.27+]: ANSI builds of AutoHotkey convert the character to Unicode before sending it. Prior to v1.1.27, ANSI builds used the `Alt+n` method.
- SendPlay uses the `Alt+n` method, which produces Unicode only if supported by the target application.
- ControlSend posts a `WM_CHAR` message.

Note: Characters sent using any of the above methods usually do not trigger keyboard shortcuts or hotkeys.

[v1.1.27+]: For characters in the range **a-z** or **A-Z** (plain ASCII letters), each character which does not exist in the current keyboard layout may be sent either as a character or as the corresponding virtual keycode (vk41-vk5A):

- If a naked letter is sent (that is, without modifiers or braces), or if `Raw` mode is in effect, it is sent as a character. For example, `Send {Raw}Regards` sends the expected text, even though pressing the `R` key (vk52) produces some other character (such as `К` in the Russian layout). `{Raw}` can be omitted in this case, unless a modifier key was put into effect by a prior `Send`.
- If one or more modifier keys have been put into effect by the `Send` command, or if the letter is wrapped in braces, it is sent as a keycode (modified with `Shift` if it is upper-case). This allows the script to easily activate standard keyboard shortcuts. For example, `^c` and `{Ctrl down}c{Ctrl up}` activate the standard `Ctrl+C` shortcut and `{C}` is equivalent to `{vk43}`.

If the letter exists in the current keyboard layout, it is always sent as whichever keycode the layout associates with that letter (unless [Text mode](#) is used, in which case the character is sent by other means). In other words, the section above is only relevant for non-Latin based layouts such as Russian.

Modifier State: When Send is required to change the state of the `Win` `Alt` or keys (such as if the user was holding one of those keys), it may inject additional keystrokes (`Ctrl` `Alt`) to prevent the Start menu or window menu from appearing. For details, see [#MenuMaskKey](#).

BlockInput Compared to SendInput/SendPlay: Although the [BlockInput](#) command can be used to prevent any keystrokes physically typed by the user from disrupting the flow of simulated keystrokes, it is often better to use [SendInput](#) or [SendPlay](#) so that keystrokes and mouse clicks become uninterruptible. This is because unlike BlockInput, SendInput/Play does not discard what the user types during the send; instead, such keystrokes are buffered and sent afterward.

When sending a large number of keystrokes, a [continuation section](#) can be used to improve readability and maintainability.

Since the operating system does not allow simulation of the `Ctrl` `Alt` `Delete` combination, doing something like `Send ^!{Delete}` will have no effect.

Send may have no effect on Windows Vista or later if the active window is running with administrative privileges and the script is not. This is due to a security mechanism called User Interface Privilege Isolation.

SendInput [v1.0.43+]

SendInput is generally the preferred method to send keystrokes and mouse clicks because of its superior speed and reliability. Under most conditions, SendInput is nearly instantaneous, even when sending long strings. Since SendInput is so fast, it is also more reliable because there is less opportunity for some other window to pop up unexpectedly and intercept the keystrokes. Reliability is further improved by the fact that anything the user types during a SendInput is postponed until afterward.

Unlike the other sending modes, the operating system limits SendInput to about 5000 characters (this may vary depending on the operating system's version and performance settings). Characters and events beyond this limit are not sent.

Note: SendInput ignores SetKeyDelay because the operating system does not support a delay in this mode. However, when SendInput reverts to SendEvent under the conditions described below, it uses SetKeyDelay -1, 0 (unless SendEvent's KeyDelay is -1, -1, in which case -1, -1 is used). When SendInput reverts to SendPlay, it uses SendPlay's KeyDelay.

If a script *other than* the one executing SendInput has a [low-level keyboard hook](#) installed, SendInput automatically reverts to SendEvent (or SendPlay if [SendMode InputThenPlay](#) is in effect). This is done because the presence of an external hook disables all of SendInput's advantages, making it inferior to both SendPlay and SendEvent. However, since SendInput is unable to detect a low-level hook in programs other than [AutoHotkey v1.0.43+], it will not revert in these cases, making it less reliable than SendPlay/Event.

When SendInput sends mouse clicks by means such as [{Click}](#), and [CoordMode Mouse, Relative](#) is in effect (the default), every click will be relative to the window that was active at the start of the send. Therefore, if SendInput intentionally activates another window (by means such as alt-tab), the coordinates of subsequent clicks within the same command will be wrong because they will still be relative to the old window rather than the new one.

SendPlay [v1.0.43+]

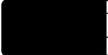
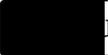
Warning: SendPlay may have no effect at all if UAC is enabled, even if the script is running as an administrator. For more information, refer to the [FAQ](#).

SendPlay's biggest advantage is its ability to "play back" keystrokes and mouse clicks in a broader variety of games than the other modes. For example, a particular game may accept [hotstrings](#) only when they have the [SendPlay option](#).

Of the three sending modes, SendPlay is the most unusual because it does not simulate keystrokes and mouse clicks per se. Instead, it creates a series of events (messages) that flow directly to the active window (similar to [ControlSend](#), but at a lower level). Consequently, SendPlay does not trigger hotkeys or hotstrings.

Like [SendInput](#), SendPlay's keystrokes do not get interspersed with keystrokes typed by the user. Thus, if the user happens to type something during a SendPlay, those keystrokes are postponed until afterward.

Although SendPlay is considerably slower than SendInput, it is usually faster than the traditional [SendEvent](#) mode (even when [KeyDelay](#) is -1).

The [Win](#)  (LWin and RWin) are automatically blocked during a SendPlay if the [keyboard hook](#) is installed. This prevents the Start Menu from appearing if the user accidentally presses a [Win](#)  during the send. By contrast, keys other than LWin and RWin do not need to be blocked because the operating system automatically postpones them until after the SendPlay (via buffering).

SendPlay does not use the standard settings of [SetKeyDelay](#) and [SetMouseDelay](#). Instead, it defaults to no delay at all, which can be changed as shown in the following examples:

```
SetKeyDelay, 0, 10, Play ; Note that both 0 and  
-1 are the same in SendPlay mode.  
SetMouseDelay, 10, Play
```

SendPlay is unable to turn on or off **CapsLock** **NumLock** **ScrollLock** unable to change a key's state as seen by [GetKeyState](#) unless the keystrokes are sent to one of the script's own windows. Even then, any changes to the left/right modifier keys (e.g. RControl) can be detected only via their neutral counterparts (e.g. Control). Also, SendPlay has other limitations described on the [SendMode page](#).

Unlike [SendInput](#) and [SendEvent](#), the user may interrupt a SendPlay by pressing **Control** **Alt** **Del** **Control** **Escape** happens, the remaining keystrokes are not sent but the script continues executing as though the SendPlay had completed normally.

Although SendPlay can send LWin and RWin events, they are sent directly to the active window rather than performing their native operating system function. To work around this, use [SendEvent](#). For example, `SendEvent #r` would show the Start Menu's Run dialog.

Related

[SendMode](#), [SetKeyDelay](#), [SetStoreCapsLockMode](#), [Escape sequences \(e.g. `%\)](#), [ControlSend](#), [BlockInput](#), [Hotstrings](#), [WinActivate](#)

Examples

Send Sincerely,{enter}John Smith ; Types a two-line signature.

Send !fs ; Select the File->Save menu (Alt+F followed by S).

Send {End}+{Left 4} ; Jump to the end of the text then send four shift+left-arrow keystrokes.

SendInput {Raw}A long series of raw characters sent via the fastest method (SendInput).

SendLevel [v1.1.06+]

Controls which artificial keyboard and mouse events are ignored by hotkeys and hotstrings.

SendLevel, Level

Parameters

Level

■ An integer between 0 and 100, which can be an [expression](#).

General Remarks

By default, [hook hotkeys](#) and [hotstrings](#) ignore keyboard and mouse events generated by any AutoHotkey script. In some cases it can be useful to override this behaviour; for instance, to allow a remapped key to be used to trigger other hotkeys. SendLevel and [#InputLevel](#) provide the means to achieve this.

SendLevel sets the level for events generated by the current [script thread](#), while #InputLevel sets the level for any hotkeys or hotstrings beneath it. For any event generated by a script to trigger a hook hotkey or hotstring, the send level of the event must be higher than the input level of the hotkey or hotstring.

Compatibility:

- [SendPlay](#) is not affected by SendLevel.
- [SendInput](#) is affected by SendLevel, but the script's own hook hotkeys cannot be activated while a SendInput is in progress, since it temporarily deactivates the hook. However, when Send or SendInput [reverts to SendEvent](#), it is able to activate the script's own hotkeys.
- Hotkeys using the "reg" method are incapable of distinguishing physical and artificial input, so are not affected by SendLevel. However, hotkeys above level 0 always use the keyboard or mouse hook.
- Auto-replace [hotstrings](#) always generate keystrokes at level 0, since it is usually undesirable for the replacement text to trigger another hotstring or

hotkey. To work around this, use a non-auto-replace hotstring and the SendEvent command.

- Characters sent by the [ASC \(Alt+nnnnn\)](#) method cannot trigger a hotstring, even if SendLevel is used.
- Characters sent by SendEvent with the [{Text}](#) mode, [{U+nnnn}](#) or [Unicode fallback method](#) can trigger hotstrings.

[v1.1.23+]: The built-in variable **A_SendLevel** contains the current setting.

Every newly launched hotkey or hotstring [thread](#) starts off with a send level equal to the [input level](#) of the hotkey or hotstring. Every other newly launched thread (such as a [custom menu item](#) or [timed](#) subroutine) starts off fresh with the default setting, which is typically 0 but may be changed by using this command in the [auto-execute section](#).

If SendLevel is used in the [auto-execute section](#), it also affects [keyboard and mouse remapping](#).

AutoHotkey versions older than [v1.1.06] behave as though `#InputLevel 0` and `SendLevel 0` are in effect.

Related

[#InputLevel](#), [Send](#), [Click](#), [MouseClicked](#), [MouseClickedDrag](#)

Examples

```
SendLevel 1
Send btw{Space} ; Produces "by the way ".

; This may be defined in a separate script:
::btw::by the way
```

SendMode [v1.0.43+]

Makes [Send](#) synonymous with [SendInput](#) or [SendPlay](#) rather than the default ([SendEvent](#)). Also makes [Click](#) and [MouseMove/Click/Drag](#) use the specified method.

SendMode [Input](#) | [Play](#) | [Event](#) | [InputThenPlay](#)

The first parameter is one of the following words:

Event: This is the starting default used by all scripts. It uses the [SendEvent](#) method for [Send](#), [SendRaw](#), [Click](#), and [MouseMove/Click/Drag](#).

Input: Switches to the [SendInput](#) method for [Send](#), [SendRaw](#), [Click](#), and [MouseMove/Click/Drag](#). Known limitations:

- Windows Explorer ignores [SendInput](#)'s simulation of certain navigational hotkeys such as `Alt` `[Left]` `[Right]`. To work around this, use either `SendEvent !{Left}` or `SendInput {Backspace}`.

InputThenPlay [v1.0.43.02+]: Same as above except that rather than falling back to [Event](#) mode when [SendInput](#) is [unavailable](#), it reverts to [Play](#) mode (below). This also causes the [SendInput](#) command itself to revert to [Play](#) mode when [SendInput](#) is [unavailable](#).

Play: Switches to the [SendPlay](#) method for [Send](#), [SendRaw](#), [Click](#), and [MouseMove/Click/Drag](#).

Known limitations:

- Characters that do not exist in the current keyboard layout (such as `Ô` in English) cannot be sent. To work around this, use [SendEvent](#).
- Simulated mouse dragging might have no effect in RichEdit controls (and possibly others) such as those of WordPad and Metapad. To use an alternate mode for a particular drag, follow this example: `SendEvent {Click 6, 52, down}{Click 45, 52, up}`.
- Simulated mouse wheel rotation produces movement in only one direction

(usually downward, but upward in some applications). Also, wheel rotation might have no effect in applications such as MS Word and Notepad. To use an alternate mode for a particular rotation, follow this example:

```
SendEvent {WheelDown 5}.
```

- When using `SendMode Play` in the auto-execute section (top part of the script), all remapped keys are affected and might lose some of their functionality. See [SendPlay remapping limitations](#) for details.
- `SendPlay` does not trigger AutoHotkey's hotkeys or hotstrings, or global hotkeys registered by other programs or the OS.

Remarks

Since `SendMode` also changes the mode of [Click](#) and [MouseMove/Click/Drag](#), there may be times when you wish to use a different mode for a particular mouse event. The easiest way to do this is via `{Click}`. For example:

```
SendEvent {Click 100, 200} ; SendEvent uses the  
older, traditional method of clicking.
```

If `SendMode` is used in the auto-execute section (top part of the script), it also affects [keyboard and mouse remapping](#). In particular, if you use `SendMode Play` with remapping, see [SendPlay remapping limitations](#).

[v1.1.23+]: The built-in variable `A_SendMode` contains the current setting.

Every newly launched [thread](#) (such as a [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

[Send](#), [SetKeyDelay](#), [SetMouseDelay](#), [Click](#), [MouseClicked](#), [MouseClickedDrag](#), [MouseMove](#)

Examples

SendMode Input

SendMode InputThenPlay

SetCapsLockState / SetNumLockState / SetScrollLockState

Sets the state of the `CapsLock` `NumLock` `ScrollLock` also force the key to stay on or off.

```
SetCapsLockState [, State]
SetNumLockState [, State]
SetScrollLockState [, State]
```

Parameters

State

If this parameter is omitted, the AlwaysOn/Off attribute of the key is removed (if present). Otherwise, specify one of the following words:

On: Turns on the key and removes the AlwaysOn/Off attribute of the key (if present).

Off: Turns off the key and removes the AlwaysOn/Off attribute of the key (if present).

AlwaysOn: Forces the key to stay on permanently.

AlwaysOff: Forces the key to stay off permanently.

Remarks

A key can also be toggled to its opposite state via the [Send](#) command; for example: `Send {CapsLock}`.

Keeping a key *AlwaysOn* or *AlwaysOff* requires the [keyboard hook](#), which will be automatically installed in such cases.

Related

[SetStoreCapsLockMode](#), [GetKeyState](#)

Examples

```
SetNumLockState, On  
SetScrollLockState, AlwaysOff
```

SetDefaultMouseSpeed

Sets the mouse speed that will be used if unspecified in [Click](#) and [MouseMove/Click/Drag](#).

SetDefaultMouseSpeed, Speed

Parameters

Speed

The speed to move the mouse in the range 0 (fastest) to 100 (slowest). This parameter can be an [expression](#).

Note: A speed of 0 will move the mouse instantly.

Remarks

SetDefaultMouseSpeed is ignored for [SendInput/Play modes](#); they move the mouse instantaneously (however, [SetMouseDelay](#) has a mode that applies to SendPlay). To visually move the mouse more slowly -- such as a script that performs a demonstration for an audience -- use `SendEvent {Click 100, 200}` or `SendMode Event` (optionally in conjunction with [BlockInput](#)).

If this command is not used, the default mouse speed is 2. The built-in variable **A_DefaultMouseSpeed** contains the current setting.

The commands [MouseClick](#), [MouseMove](#), and [MouseClickDrag](#) all have a parameter to override the default mouse speed.

Whenever *Speed* is greater than zero, [SetMouseDelay](#) also influences the speed by producing a delay after each incremental move the mouse makes toward its destination.

Every newly launched [thread](#) (such as a [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

[SetMouseDelay](#), [SendMode](#), [Click](#), [MouseClicked](#), [MouseMove](#), [MouseClickedDrag](#), [SetWinDelay](#), [SetControlDelay](#), [SetKeyDelay](#), [SetKeyDelay](#)

Example

```
SetDefaultMouseSpeed, 0 ; Move the mouse  
instantly.
```

SetKeyDelay

Sets the delay that will occur after each keystroke sent by [Send](#) and [ControlSend](#).

```
SetKeyDelay [, Delay, PressDuration, Play]
```

Parameters

Delay

Time in milliseconds, which can be an [expression](#). Use -1 for no delay at all and 0 for the smallest possible delay (however, if the *Play* parameter is present, both 0 and -1 produce no delay). Leave this parameter blank to retain the current *Delay*.

If `SetKeyDelay` is never used by a script, the default *Delay* for the traditional `SendEvent` mode is 10. For [SendPlay mode](#), the default *Delay* is -1. The default *PressDuration* (below) is -1 for both modes.

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 (however, if the *Play* parameter is present, both 0 and -1 produce no delay). Omit this parameter to leave the current *PressDuration* unchanged.

Note: *PressDuration* also produces a delay after any change to the modifier key state (`Ctrl` , `Alt` , `Shift` , and `Win`) needed to support the keys being sent.

This parameter can be an [expression](#).

Play

[v1.0.43+]

The word *Play* applies the above settings to the [SendPlay mode](#) rather than the traditional [SendEvent](#) mode. If a script never uses this parameter, the delay is always -1/-1 for [SendPlay](#).

Remarks

Note: [SetKeyDelay](#) is not obeyed by [SendInput](#); there is no delay between keystrokes in that mode. This same is true for [Send](#) when [SendMode Input](#) is in effect.

A short delay (sleep) is done automatically 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 or 15. For example, a delay between 1 and 10 (inclusive) is equivalent to 10 or 15 on most Windows XP systems (and probably 2k).

For [Send/SendEvent](#) mode, 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.

When the delay is set to -1, a script's process-priority becomes an important factor in how fast it can send keystrokes when using the traditional [SendEvent mode](#). To raise a script's priority, use `Process, Priority,, High`. Although this typically causes keystrokes to be sent faster than the [active window](#) can process them, the system automatically buffers them. Buffered keystrokes continue to arrive in the target window after the [Send](#) command completes (even if the window is no longer active). This is usually harmless because any subsequent keystrokes sent to the same window get queued up behind the ones already in the buffer.

The built-in variable **A_KeyDelay** contains the current setting of *Delay* for [Send/SendEvent](#) mode. [v1.1.23+]: **A_KeyDuration** contains the setting for *PressDuration*, while **A_KeyDelayPlay** and **A_KeyDurationPlay** contain the

settings for [SendPlay](#).

Every newly launched [thread](#) (such as a [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

[Send](#), [ControlSend](#), [SendMode](#), [SetMouseDelay](#), [SetControlDelay](#), [SetWinDelay](#), [SetBatchLines](#), [Click](#)

Example

```
SetKeyDelay, 0
```

SetMouseDelay

Sets the delay that will occur after each mouse movement or click.

```
SetMouseDelay, Delay [, Play]
```

Parameters

Delay

Time in milliseconds, which can be an [expression](#). Use -1 for no delay at all and 0 for the smallest possible delay (however, if the *Play* parameter is present, both 0 and -1 produce no delay). If unset, the default delay is 10 for the traditional [SendEvent mode](#) and -1 for [SendPlay mode](#).

Play

[v1.0.43+]

The word *Play* applies the delay to the [SendPlay mode](#) rather than the traditional [Send/SendEvent mode](#). If a script never uses this parameter, the delay is always -1 for [SendPlay](#).

Remarks

A short delay (sleep) is done automatically after every mouse movement or click generated by [Click](#) and [MouseMove/Click/Drag](#) (except for [SendInput mode](#)). 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 or 15. For example, a delay between 1 and 10 (inclusive) is equivalent to 10 or 15 on most Windows XP systems (and probably 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.

The built-in variable **A_MouseDelay** contains the current setting for Send/SendEvent mode. [v1.1.23+]: **A_MouseDelayPlay** contains the current setting for [SendPlay mode](#).

Every newly launched [thread](#) (such as a [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

[SetDefaultMouseSpeed](#), [Click](#), [MouseMove](#), [MouseClicked](#), [MouseClickedDrag](#), [SendMode](#), [SetKeyDelay](#), [SetControlDelay](#), [SetWinDelay](#), [SetBatchLines](#)

Example

```
SetMouseDelay, 0
```

SetStoreCapsLockMode

Whether to restore the state of `CapsLock` `Send`

`SetStoreCapsLockMode, On|Off`

Parameters

On|Off

On: This is the initial setting for all scripts: The `CapsLock` `Send` be restored to its former value if `Send` needed to change it temporarily for its operation.

Off: The state of the `CapsLock` `Send` changed at all. As a result, `Send` will invert the case of the characters if `CapsLock` `Send` ON during the operation.

Remarks

This means that the `CapsLock` `Send` 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.

[v1.1.23+]: The built-in variable `A_StoreCapsLockMode` contains the current setting.

Every newly launched `thread` (such as a `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
```

AutoTrim

Determines whether [traditional assignments](#) like `Var1 = %Var2%` omit spaces and tabs from the beginning and end of *Var2*.

AutoTrim, On|Off

Parameters

On|Off

On: In a [traditional assignment](#) like `Var1 = %Var2%`, tabs and spaces at the beginning and end of a *Var2* are omitted from *Var1*. This is the default.

Off: Such tabs and spaces are not omitted.

Any literal tabs and spaces are omitted regardless of this setting. Prior to [v1.1.06], this included ``t` and any escaped literal spaces or tabs. For example, when AutoTrim is Off, the statement `Var = `t` assigns a tab character on [v1.1.06] and an empty string on earlier versions.

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).

The built-in variables [A_Space](#) and [A_Tab](#) contain a single space and single tab character, respectively.

AutoTrim does not affect [expression assignments](#) such as `Var := " string "`. In other words, leading and trailing spaces and tabs are always retained in such cases.

Every newly launched [thread](#) (such as a [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

[SetEnv](#)

Examples

```
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.
```

```
Var1 := "`t" . Var2 . " " ; The setting of
AutoTrim doesn't matter because this is an
expression.
```

Edit

Opens the current script for editing in the associated editor.

Edit

The Edit command opens the current script for editing using the associated "edit" verb in the registry (or Notepad if no verb). However, if an editor window appears to have the script already open (based on its window title), that window is activated instead of opening a new instance of the editor.

This command has no effect when operating from within a compiled script.

On a related note, AutoHotkey syntax highlighting can be enabled for various editors - see below. In addition, context sensitive help for AutoHotkey commands can be enabled in any editor via [this example](#). Finally, your productivity may be improved by using an auto-completion utility like [ISense](#), which works in almost any editor. It watches what you type and displays menus and parameter lists, which does some of the typing for you and reminds you of the order of parameters.

Related

[Reload](#), [How to edit a script](#)

Example

```
Edit ; opens the script for editing.
```

```
; If your editor's command-line usage is something  
like the following,  
; this script can be used to set it as the default  
editor for ahk files:  
;
```

```
; Editor.exe "Full path of script.ahk"
;
; When you run the script, it will prompt you to
select the executable
; file of your editor.
;
FileSelectFile Editor, 2,, Select your editor,
Programs (*.exe)
if ErrorLevel
    ExitApp
RegWrite REG_SZ, HKCR,
AutoHotkeyScript\Shell\Edit\Command,, "%Editor%"
"%1"
```

Editors with AutoHotkey Support

SciTE4AutoHotkey is a custom version of the text editor known as SciTE, tailored for editing AutoHotkey scripts. Its features include:

- Syntax highlighting
- Smart auto-indent
- Auto-complete
- Calltips (also known as IntelliSense)
- Code folding
- Support for [interactive debugging](#)
- Other tools for AutoHotkey scripting

SciTE4AutoHotkey can be found here: <http://fincs.ahk4.net/scite4ahk/>

AHK Studio is a script editor built using AutoHotkey, for editing AutoHotkey scripts. See the following forum thread for details, demonstration videos and an ever-growing list of features: [AHK Studio](#)

AutoGUI is an integrated development environment for AutoHotkey which combines a GUI designer with a script editor. It can be found here: [AutoGUI - GUI Designer and Script Editor](#)

Other editors for which AutoHotkey syntax highlighting can be enabled:

- [AkelPad](#)
- [Crimson Editor](#)
- [Eclipse, FAR manager, and any other editors which use Colorer take5](#)
- [Emacs](#)
- [Notepad++](#)
- [Notepad2](#)
- [PSPad](#)
- [SciTE and possibly other Scintilla based editors](#)
- [Sublime Text Editor](#)
- [Total Commander with Synplus plugin](#)

Additionally, the zip download of AutoHotkey Basic (<http://www.autohotkey.com/download/1.0/>) includes files for enabling syntax highlighting in the following editors. However, some of these files are badly out of date and may or may not work:

- [ConTEXT](#)
- [EditPlus](#)
- [EmEditor](#)
- [jEdit](#)
- [MED](#)
- [TextPad](#)
- [UltraEdit](#)
- [Vim](#)

If your favourite editor isn't listed here, try your luck by searching the [forums](#).

To get an editor added to this page, contact Lexikos [via the forums](#) or [GitHub](#).

Func() [v1.1.00+]

Retrieves a reference to the specified function.

```
FunctionReference := Func(FunctionName)
```

Parameters

FunctionName

The name of the function whose reference is retrieved. *FunctionName* must exist explicitly in the script.

Return Value

This function returns a [reference to *FunctionName*](#). If *FunctionName* does not exist explicitly in the script (by means such as [#Include](#) or a non-dynamic call to a [library function](#)), it returns 0.

Remarks

This function can be used to call the function or retrieve [information](#) such as the minimum and maximum number of parameters.

Related

[Function References](#), [Func Object](#)

Examples

```
; Retrieve a reference to the function named  
"StrLen".  
fn := Func("StrLen")  
  
; Display information about the function.
```

```
MsgBox % fn.Name "()" is " (fn.IsBuiltin ? "built-  
in." : "user-defined.")
```

IsByRef()

 [v1.1.01+]

Returns a non-zero number if a ByRef parameter of a function was supplied with the specified variable.

```
TrueOrFalse := IsByRef(ParameterVar)
```

Parameters

ParameterVar

A reference to the variable. For example: `IsByRef(MyParameter)`.

Return Value

This function returns 1 if *ParameterVar* is a [ByRef parameter](#) and the caller supplied a variable; or 0 if *ParameterVar* is any other kind of variable.

Related

[ByRef parameters](#)

Examples

```
MsgBox, % Function(MyVar)

Function(ByRef Param)
{
    return IsByRef(Param)
}
```

IsFunc() [v1.0.48+]

Returns a non-zero number if the specified function exists in the script.

```
MinParamsPlus1 := IsFunc(FunctionName)
```

Parameters

FunctionName

The name of the function whose minimum number of parameters is retrieved. *FunctionName* must exist explicitly in the script. In [v1.1.00+], *FunctionName* can be a [function reference](#) instead of a name.

Return Value

This function returns one plus the minimum number of parameters (e.g. 1 for a function that requires zero parameters, 2 for a function that requires 1 parameter, etc.). If *FunctionName* does not exist explicitly in the script (by means such as [#Include](#) or a non-dynamic call to a [library function](#)), it returns 0.

Related

[Dynamically Calling a Function](#), [Function References](#), [Func Object](#), [Func\(\)](#), [A_ThisFunc](#)

Examples

```
count := IsFunc("RegExReplace") ; Any function
name can used here.
if count
    MsgBox, % "This function exists and has "
count-1 " mandatory parameters."
else
    MsgBox, % "This function does not exist."
```


IsLabel()

Returns a non-zero number if the specified label exists in the script.

```
TrueOrFalse := IsLabel(LabelName)
```

Parameters

LabelName

The name of a [subroutine](#), [hotkey](#), or [hotstring](#) (do not include the trailing colon(s) in *LabelName*).

Return Value

This function returns a non-zero number if *LabelName* exists in the script.

Remarks

This function is useful to avoid runtime errors when specifying a dynamic label in commands such as [Gosub](#), [Hotkey](#), [Menu](#), and [Gui](#).

Related

[Labels](#)

Examples

```
if IsLabel("Label")
    MsgBox, Subroutine exists
else
    MsgBox, Subroutine doesn't exist

Label:
return
```

```
if IsLabel("^#h")
    MsgBox, Hotkey exists
else
    MsgBox, Hotkey doesn't exist

^#h::return
```

```
if IsLabel("::btw")
    MsgBox, Hotstring exists
else
    MsgBox, Hotstring doesn't exist

::btw::by the way
```

IsObject() [v1.0.90+]

Returns a non-zero number if the specified value is an object.

```
TrueOrFalse := IsObject(ObjectValue)
```

Parameters

ObjectValue

A [object](#) stored in a variable, returned from a function, stored in another object or written directly.

Return Value

This function returns 1 if *ObjectValue* is an object; otherwise 0.

Related

[Objects](#)

Examples

```
object := {key: "value"}

if IsObject(object)
    MsgBox, This is an object.
else
    MsgBox, This is not an object.
```

ListLines

Displays the script lines most recently executed.

```
ListLines [, On|Off]
```

Parameters

On|Off

If blank or omitted, the history of lines most recently executed is shown. [v1.0.48.01+]: An optional first parameter was added, which can be either On or Off. It affects only the behavior of the [current thread](#) as follows:

On: Includes subsequently-executed lines in the history. This is the starting default for all scripts.

Off: Omits subsequently-executed lines from the history.

Remarks

ListLines (with no parameter) is equivalent to selecting the "View->Lines most recently executed" menu item in the main window. It can help [debug a script](#).

`ListLines Off/On` can be used to selectively omit some lines from the history, which can help prevent the history from filling up too quickly (such as in a loop with many fast iterations). `ListLines Off` may also improve performance by a few percent.

Every newly launched [thread](#) (such as a [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).

[v1.1.28+]: The built-in variable `A_ListLines` contains 1 if ListLines is enabled and 0 otherwise.

On a related note, the built-in variables [A_LineNumber](#) and [A_LineFile](#) contain the currently executing line number and the file name to which it belongs.

Related

[KeyHistory](#), [ListHotkeys](#), [ListVars](#)

Example

```
ListLines  
ListLines Off
```

ListVars

Displays the script's [variables](#): their names and current contents.

ListVars

Remarks

This command is equivalent to selecting the "View->Variables" menu item in the main window. It can help you [debug a script](#).

For each variable in the list, the variable's name and contents are shown, along with other information depending on what the variable contains. Each item is terminated with a carriage return and newline (`\r\n`), but may span multiple lines if the variable contains `\r\n`.

List items may take the following forms (where words in *italics* are placeholders):

```
VarName[Length of Capacity]: String  
; [v1.1.26+]  
VarName: TypeName object {Info}  
; Prior to [v1.1.26]:  
VarName[Object]: Address  
VarName[Object]: Address <= ComObject(VarType,  
Value)
```

Capacity is the variable's current [capacity](#) measured in characters, not bytes.

String is the first 60 characters of the variable's contents.

Info depends on the type of object, but is currently very limited.

If this command is used inside a [function](#), the function's [local variables](#) will be listed first (above the script's global variables).

Known limitation: If a [function](#) (or the list of global variables itself) contains more than 10,000 variables, this command might not show them in exact alphabetical order; that is, some might be missing from the display.

Related

[KeyHistory](#), [ListHotkeys](#), [ListLines](#)

The [DebugVars](#) script can be used to inspect and change the contents of variables and objects.

Example

```
ListVars
```

OutputDebug

Sends a string to the debugger (if any) for display.

OutputDebug, Text

Parameters

Text

The text to send to the debugger for display. This text may include linefeed characters (`\n`) to start new lines. In addition, a single long line can be broken up into several shorter ones by means of a [continuation section](#).

Remarks

If the script's process has no debugger, the system debugger displays the string. If the system debugger is not active, this command has no effect.

One example of a debugger is DebugView, which is free and available at www.sysinternals.com.

See also: [other debugging methods](#)

Related

[FileAppend](#), [continuation sections](#)

Example

```
OutputDebug, %A_Now%: Because the window
"%TargetWindowTitle%" did not exist, the process
was aborted.
```

UrlDownloadToFile

Downloads a file from the Internet.

```
UrlDownloadToFile, URL, Filename
```

Parameters

URL

URL of the file to download. For example, <http://someorg.org> might retrieve the welcome page for that organization.

Filename

Download to a file: Specify the name of the file to be created locally, which is assumed to be in [%A_WorkingDir%](#) if an absolute path isn't specified. Any existing file will be **overwritten** by the new file.

Download to a variable: See the [example](#) below.

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[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*.

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.

Caching:

- [v1.0.44.07+]: The URL is retrieved directly from the remote server (that is, never from Internet Explorer's cache). To permit caching, precede the URL with *0 followed by a space; for example: `*0 http://someorg.org`. The zero following the asterisk may be replaced by any valid dwFlags number; for details, search www.microsoft.com for InternetOpenUrl.
- In versions older than 1.0.44.07, the file is retrieved from the cache whenever possible. To avoid this, specify a query string at the end of the URL. For example: `http://www.someorg.org/doc.html?fakeParam=42`. Note: If you download the same file frequently, the query string should be varied.

Proxies: `UrlDownloadToFile` will use a proxy server to access the Internet if such a proxy has been configured in Microsoft Internet Explorer's settings.

FTP and Gopher: [v1.0.48.04+] supports FTP and Gopher URLs. For example:

```
UrlDownloadToFile, ftp://example.com/home/My
File.zip, C:\My Folder\My File.zip ; Log in
anonymously.
UrlDownloadToFile,
ftp://user:pass@example.com:21/home/My File.zip,
C:\My Folder\My File.zip ; Log in as a specific
user.
UrlDownloadToFile,
ftp://user:pass@example.com/My Directory, C:\Dir
Listing.html ; Gets a directory listing in HTML
format.
```

Related

[FileRead](#), [FileCopy](#)

Examples

```
UrlDownloadToFile,  
https://autohotkey.com/download/1.1/version.txt,  
C:\AutoHotkey Latest Version.txt  
UrlDownloadToFile, http://someorg.org/archive.zip,  
C:\SomeOrg's Archive.zip
```

```
; Example: Download text to a variable:  
whr := ComObjCreate("WinHttp.WinHttpRequest.5.1")  
whr.Open("GET",  
"https://autohotkey.com/download/1.1/version.txt",  
true)  
whr.Send()  
; Using 'true' above and the call below allows the  
script to remain responsive.  
whr.WaitForResponse()  
version := whr.ResponseText  
MsgBox % version
```

```
; Example: Make an asynchronous HTTP request.
```

```
req := ComObjCreate("Msxml2.XMLHTTP")  
; Open a request with async enabled.  
req.open("GET",  
"https://autohotkey.com/download/1.1/version.txt",  
true)  
; Set our callback function [requires v1.1.17+].  
req.onreadystatechange := Func("Ready")  
; Send the request. Ready() will be called when  
it's complete.  
req.send()  
/*  
; If you're going to wait, there's no need for  
onreadystatechange.  
; Setting async=true and waiting like this allows  
the script to remain
```

```
; responsive while the download is taking place,
whereas async=false
; will make the script unresponsive.
while req.readyState != 4
    sleep 100
*/
#Persistent

Ready() {
    global req
    if (req.readyState != 4) ; Not done yet.
        return
    if (req.status == 200) ; OK.
        MsgBox % "Latest AutoHotkey version: "
req.responseText
    else
        MsgBox 16,, % "Status " req.status
    ExitApp
}
```

Object

AutoHotkey's basic object datatype is an associative array with features which allow its behaviour to be [customized](#). By default, all objects created by `{}`, `[]`, `Object()` and `Array()` support the following methods:

- [InsertAt / RemoveAt](#)
- [Push / Pop](#)
- [Delete](#)
- [MinIndex / MaxIndex / Length](#)
- [SetCapacity / GetCapacity](#)
- [GetAddress](#)
- [_NewEnum](#)
- [HasKey](#)
- [Clone](#)
- [ObjRawSet](#) (function)

Deprecated (not recommended for use):

- [Insert](#)
- [Remove](#)

Each method also has an equivalent function, which can be used to bypass any [custom behaviour](#) implemented by the object -- it is recommended that these functions only be used for that purpose. To call one, prefix the method name with "Obj" and pass the target object as the first parameter. For example:

```
array := [1, 2, 3]
MsgBox % ObjMaxIndex(array) " = "
array.MaxIndex()
```

If an Obj function is called with an object or value of the wrong type, it returns an empty string.

InsertAt [v1.1.21+]

Inserts one or more values at a given position within a linear array.

```
Object.InsertAt(Pos, Value1 [, Value2, ... ValueN])
```

Pos

The position to insert *Value1* at. Subsequent values are inserted at Pos+1, Pos+2, etc.

Value1 ...

One or more values to insert. To insert an array of values, pass `theArray*` as the last parameter.

Remarks

InsertAt is the counterpart of [RemoveAt](#).

As Objects are associative arrays, *Pos* is also the integer key which will be associated with *Value1*. Any items previously at or to the right of *Pos* are shifted to the right by the exact number of value parameters, even if some values are missing (i.e. the object is a sparse array). For example:

```
x := []
x.InsertAt(1, "A", "B") ; => ["A", "B"]
x.InsertAt(2, "C")      ; => ["A", "C", "B"]

; Sparse/unassigned elements are preserved:
x := ["A", , "C"]
x.InsertAt(2, "B")      ; => ["A", "B", , "C"]

x := ["C"]
x.InsertAt(1, , "B")    ; => [ , "B", "C"]
```

InsertAt should be used only when the object's integer keys represent positions in a linear array. If the object contains arbitrary integer keys such as IDs or handles, InsertAt is likely to cause unwanted side-effects. For example:

```
x := [], handleX := 0x4321, handleY := 0x1234
x.InsertAt(handleX, "A")
MsgBox % x[handleX] ; A - okay
x.InsertAt(handleY, "B")
MsgBox % x[handleX] ; Empty
MsgBox % x[handleX+1] ; This is the new
"position" of "A"
```

InsertAt does not affect string or object keys, so can be safely used with objects containing mixed key types.

RemoveAt [v1.1.21+]

Removes items from the given position in a linear array.

```
Object.RemoveAt(Pos [, Length])
```

Pos

The position of the value or values to remove.

Length

The length of the range of values to remove. Items from `Pos` to `Pos+Length-1` are removed. If omitted, one item is removed.

Return Value

If *Length* is omitted, the value removed from *Pos* is returned (blank if none). Otherwise the return value is the number of removed items which had values, which can differ from *Length* in a sparse array, but is always between 0 and *Length* (inclusive).

Remarks

RemoveAt is the counterpart of [InsertAt](#).

The remaining items to the right of *Pos* are shifted to the left by *Length* (or 1 if omitted), even if some items in the removed range did not have values. For example:

```
x := ["A", "B"]
MsgBox % x.RemoveAt(1) ; A
MsgBox % x[1] ; B

x := ["A", , "C"]
MsgBox % x.RemoveAt(1, 2) ; 1
MsgBox % x[1] ; C
```

RemoveAt should be used only when the object's integer keys represent positions in a linear array. If the object contains arbitrary integer keys such as IDs or handles, RemoveAt is likely to cause unwanted side-effects. For example:

```
x := {0x4321: "A", 0x1234: "B"}  
MsgBox % x.RemoveAt(0x1234) ; B  
MsgBox % x[0x4321] ; Empty  
MsgBox % x[0x4321-1] ; A
```

RemoveAt does not affect string or object keys, so can be safely used with objects containing mixed key types.

Push [v1.1.21+]

Appends values to the end of an array.

```
Object.Push([ Value, Value2, ..., ValueN ])
```

Value ...

One or more values to insert. To insert an array of values, pass `theArray*` as the last parameter.

Return Value

The position of the last inserted value. Can be negative if the array only contained elements at negative indices.

Remarks

The first value is inserted at position 1 if the array is empty or contains only string or object keys.

Otherwise, the first value is inserted at `Object.MaxIndex() + 1`, even if that position is negative or zero. If this is undesired and the object can contain negative keys, `Object.InsertAt(Object.Length() + 1, ...)` can be used instead.

Pop [v1.1.21+]

Removes and returns the last array element.

```
Value := Object.Pop()
```

If there are no array elements, the return value is an empty string. Otherwise, it is equivalent to the following:

```
Value := Object.RemoveAt(Object.Length())
```

Delete [v1.1.21+]

Removes key-value pairs from an object.

```
Object.Delete(Key)
Object.Delete(FirstKey, LastKey)
```

Key

Any single key.

FirstKey, LastKey

Any valid range of integer or string keys, where *FirstKey* <= *LastKey*. Both keys must be the same type.

Return Value

If there is exactly one parameter, the removed value is returned (blank if none). Otherwise the return value is the number of matching keys which were found and removed.

Remarks

Unlike [RemoveAt](#), Delete does not affect any of the key-value pairs that it does not remove. For example:

```
x := ["A", "B"]
MsgBox % x.RemoveAt(1) ; A
MsgBox % x[1] ; B

x := ["A", "B"]
MsgBox % x.Delete(1) ; A
MsgBox % x[1] ; Empty
```

MinIndex / MaxIndex [AHK_L 31+]

```
MinIndex := Object.MinIndex()
```

```
MaxIndex := Object.MaxIndex()
```

If any integer keys are present, MinIndex returns the lowest and MaxIndex returns the highest. Otherwise an empty string is returned.

Length [v1.1.21+]

```
Length := Object.Length()
```

Returns the length of a linear array beginning at position 1; that is, the highest positive integer key contained by the object, or 0 if there aren't any.

```
MsgBox % ["A", "B", "C"].Length() ; 3  
MsgBox % ["A", , "C"].Length() ; 3  
MsgBox % {-10: 0, 10: 0}.Length() ; 10  
MsgBox % {-10: 0, -1: 0}.Length() ; 0
```

SetCapacity [AHK_L 31+]

Adjusts the capacity of an object or one of its fields.

```
Object.SetCapacity(MaxItems)
```

```
Object.SetCapacity(Key, ByteSize)
```

MaxItems The maximum number of key-value pairs the object should be able to contain before it must be automatically expanded. If less than the current number of key-value pairs, that number is used instead, and any unused space is freed.

Key Any valid key.

ByteSize The new size in bytes of the field's string buffer, excluding the null-terminator. If the field does not exist, it is created. If *ByteSize* is zero, the buffer is freed but the empty field is not removed. If *ByteSize* is less than the current size, excess data is truncated; otherwise all existing data is preserved.

Returns The new capacity if successful, otherwise an empty string.

GetCapacity [AHK_L 31+]

```
MaxItems := Object.GetCapacity()  
ByteSize := Object.GetCapacity(Key)
```

Returns the current capacity of an object or one of its fields.

Returns an empty string if the field does not exist or does not contain a string.

GetAddress [AHK_L 31+]

```
Ptr := Object.GetAddress(Key)
```

Returns the current address of the field's string buffer, if it has one.

NewEnum [AHK_L 49+]

```
Enum := Object._NewEnum()
```

Returns a new [enumerator](#) to enumerate this object's key-value pairs. This method is usually not called directly, but by the [for-loop](#).

HasKey [AHK_L 53+]

`Object.HasKey(Key)`

Returns true if *Key* is associated with a value (even "") within *Object*, otherwise false.

Clone [AHK_L 60+]

```
Clone := Object.Clone()
```

Returns a shallow copy of the object.

ObjRawSet

Stores or overwrites a key-value pair in the object.

```
ObjRawSet(Object, Key, Value)
```

This function is provided to allow scripts to bypass the `__Set` [meta-function](#). If that isn't required, a normal assignment should be used instead. For example: `Object[Key] := Value`

Since the purpose is to bypass meta-functions, this is a function only, not a method. Calling a built-in method generally causes the `__Call` meta-function to be called.

Insert [AHK_L 31+]

Deprecated: Insert is not recommended for use in new scripts. Use [InsertAt](#), [Push](#), [ObjRawSet](#) or a simple assignment instead.

Inserts key-value pairs into the object, automatically adjusting existing keys if given an integer key.

```
Object.Insert(Pos, Value1 [, Value2, ... ValueN ])  
Object.Insert(Value)  
Object.Insert(StringOrObjectKey, Value)
```

The behaviour of Insert depends on the number and type of its parameters:

- If there are multiple parameters and the first parameter is an integer, Insert behaves like [InsertAt](#).
- If there are multiple parameters and the first parameter is not an integer, Insert behaves like [ObjRawSet](#).
- If there is only one parameter, Insert behaves like [Push](#).

Insert returns *true*. In [v1.1.21] and later, an exception is thrown if a memory allocation fails. Earlier versions returned an empty string in that case.

Remove [AHK_L 31+]

Deprecated: Remove is not recommended for use in new scripts. Use [RemoveAt](#), [Delete](#) or [Pop](#) instead.

Removes key-value pairs from an object.

```
Object.Remove(FirstKey, LastKey)
```

The behaviour of Remove depends on the number and type of parameters:

- `Object.Remove(Integer)` behaves like `Object.RemoveAt(Integer)`.
- `Object.Remove(Integer, "")` behaves like `Object.Delete(Integer)`.
- `Object.Remove(Integer1, Integer2)` behaves like `Object.RemoveAt(Integer1, Integer2 - Integer1 + 1)`.
- `Object.Remove()` behaves like `Object.Pop()`.
- Any other valid combination of parameters behaves like [Delete](#).

Enumerator Object [AHK_L 49+]

Allows items in a collection to be enumerated.

Next

Retrieves the next item or items in an enumeration.

```
Enum.Next(OutputVar1 [, OutputVar2, ...])
```

OutputVar1, Receives an implementation-specific value.

OutputVar2

... Additional parameters, if supported.

Returns A non-zero integer if successful or zero if there were no items remaining.

Object

Enumerators returned by [ObjNewEnum\(\)](#) are called once for each key-value pair, and allow up to two parameters:

OutputVar1 Receives the **key** in a key-value pair.

OutputVar2 Receives the **value** associated with *OutputVar1*.

Key-value pairs are returned in an implementation-defined order. That is, they are typically not returned in the same order that they were assigned. Existing key-value pairs may be modified during enumeration, but inserting or removing keys may cause some items to be enumerated multiple times or not at all.

Related

[For-loop](#), [Object.NewEnum\(\)](#)

Example

```
; Create some sample data.
```

```
obj := Object("red", 0xFF0000, "blue", 0x0000FF,  
"green", 0x00FF00)
```

```
; Enumerate!
```

```
enum := obj._NewEnum()  
While enum[k, v]  
    t .= k "=" v "`n"  
MsgBox % t
```

```
; Requires [AHK_L 59+]
```

```
For k, v in obj  
    s .= k "=" v "`n"  
MsgBox % s
```

File Object [AHK_L 42+]

Provides an interface for file input/output. [FileOpen](#) returns an object of this type.

Object Members

Properties:

- **Pos**: Retrieves or sets the position of the file pointer. Equivalent to `POS := File.Tell()` or `File.Seek(Distance)`.
- **Length**: Retrieves or sets the size of the file.
- **AtEOF**: Retrieves a non-zero value if the file pointer has reached the end of the file.
- **Encoding**: Retrieves or sets the text encoding used by this file object.
- **__Handle**: Retrieves a system file handle, intended for use with `DllCall`.

Methods:

- **Read**: Reads a string of characters from the file and advances the file pointer.
- **Write**: Writes a string of characters to the file and advances the file pointer.
- **ReadLine**: Reads a line of text from the file and advances the file pointer.
- **WriteLine**: Writes a string of characters followed by ``n` or ``r`n` depending on the flags used to open the file. Advances the file pointer.
- **ReadNumType**: Reads a number from the file and advances the file pointer.
- **WriteNumType**: Writes a number to the file and advances the file pointer.
- **RawRead**: Read raw binary data from the file into memory. If a var is specified, it is expanded automatically when necessary.
- **RawWrite**: Write raw binary data to the file.
- **Seek**: Moves the file pointer. If the second parameter is omitted, it is equivalent to `File.Pos := Distance`.
- **Tell**: Returns the position of the file pointer. Equivalent to `POS := File.Pos`.
- **Close**: Closes the file, flushes any data in the cache to disk and releases the share locks.

Read

Reads a string of characters from the file and advances the file pointer.

```
String := File.Read([Characters])
```

Characters

The maximum number of characters to read. If omitted, the rest of the file is read and returned as one string. If the File object was created from a handle to a non-seeking device such as a console buffer or pipe, omitting this parameter may cause the method to fail or return only what data is currently available.

Returns a string.

Write

Writes a string of characters to the file and advances the file pointer.

```
File.Write(String)
```

String

A string to write.

Returns the number of bytes (not characters) that were written.

ReadLine

Reads a line of text from the file and advances the file pointer.

```
TextLine := File.ReadLine()
```

Returns a line of text. This may include ``n`, ``r`n` or ``r` depending on the file and EOL flags used to open the file.

Lines up to 65,534 characters long can be read. If the length of a line exceeds this, the remainder of the line is returned by subsequent calls to this method.

WriteLine

Writes a string of characters followed by `\n` or `\r\n` depending on the flags used to open the file. Advances the file pointer.

```
File.WriteLine([String])
```

String

An optional string to write.

Returns the number of bytes (not characters) that were written.

ReadNumType

Reads a number from the file and advances the file pointer.

```
Num := File.ReadNumType()
```

NumType is either UInt, Int, Int64, Short, UShort, Char, UChar, Double, or Float. These type names have the same meanings as with [DllCall](#).

Returns a number if successful, otherwise an empty string.

If a Try statement is active and no bytes were read, an exception is thrown. However, no exception is thrown if at least one byte was read, even if the size of the given *NumType* is greater than the number of bytes read. Instead, the missing bytes are assumed to be zero.

WriteNumType

Writes a number to the file and advances the file pointer.

```
File.WriteNumType(Num)
```

Num

A number to write.

NumType is either UInt, Int, Int64, Short, UShort, Char, UChar, Double, or Float. These type names have the same meanings as with [DllCall](#).

Returns the number of bytes that were written. For instance, WriteUInt returns 4 if successful.

RawRead

Read raw binary data from the file into memory. If a var is specified, it is expanded automatically when necessary.

```
File.RawRead(VarOrAddress, Bytes)
```

VarOrAddress

A variable or memory address to which the data will be copied. Usage is similar to [NumGet](#).

Bytes

The maximum number of bytes to read.

Returns the number of bytes that were read.

If a Try statement is active and *Bytes* is non-zero but no bytes were read, an exception is thrown. [AtEOF](#) can be used to avoid this, if needed.

RawWrite

Write raw binary data to the file.

```
File.RawWrite(VarOrAddress, Bytes)
```

VarOrAddress

A variable containing the data or the address of the data in memory.
Usage is similar to [NumPut](#).

Bytes

The number of bytes to write.

Returns the number of bytes that were written.

Seek

Moves the file pointer.

```
File.Seek(Distance [, Origin := 0])  
File.Position := Distance  
File.Pos := Distance
```

Distance

Distance to move, in bytes. Lower values are closer to the beginning of the file.

Origin

Starting point for the file pointer move. Must be one of the following:

- 0 (SEEK_SET): Beginning of the file. *Distance* must be zero or greater.
- 1 (SEEK_CUR): Current position of the file pointer.
- 2 (SEEK_END): End of the file. *Distance* should usually be negative.

If omitted, *Origin* defaults to SEEK_END when *Distance* is negative and SEEK_SET otherwise.

Returns a non-zero value if successful, otherwise zero.

Tell

Returns the current position of the file pointer, where 0 is the beginning of the file.

```
Pos := File.Tell()  
Pos := File.Position  
Pos := File.Pos
```

Length

Retrieves or sets the size of the file.

```
FileSize := File.Length
```

```
File.Length := NewSize
```

FileSize and *NewSize* is the size of the file, in bytes.

This property should be used only with an actual file. If the File object was created from a handle to a pipe, it may return the amount of data currently available in the pipe's internal buffer, but this behaviour is not guaranteed.

AtEOF

Retrieves a non-zero value if the file pointer has reached the end of the file, otherwise zero.

```
IsAtEOF := File.AtEOF
```

This property should be used only with an actual file. If the File object was created from a handle to a non-seeking device such as a console buffer or pipe, the returned value may not be meaningful, as such devices do not logically have an "end of file".

Close

Closes the file, flushes any data in the cache to disk and releases the share locks.

```
File.Close()
```

Although the file is closed automatically when the object is freed, it is recommended to close the file as soon as possible.

Encoding

Retrieves or sets the text encoding used by this file object.

```
RetrievedEncoding := File.Encoding
```

```
File.Encoding := NewEncoding
```

RetrievedEncoding and *NewEncoding* is a numeric code page identifier (see [MSDN](#)) or one of the following strings:

- **UTF-8** : Unicode UTF-8, equivalent to CP65001.
- **UTF-16** : Unicode UTF-16 with little endian byte order, equivalent to CP1200.
- **CPnnn** : a code page with numeric identifier *nnn*.

RetrievedEncoding is never a value with the **-RAW** suffix, regardless of how the file was opened or whether it contains a byte order mark (BOM). Setting *NewEncoding* never causes a BOM to be added or removed, as the BOM is normally written to the file when it is first created.

[v1.1.15.04+]: Setting *NewEncoding* to **UTF-8-RAW** or **UTF-16-RAW** is valid, but the **-RAW** suffix is ignored. In earlier versions, **UTF-8-RAW** and **UTF-16-RAW** behaved like an invalid 8-bit encoding, causing all non-ASCII characters to be discarded. This only applies to **File.Encoding**, not [FileOpen\(\)](#).

__Handle

Returns a system file handle, intended for use with DllCall. See [CreateFile](#).

File.__Handle

File objects internally buffer reads or writes. If data has been written into the object's internal buffer, it is committed to disk before the handle is returned. If the buffer contains data read from file, it is discarded and the actual file pointer is reset to the logical position indicated by **File.Pos**.

Func Object [v1.1.00+]

Represents a user-defined or built-in function which can be called by the script. `Func` returns an object of this type.

For information about other objects which can be called like functions, see [Function Objects](#).

A reference to a Func object is also known as a *function reference*. To retrieve a function reference, use the Func function as in the following example:

```
; Retrieve a reference to the function named
"StrLen".
fn := Func("StrLen")

; Display information about the function.
MsgBox % fn.Name "()" is " (fn.IsBUILTIn ?
"built-in." : "user-defined.")
```

Object Members

Properties:

- **Name**: Returns the function's name.
- **IsBuiltIn**: Returns *true* if the function is built-in and *false* otherwise.
- **IsVariadic**: Returns *true* if the function is **variadic** and *false* otherwise.
- **MinParams**: Returns the number of required parameters.
- **MaxParams**: Returns the number of formally-declared parameters for a user-defined function or maximum parameters for a built-in function.

Methods:

- **Call**: Calls the function.
- **Bind**: Binds parameters to the function and returns a **BoundFunc object**.
- **IsByRef**: Determines whether a parameter is ByRef.
- **IsOptional**: Determines whether a parameter is optional.

Call

Calls the function.

```
Func.Call(Param1, Param2, ...) ; Requires  
[v1.1.19+]  
Func.(Param1, Param2, ...) ; Old form - deprecated
```

Param1, Param2, ...

Parameters and return value are defined by the function.

[v1.1.07+]: `%Func%()` can be used to call a function by name or reference, or to call an object which implements the `__Call` [meta-function](#). This should be used instead of `Func.()` wherever possible.

Bind [v1.1.20+]

Binds parameters to the function and returns a [BoundFunc object](#).

```
BoundFunc := Func.Bind(Param1, Param2, ...)
```

Param1, Param2, ...

Any number of parameters.

For details and examples, see [BoundFunc object](#).

Name

Returns the function's name.

```
FunctionName := Func.Name
```

IsBUILTIn

Returns *true* if the function is built-in and *false* otherwise.

```
Boolean := Func.IsBUILTIn
```

IsVariadic

Returns *true* if the function is [variadic](#) and *false* otherwise.

```
Boolean := Func.IsVariadic
```

MinParams

Returns the number of required parameters.

```
ParamCount := Func.MinParams
```

MaxParams

Returns the number of formally-declared parameters for a user-defined function or maximum parameters for a built-in function.

```
ParamCount := Func.MaxParams
```

If the function is [variadic](#), *ParamCount* indicates the maximum number of parameters which can be accepted by the function without overflowing into the "variadic*" parameter.

IsByRef

Determines whether a parameter is ByRef.

```
Boolean := Func.IsByRef([ParamIndex])
```

ParamIndex

Optional: the one-based index of a parameter. If omitted, *Boolean* indicates whether the function has any ByRef parameters.

Returns an empty string if the function is built-in or *ParamIndex* is invalid; otherwise, a boolean value indicating whether the parameter is ByRef.

IsOptional

Determines whether a parameter is optional.

```
Boolean := Func.IsOptional([ParamIndex])
```

ParamIndex

Optional: the one-based index of a parameter. If omitted, *Boolean* indicates whether the function has any optional parameters.

Returns an empty string if *ParamIndex* is invalid; otherwise, a boolean value indicating whether the parameter is optional.

Parameters do not need to be formally declared if the function is variadic. Built-in functions are supported.

Function Objects

"Function object" usually means any of the following:

- A reference to a [Func object](#), which represents an actual [function](#); either built-in or defined by the script.
- A user-defined object which can be called like a function. This is sometimes also referred to as a "functor".
- Any other object which can be called like a function, such as a [BoundFunc object](#) or a JavaScript function object returned by a COM method.

Function objects can be used with the following:

- [Gui control events](#) (g-labels)
- [Hotkey](#)
- [Menu](#)
- [OnClipboardChange\(\)](#)
- [OnExit\(\)](#)
- [OnMessage\(\)](#)
- [SetTimer](#)

User-Defined

User-defined function objects should follow this general pattern:

```
class YourClassName {
    Call(a, b) { ; Declare parameters as
needed, or an array*.
        ;...
    }
    __Call(method, args*) {
        if (method = "") ; For %fn%() or fn.()
            return this.Call(args*)
        if (IsObject(method)) ; If this
function object is being used as a method.
            return this.Call(method, args*)
        }
        ;...
    }
}
```

Exactly which parts are needed depends on the usage:

- `method` is an empty string if the script used `%this%()` or `this.()`.
- If the object is being used as a method, `IsObject(method)` is true and `method` contains a reference to the target object. For example, if `x.y` refers to `this` function object, `x.y()` → `this[x]()` → `this.__Call(x)` → `this.Call(x)`.
- [v1.1.20+]: If the object is being used by one of the built-in functions which accept a callback function, such as `OnMessage` or `SetTimer`, only the `Call` method is needed.

The work can also be done directly in `__Call`. However, having `__Call` redirect to `Call` is recommended to ease the transition to [AutoHotkey v2](#), which will change the behaviour of `%this%()` and method calls to call the `Call` method directly.

Examples

If you are defining multiple function object types, boilerplate code should be delegated to a base class (but if you'll ever combine your code with someone else's, be wary of conflicts). For example:

```
class FunctionObject {
  __Call(method, args*) {
    if (method = "")
      return this.Call(args*)
    if (IsObject(method))
      return this.Call(method, args*)
  }
}
```

The following example defines a function array which can be called; when called, it calls each element of the array in turn.

```
class FuncArrayType extends FunctionObject {
  Call(obj, params*) {
    ; Call a list of functions.
    Loop % this.Length()
      this[A_Index].Call(params*)
  }
}

; Create an array of functions.
funcArray := new FuncArrayType
; Add some functions to the array (can be done
at any point).
funcArray.Push(Func("One"))
funcArray.Push(Func("Two"))
; Create an object which uses the array as a
method.
obj := {method: funcArray}
; Call the method.
```

```
obj.method("foo", "bar")
```

```
One(param1, param2) {
```

```
    ListVars
```

```
    MsgBox
```

```
}
```

```
Two(param1, param2) {
```

```
    ListVars
```

```
    MsgBox
```

```
}
```

BoundFunc Object [v1.1.20+]

Acts like a function, but just passes predefined parameters to another function.

There are two ways that BoundFunc objects can be created:

- By calling the [Func.Bind\(\)](#) method, which binds parameter values to a function.
- By calling the [ObjBindMethod\(\)](#) function, which binds parameter values and a method name to a target object.

BoundFunc objects can be called as shown in the example below. No other methods are supported. When the BoundFunc is called, it calls the function or method to which it is bound, passing any bound parameters followed by any which were passed by the caller. For example:

```
fn := Func("RealFn").Bind(1)

%fn%(2)      ; Shows "1, 2"
fn.Call(3)   ; Shows "1, 3"

RealFn(a, b) {
    MsgBox %a%, %b%
}
```

[ObjBindMethod\(\)](#) can be used to bind to a method when it isn't possible to retrieve a reference to the method itself. For example:

```
file := FileOpen(A_ScriptFullPath, "r")
getLine := ObjBindMethod(file, "ReadLine")
MsgBox % %getLine%() ; Shows the first line of
this file.
```

For a more complex example, see [SetTimer](#).

Process

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

Cmd

One of the following words:

Exist: Sets [ErrorLevel](#) to the Process ID (PID) if a matching process exists, or 0 otherwise. If the *PID-or-Name* parameter is blank, the script's own PID is retrieved. An alternate, single-line method to retrieve the script's PID is `PID := DllCall("GetCurrentProcessId")`.

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.

List: Although *List* is not yet supported, the [examples section](#) demonstrates how to retrieve a list of processes via DllCall.

Priority: Changes the priority (as seen in Windows Task Manager) of the first matching process to *Param3* and sets [ErrorLevel](#) to its Process ID (PID). If the *PID-or-Name* 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.

Param3 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). Note: Any process not designed to run at Realtime priority might reduce system stability if set to that level.

Wait: Waits up to *Param3* seconds (can contain a decimal point) for a matching process to exist. If *Param3* 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.

WaitClose: Waits up to *Param3* seconds (can contain a decimal point) for ALL matching processes to close. If *Param3* 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.

PID-or-Name

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.

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.

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 stops waiting. In other words, rather than waiting for the timeout to expire, it immediately sets [ErrorLevel](#) as described above, then continues 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](#).

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 ; 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:
```

```
#z:: ; 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`npriority level for the
following window:`n%active_title%
Gui, 5:Add, ListBox, vMyListBox gMyListBox r5,
Normal|High|Low|BelowNormal|AboveNormal
Gui, 5:Add, Button, default, OK
Gui, 5:Show,, Set Priority
return

5GuiEscape:
5GuiClose:
Gui, Destroy
return

MyListBox:
if A_GuiEvent <> DoubleClick
    return
; else fall through to the next label:
5ButtonOK:
```

```

GuiControlGet, MyListBox
Gui, Destroy
Process, Priority, %active_pid%, %MyListBox%
if ErrorLevel
    MsgBox Success: Its priority was changed to
"%MyListBox%".
else
    MsgBox Error: Its priority could not be
changed to "%MyListBox%".
return

```

; Example #4: Retrieves a list of running processes via DllCall then shows them in a MsgBox.

```

d := " | " ; string separator
s := 4096 ; size of buffers and arrays (4 KB)

Process, Exist ; sets ErrorLevel to the PID of
this running script
; Get the handle of this script with
PROCESS_QUERY_INFORMATION (0x0400)
h := DllCall("OpenProcess", "UInt", 0x0400, "Int",
false, "UInt", ErrorLevel, "Ptr")
; Open an adjustable access token with this
process (TOKEN_ADJUST_PRIVILEGES = 32)
DllCall("Advapi32.dll\OpenProcessToken", "Ptr", h,
"UInt", 32, "PtrP", t)
VarSetCapacity(ti, 16, 0) ; structure of
privileges
NumPut(1, ti, 0, "UInt") ; one entry in the
privileges array...
; Retrieves the locally unique identifier of the
debug privilege:
DllCall("Advapi32.dll\LookupPrivilegeValue",

```

```

"Ptr", 0, "Str", "SeDebugPrivilege", "Int64P",
luid)
NumPut(luid, ti, 4, "Int64")
NumPut(2, ti, 12, "UInt") ; enable this
privilege: SE_PRIVILEGE_ENABLED = 2
; Update the privileges of this process with the
new access token:
r := DllCall("Advapi32.dll\AdjustTokenPrivileges",
"Ptr", t, "Int", false, "Ptr", &ti, "UInt", 0,
"Ptr", 0, "Ptr", 0)
DllCall("CloseHandle", "Ptr", t) ; close this
access token handle to save memory
DllCall("CloseHandle", "Ptr", h) ; close this
process handle to save memory

hModule := DllCall("LoadLibrary", "Str",
"Psapi.dll") ; increase performance by preloading
the library
s := VarSetCapacity(a, s) ; an array that
receives the list of process identifiers:
c := 0 ; counter for process identifiers
DllCall("Psapi.dll\EnumProcesses", "Ptr", &a,
"UInt", s, "UIntP", r)
Loop, % r // 4 ; parse array for identifiers as
DWORDs (32 bits):
{
    id := NumGet(a, A_Index * 4, "UInt")
    ; Open process with: PROCESS_VM_READ (0x0010) |
PROCESS_QUERY_INFORMATION (0x0400)
    h := DllCall("OpenProcess", "UInt", 0x0010 |
0x0400, "Int", false, "UInt", id, "Ptr")
    if !h
        continue
    VarSetCapacity(n, s, 0) ; a buffer that
receives the base name of the module:
    e := DllCall("Psapi.dll\GetModuleBaseName",
"Ptr", h, "Ptr", 0, "Str", n, "UInt", A_IsUnicode

```

```

? s//2 : s)
    if !e ; fall-back method for 64-bit
processes when in 32-bit mode:
    if e :=
DllCall("Psapi.dll\GetProcessImageFileName",
"Ptr", h, "Str", n, "UInt", A_IsUnicode ? s//2 :
s)
        SplitPath n, n
    DllCall("CloseHandle", "Ptr", h) ; close
process handle to save memory
    if (n && e) ; if image is not null add to
list:
        l .= n . d, c++
}
DllCall("FreeLibrary", "Ptr", hModule) ; unload
the library to free memory
;Sort, l, C ; uncomment this line to sort the
list alphabetically
MsgBox, 0, %% Processes, %l%

```

```

; Example #5: Retrieves a list of running
processes via COM.

```

```

Gui, Add, ListView, x2 y0 w400 h500, Process
Name|Command Line
for process in
ComObjGet("winmgmts:").ExecQuery("Select * from
Win32_Process")
    LV_Add("", process.Name, process.CommandLine)
Gui, Show,, Process List

```

```

; Win32_Process: http://msdn.microsoft.com/en-us/library/aa394372.aspx

```

Run[Wait]

Runs an external program. Unlike Run, RunWait will wait until the program finishes before continuing.

```
Run, Target [, WorkingDir,  
Max|Min|Hide|UseErrorLevel, OutputVarPID]  
RunWait, Target [, WorkingDir,  
Max|Min|Hide|UseErrorLevel, OutputVarPID]
```

Parameters

Target

A document, URL, executable file (.exe, .com, .bat, etc.), shortcut (.lnk), or [system verb](#) to launch (see remarks). If *Target* 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, *Target* 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)

Note: Some applications (e.g. Calc.exe) do not obey the requested startup state and thus Max/Min/Hide will have no effect.

UseErrorLevel: UseErrorLevel can be specified alone or in addition to one of the above words (by separating it from the other word with a space). If the launch fails, this option skips the warning dialog, sets [ErrorLevel](#) to the word ERROR, and allows the [current thread](#) to continue. If the launch succeeds, RunWait sets [ErrorLevel](#) to the program's exit code, and Run sets it to 0.

When UseErrorLevel is specified, the variable **A_LastError** is set to the result of the operating system's GetLastError() function. A_LastError is a number between 0 and 4294967295 (always formatted as decimal, not hexadecimal). Zero (0) means success, but any other number means the launch failed. Each number corresponds to a specific error condition (to get a list, search www.microsoft.com for "system error codes"). Like [ErrorLevel](#), A_LastError is a per-thread setting; that is, interruptions by other [threads](#) cannot change it. However, A_LastError is also set by [DllCall](#).

OutputVarPID

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. RunWait also supports this parameter, though its *OutputVarPID* must be checked in [another thread](#) (otherwise, the PID will be invalid because the process will have terminated by the time the line following RunWait executes).

After the Run command retrieves a PID, any windows to be created by

the process might not exist yet. To wait for at least one window to be created, use `WinWait ahk_pid %OutputVarPID%`.

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

Run: Does not set [ErrorLevel](#) unless UseErrorLevel (above) is in effect, in which case ErrorLevel is set to the word ERROR upon failure or 0 upon success.

RunWait: Sets ErrorLevel to the program's exit code (a signed 32-bit integer). If UseErrorLevel is in effect and the launch failed, the word ERROR is stored.

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 integer). Some programs will appear to return immediately even though they are still running; these programs spawn another process.

If *Target* contains any commas, they must be [escaped](#) as shown three times in the following example:

```
Run rundll32.exe shell32.dll``,`Control_RunDLL
desk.cpl``,``,`3 ; Opens Control Panel > Display
Properties > Settings
```

When running a program via [ComSpec](#) (cmd.exe) -- perhaps because you need to redirect the program's input or output -- if the path or name of the executable contains spaces, the entire string should be enclosed in an outer pair of quotes. In the following example, the outer quotes are shown in red and all the inner quotes are shown in black:

```
Run %ComSpec% /c ""C:\My Utility.exe" "param 1"
"second param" >"C:\My File.txt""
```

If *Target* cannot be launched, an error window is displayed and the current thread is exited, unless the string **UseErrorLevel** is included in the third parameter or the error is caught by a [Try/Catch](#) statement.

Performance may be slightly improved if *Target* is an exact path, e.g. `Run, C:\Windows\notepad.exe "C:\My Documents\Test.txt"` rather than `Run, C:\My Documents\Test.txt`.

Special [CLSID folders](#) may be opened via Run. For example:

```
Run ::{20d04fe0-3aea-1069-a2d8-08002b30309d} ;  
Opens the "My Computer" folder.  
Run ::{645ff040-5081-101b-9f08-00aa002f954e} ;  
Opens the Recycle Bin.
```

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:

<i>*verb</i>	[AHK_L 57+]: Any system-defined or custom verb. For example: <code>Run *Compile %A_ScriptFullPath%</code> On Windows Vista and later, the *RunAs verb may be used in place of the <i>Run as administrator</i> right-click menu item.
properties	Displays the Explorer's properties window for the indicated file. For example: <code>Run, properties "C:\My File.txt"</code> Note: The properties window will automatically close when the script terminates. To prevent this, use WinWait to wait for the window to appear, then use WinWaitClose to wait for the user to close it.
find	Opens an instance of the Explorer's Search Companion or Find File window at the indicated folder. For example: <code>Run, find D:\</code>

explore	Opens an instance of Explorer at the indicated folder. For example: <code>Run, explore %A_ProgramFiles%</code> .
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. For example: <code>Run, edit "C:\My File.txt"</code>
open	Opens the indicated file (normally not needed because it is the default action for most file types). For example: <code>Run, open "My File.txt"</code> .
print	Prints the indicated file with the associated application, if any. For example: <code>Run, print "My File.txt"</code>

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

Run as Administrator [AHK_L 57+]:

For an executable file, the **RunAs* verb is equivalent to selecting *Run as administrator* from the right-click menu of the file. For example, the following code attempts to restart the current script as admin:

```
full_command_line := DllCall("GetCommandLine",
"str")

if not (A_IsAdmin or
RegexMatch(full_command_line, "
/restart(?:\S)"))
{
    try
    {
        if A_IsCompiled
            Run *RunAs "%A_ScriptFullPath%"
/restart
        else
            Run *RunAs "%A_AhkPath%" /restart
"%A_ScriptFullPath%"

```

```
    }
    ExitApp
}

MsgBox A_IsAdmin: %A_IsAdmin%`nCommand line:
%full_command_line%
```

If the user cancels the UAC dialog or Run fails for some other reason, the script will simply exit.

Using `/restart` ensures that a [single instance](#) prompt is not shown if the new instance of the script starts before `ExitApp` is called.

If UAC is disabled, `*RunAs` will launch the process without elevating it. Checking for `/restart` in the command line ensures that the script does not enter a runaway loop in that case. Note that `/restart` is a built-in switch, so is not included in the [array of command-line parameters](#).

The example can be modified to fit the script's needs:

- If the script absolutely requires admin rights, check `A_IsAdmin` a second time in case `*RunAs` failed to elevate the script (i.e. because UAC is disabled).
- To keep the script running even if the user cancels the UAC prompt, move `ExitApp` into the try block.
- To keep the script running even if it failed to restart (i.e. because the script file has been changed or deleted), remove `ExitApp` and use `RunWait` instead of `Run`. On success, `/restart` causes the new instance to terminate the old one. On failure, the new instance exits and `RunWait` returns.

[v1.0.92.01+]: If UAC is enabled, the AutoHotkey installer registers the `RunAs` verb for `.ahk` files, which allows `Run *RunAs script.ahk` to launch a script as admin with the default executable.

Related

[RunAs](#), [Process](#), [Exit](#), [CLSID List](#), [DllCall](#)

Examples

```
Run, Notepad.exe, C:\My Documents, max
```

```
Run, mailto:someone@domain.com?subject=This is the  
subject line&body=This is the message body's text.
```

```
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, %ComSpec% /c dir c:\ >>c:\DirTest.txt, ,  
min
```

```
Run, c:\DirTest.txt
```

```
Run, properties c:\DirTest.txt
```

```
Run, http://www.google.com ; i.e. any URL can be  
launched.
```

```
Run, mailto:someone@somedomain.com ; This should  
open the default e-mail application.
```

```
Run ::{20d04fe0-3aea-1069-a2d8-08002b30309d} ;  
Opens the "My Computer" folder.
```

```
Run ::{645ff040-5081-101b-9f08-00aa002f954e} ;  
Opens the Recycle Bin.
```

```
; To run multiple commands consecutively, use "&&"  
between each:
```

```
Run, %ComSpec% /c dir /b > C:\list.txt && type  
C:\list.txt && pause
```

```
; The following can be used to run a command and  
retrieve its output:
```

```
MsgBox % RunWaitOne("dir " A_ScriptDir)
```

```
; ...or run multiple commands in one go and
```

retrieve their output:

```
MsgBox % RunWaitMany(  
(  
echo Put your commands here,  
echo each one will be run,  
echo and you'll get the output.  
))
```

```
RunWaitOne(command) {  
    ; WshShell object:  
http://msdn.microsoft.com/en-us/library/aew9yb99  
    shell := ComObjCreate("WScript.Shell")  
    ; Execute a single command via cmd.exe  
    exec := shell.Exec(ComSpec " /C " command)  
    ; Read and return the command's output  
    return exec.StdOut.ReadAll()  
}
```

```
RunWaitMany(commands) {  
    shell := ComObjCreate("WScript.Shell")  
    ; Open cmd.exe with echoing of commands  
    disabled  
    exec := shell.Exec(ComSpec " /Q /K echo off")  
    ; Send the commands to execute, separated by  
    newline  
    exec.StdIn.WriteLine(commands "`nexit") ;  
    Always exit at the end!  
    ; Read and return the output of all commands  
    return exec.StdOut.ReadAll()  
}
```

; ExecScript: Executes the given code as a new
AutoHotkey process.

```
ExecScript(Script, Wait:=true)  
{  
    shell := ComObjCreate("WScript.Shell")
```

```
    exec := shell.Exec("AutoHotkey.exe  
/ErrorStdOut *")  
    exec.StdIn.Write(script)  
    exec.StdIn.Close()  
    if Wait  
        return exec.Stdout.ReadAll()  
}
```

; Example:

```
InputBox expr,, Enter an expression to evaluate as  
a new script,,,,,,,, Asc("*")  
result := ExecScript("FileAppend % (" expr "), *")  
MsgBox % "Result: " result
```

RunAs

Specifies a set of user credentials to use for all subsequent uses of [Run](#) and [RunWait](#).

```
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

User's password.

Domain

User's domain. To use a local account, leave this blank. If that fails to work, try using @YourComputerName.

Remarks

If the script is running with restricted privileges due to User Account Control (UAC), any programs it launches will typically also be restricted, even if RunAs is used. To elevate a process, use [Run *RunAs](#) instead.

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 (unless their [UseErrorLevel option](#) is in effect).

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 listed below.

Remarks

The shutdown code is a combination of the following values:

Logoff	0
Shutdown	1
Reboot	2
Force	4
Power down	8
Suspend/Hibernate	See DllCall example at the bottom of this page.
Turn monitor off	See PostMessage examples .

Add the required values together. For example, to shutdown and power down the code would be **9** (shutdown + power down = 1 + 8 = 9). Alternatively, an [expression](#) such as 1+8 can be specified.

The "Force" value (4) forces all open applications to close. It should only be used in an emergency because it may cause any open applications to lose data.

The "Power down" value shuts down the system and turns off the power.

On a related note, 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  
  
; Call the Windows API function "SetSuspendState"  
to have the system suspend or hibernate.  
; Parameter #1: Pass 1 instead of 0 to hibernate  
rather than suspend.  
; Parameter #2: Pass 1 instead of 0 to suspend  
immediately rather than asking each application  
for permission.  
; Parameter #3: Pass 1 instead of 0 to disable all  
wake events.  
DllCall("PowrProf\SetSuspendState", "int", 0,  
"int", 0, "int", 0)
```

RegDelete

Deletes a subkey or value from the registry.

New Syntax [v1.1.21+]

```
RegDelete, KeyName [, ValueName]
```

Parameters

KeyName

The full name of the registry key.

This must start with HKEY_LOCAL_MACHINE, HKEY_USERS, HKEY_CURRENT_USER, HKEY_CLASSES_ROOT, or HKEY_CURRENT_CONFIG (or the abbreviations for each of these, such as HKLM). To access a [remote registry](#), prepend the computer name and a colon (or [in v1.1.21+] a slash), as in this example:

```
\\workstation01:HKEY_LOCAL_MACHINE
```

ValueName

The name of the value to delete. **If omitted, the entire *KeyName* will be deleted.** To delete *KeyName*'s default value -- which is the value displayed as "(Default)" by RegEdit -- use the phrase `HK_DEFAULT` for this parameter.

Examples

```
RegDelete,  
HKEY_LOCAL_MACHINE\Software\SomeApplication,  
TestValue
```

Old Syntax

Deprecated: This syntax is not recommended for use in new scripts. Use the [new syntax](#) instead.

```
RegDelete, RootKey, SubKey [, ValueName]
```

Parameters

RootKey

Must be either HKEY_LOCAL_MACHINE, HKEY_USERS, HKEY_CURRENT_USER, HKEY_CLASSES_ROOT, or HKEY_CURRENT_CONFIG (or the abbreviations for each of these, such as HKLM). To access a [remote registry](#), prepend the computer name and a colon (or [in v1.1.21+] a slash), as in this example:

```
\\workstation01:HKEY_LOCAL_MACHINE
```

SubKey

The name of the subkey (e.g. Software\SomeApplication).

ValueName

The name of the value to delete. **If omitted, the entire *SubKey* will be deleted.** To delete *SubKey*'s default value -- which is the value displayed as "(Default)" by RegEdit -- use the phrase `AHK_DEFAULT` for this parameter.

Examples

```
RegDelete, HKEY_LOCAL_MACHINE,  
Software\SomeApplication, TestValue
```

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

[A_LastError](#) is set to the result of the operating system's GetLastError() function.

Remarks

Warning: Deleting from the registry is potentially dangerous - please exercise caution!

To retrieve and operate upon multiple registry keys or values, consider using a [registry-loop](#).

For details about how to access the registry of a remote computer, see the remarks in [registry-loop](#).

To delete entries from the 64-bit sections of the registry in a 32-bit script or vice versa, use [SetRegView](#).

Related

[RegRead](#), [RegWrite](#), [Registry-loop](#), [SetRegView](#), [IniDelete](#)

RegRead

Reads a value from the registry.

New Syntax [v1.1.21+]

```
RegRead, OutputVar, KeyName [, ValueName]
```

Parameters

OutputVar

The name of the variable in which to store the retrieved value. If the value cannot be retrieved, the variable is made blank and [ErrorLevel](#) is set to 1.

KeyName

The full name of the registry key.

This must start with HKEY_LOCAL_MACHINE, HKEY_USERS, HKEY_CURRENT_USER, HKEY_CLASSES_ROOT, or HKEY_CURRENT_CONFIG (or the abbreviations for each of these, such as HKLM). To access a [remote registry](#), prepend the computer name and a colon (or [in v1.1.21+] a slash), as in this example:

```
\\workstation01:HKEY_LOCAL_MACHINE
```

ValueName

The name of the value to retrieve. If omitted, *KeyName*'s default value will be retrieved, which is the value displayed as "(Default)" by RegEdit. If there is no default value (that is, if RegEdit displays "value not set"), *OutputVar* is made blank and ErrorLevel is set to 1.

Examples

```
; Example: Retrieve the path of the Program Files  
directory.
```

```
; The line below ensures that the path of the 64-  
bit Program Files  
; directory is returned if the OS is 64-bit and
```

the script is not.

```
SetRegView 64 ; Requires [v1.1.08+]
```

```
RegRead, OutputVar,  
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion, ProgramFilesDir  
MsgBox, Program files are in: %OutputVar%
```

; Another way to retrieve the path of the Program Files directory:

```
EnvGet OutputVar, % A_Is64bitOS ? "ProgramW6432" :  
"ProgramFiles"  
MsgBox, Program files are in: %OutputVar%
```

Old Syntax

Deprecated: This syntax is not recommended for use in new scripts. Use the [new syntax](#) instead.

```
RegRead, OutputVar, RootKey, SubKey [, ValueName]
```

Parameters

OutputVar

The name of the variable in which to store the retrieved value. If the value cannot be retrieved, the variable is made blank and [ErrorLevel](#) is set to 1.

RootKey

Must be either HKEY_LOCAL_MACHINE, HKEY_USERS, HKEY_CURRENT_USER, HKEY_CLASSES_ROOT, or HKEY_CURRENT_CONFIG (or the abbreviations for each of these, such as HKLM). To access a [remote registry](#), prepend the computer name and a colon (or [in v1.1.21+] a slash), as in this example:

```
\\workstation01:HKEY_LOCAL_MACHINE
```

SubKey

The name of the subkey (e.g. Software\SomeApplication).

ValueName

The name of the value to retrieve. If omitted, *SubKey*'s default value will be retrieved, which is the value displayed as "(Default)" by RegEdit. If there is no default value (that is, if RegEdit displays "value not set"), *OutputVar* is made blank and [ErrorLevel](#) is set to 1.

Examples

```
; Example: Retrieve the path of the Program Files
```

directory.

```
; The line below ensures that the path of the 64-bit Program Files  
; directory is returned if the OS is 64-bit and the script is not.
```

```
SetRegView 64 ; Requires [v1.1.08+]
```

```
RegRead, OutputVar, HKEY_LOCAL_MACHINE,  
SOFTWARE\Microsoft\Windows\CurrentVersion,  
ProgramFilesDir  
MsgBox, Program files are in: %OutputVar%
```

```
; Another way to retrieve the path of the Program Files directory:
```

```
EnvGet OutputVar, % A_Is64bitOS ? "ProgramW6432" :  
"ProgramFiles"  
MsgBox, Program files are in: %OutputVar%
```

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[ErrorLevel](#) is set to 1 if there was a problem (such as a nonexistent key or value) or 0 otherwise.

[A_LastError](#) is set to the result of the operating system's GetLastError() function.

Remarks

Currently only the following value types are supported: REG_SZ, REG_EXPAND_SZ, REG_MULTI_SZ, REG_DWORD, and REG_BINARY.

REG_DWORD values are always expressed as positive decimal numbers.

When reading a REG_BINARY key the result is a string of hex characters. For example, the REG_BINARY value of 01,a9,ff,77 will be read as the string 01A9FF77.

When reading a REG_MULTI_SZ key, each of the components ends in a linefeed character (`\n`). If there are no components, *OutputVar* will be made blank. See [FileSelectFile](#) for an example of how to extract the individual components from *OutputVar*.

[v1.1.10.01+]: REG_BINARY values larger than 64K can also be read.

To retrieve and operate upon multiple registry keys or values, consider using a [registry-loop](#).

For details about how to access the registry of a remote computer, see the remarks in [registry-loop](#).

To read and write entries from the 64-bit sections of the registry in a 32-bit script or vice versa, use [SetRegView](#).

Related

[RegDelete](#), [RegWrite](#), [Registry-loop](#), [SetRegView](#), [IniRead](#)

Examples

```
; The following example retrieves the TYPE of a
registry value (e.g. REG_SZ or REG_DWORD).
MsgBox % RegKeyType("HKCU", "Environment", "TEMP")
return
```

```
RegKeyType(RootKey, SubKey, ValueName) ; This
function returns the type of the specified value.
{
    Loop, %RootKey%, %SubKey%
        if (A_LoopRegName = ValueName)
            return A_LoopRegType
    return "Error"
}
```

RegWrite

Writes a value to the registry.

New Syntax [v1.1.21+]

```
RegWrite, ValueType, KeyName [, ValueName, Value]
```

Parameters

ValueType

Must be either REG_SZ, REG_EXPAND_SZ, REG_MULTI_SZ, REG_DWORD, or REG_BINARY.

KeyName

The full name of the registry key.

This must start with HKEY_LOCAL_MACHINE, HKEY_USERS, HKEY_CURRENT_USER, HKEY_CLASSES_ROOT, or HKEY_CURRENT_CONFIG (or the abbreviations for each of these, such as HKLM). To access a [remote registry](#), prepend the computer name and a colon (or [in v1.1.21+] a slash), as in this example:

```
\\workstation01:HKEY_LOCAL_MACHINE
```

ValueName

The name of the value that will be written to. If blank or omitted, *KeyName*'s default value will be used, which is the value displayed as "(Default)" by RegEdit.

Value

The value to be written. If omitted, it will default to an empty (blank) string, or 0, depending on *ValueType*. If the text is long, it can be broken up into several shorter lines by means of a [continuation section](#), which might improve readability and maintainability.

Examples

```
RegWrite, REG_SZ,
```

```
HKEY_LOCAL_MACHINE\SOFTWARE\TestKey, MyValueName,  
Test Value  
RegWrite, REG_BINARY,  
HKEY_CURRENT_USER\Software\TEST_APP, TEST_NAME,  
01A9FF77  
RegWrite, REG_MULTI_SZ,  
HKEY_CURRENT_USER\Software\TEST_APP, TEST_NAME,  
Line1`nLine2
```

Old Syntax

Deprecated: This syntax is not recommended for use in new scripts. Use the [new syntax](#) instead.

```
RegWrite, ValueType, RootKey, SubKey [, ValueName, Value]
```

Parameters

ValueType

Must be either REG_SZ, REG_EXPAND_SZ, REG_MULTI_SZ, REG_DWORD, or REG_BINARY.

RootKey

Must be either HKEY_LOCAL_MACHINE, HKEY_USERS, HKEY_CURRENT_USER, HKEY_CLASSES_ROOT, or HKEY_CURRENT_CONFIG (or the abbreviations for each of these, such as HKLM). To access a [remote registry](#), prepend the computer name and a colon (or [in v1.1.21+] a slash), as in this example:

```
\\workstation01:HKEY_LOCAL_MACHINE
```

SubKey

The name of the subkey (e.g. Software\SomeApplication). If *SubKey* does not exist, it is created (along with its ancestors, if necessary). If *SubKey* is left blank, the value is written directly into *RootKey* (though some operating systems might refuse to write in HKEY_CURRENT_USER's top level).

ValueName

The name of the value that will be written to. If blank or omitted, *SubKey*'s default value will be used, which is the value displayed as "(Default)" by RegEdit.

Value

The value to be written. If omitted, it will default to an empty (blank) string, or 0, depending on *ValueType*. If the text is long, it can be broken up into several shorter lines by means of a [continuation section](#), which might improve readability and maintainability.

Examples

```
RegWrite, REG_SZ, HKEY_LOCAL_MACHINE,  
SOFTWARE\TestKey, MyValueName, Test Value  
RegWrite, REG_BINARY, HKEY_CURRENT_USER,  
Software\TEST_APP, TEST_NAME, 01A9FF77  
RegWrite, REG_MULTI_SZ, HKEY_CURRENT_USER,  
Software\TEST_APP, TEST_NAME, Line1`nLine2
```

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

[A_LastError](#) is set to the result of the operating system's GetLastError() function.

Remarks

If *ValueType* is REG_DWORD, *Value* should be between -2147483648 and 4294967295 (0xFFFFFFFF).

When writing a REG_BINARY key, use a string of hex characters, e.g. the REG_BINARY value of 01,a9,ff,77 can be written by using the string 01A9FF77.

When writing a REG_MULTI_SZ key, you must separate each component from the next with a linefeed character (`\n`). The last component may optionally end with a linefeed as well. No blank components are allowed. In other words, do not specify two linefeeds in a row (`\n\n`) because that will result in a shorter-than-expected value being written to the registry.

[v1.1.10.01+]: REG_BINARY and REG_MULTI_SZ values larger than 64K are also supported. In older versions, they are truncated to 64K.

To retrieve and operate upon multiple registry keys or values, consider using a [registry-loop](#).

For details about how to access the registry of a remote computer, see the remarks in [registry-loop](#).

To read and write entries from the 64-bit sections of the registry in a 32-bit script or vice versa, use [SetRegView](#).

Related

[RegDelete](#), [RegRead](#), [Registry-loop](#), [SetRegView](#), [IniWrite](#)

SetRegView [v1.1.08+]

Sets the registry view used by RegRead, RegWrite, RegDelete and registry loops.

SetRegView, RegView

Parameters

RegView

Specify **32** to view the registry as a 32-bit application would, or **64** to view the registry as a 64-bit application would.

Specify the word **Default** to restore normal behaviour.

General Remarks

This command is only useful on Windows 64-bit. It has no effect on Windows 32-bit.

On 64-bit systems, 32-bit applications run on a subsystem of Windows called [WOW64](#). By default, the system redirects certain [registry keys](#) to prevent conflicts. For example, in a 32-bit script, `HKLM\SOFTWARE\AutoHotkey` is redirected to `HKLM\SOFTWARE\Wow6432Node\AutoHotkey`.

SetRegView allows the registry commands in a 32-bit script to access redirected keys in the 64-bit registry view and vice versa.

The built-in variable `A_RegView` contains the current setting. Every newly launched [thread](#) (such as a [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

[RegRead](#), [RegWrite](#), [RegDelete](#), [Loop \(registry\)](#)

Examples

Example 1 shows how to set a specific registry view, and how registry redirection affects the script.

```
; Access the registry as a 32-bit application
would.
SetRegView 32
RegWrite REG_SZ, HKLM, SOFTWARE\Test.ahk, Value,
123

; Access the registry as a 64-bit application
would.
SetRegView 64
RegRead value, HKLM,
SOFTWARE\Wow6432Node\Test.ahk, Value
RegDelete HKLM, SOFTWARE\Wow6432Node\Test.ahk

MsgBox Read value '%value%' via Wow6432Node.

; Restore the registry view to the default,
which
; depends on whether the script is 32-bit or 64-
bit.
SetRegView Default
;...
```

Example 2 shows how to detect the type of EXE and operating system on which the script is running.

```
if (A_PtrSize = 8)
    script_is := "64-bit"
else ; if (A_PtrSize = 4)
    script_is := "32-bit"

if (A_Is64bitOS)
```

```
    OS_is := "64-bit"  
else  
    OS_is := "32-bit, which has only a single  
registry view"  
  
MsgBox This script is %script_is%, and the OS is  
%OS_is%.
```

ImageSearch

Searches a region of the screen for an image.

ImageSearch, OutputVarX, OutputVarY, X1, Y1, X2, Y2, ImageFile

Parameters

OutputVarX/Y

The names of the variables in which to store the X and Y coordinates of the upper-left pixel of where the image was found on the screen (if no match is found, the variables are made blank). Coordinates are relative to the active window unless **CoordMode** was used to change that.

Either or both of these parameters may be left blank, in which case ErrorLevel (see below) can be used to determine whether a match was found.

X1,Y1

The X and Y coordinates of the upper left corner of the rectangle to search, which can be **expressions**. **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, which can be **expressions**. Coordinates are relative to the active window unless **CoordMode** was used to change that.

ImageFile

The file name of an image, which is assumed to be in **%A_WorkingDir%** if an absolute path isn't specified. All operating systems support GIF, JPG, BMP, ICO, CUR, and ANI images (BMP images must be 16-bit or higher). Other sources of icons include the following types of files: EXE, DLL, CPL, SCR, and other types that contain icon resources. On

Windows XP or later, additional image formats such as PNG, TIF, 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

Options: Zero or more of the following strings may be also be present immediately before the name of the file. Separate each option from the next with a single space or tab. For example: `*2 *w100 *h-1 C:\Main Logo.bmp`.

***IconN:** To use an icon group other than the first one in the file, specify `*Icon` followed immediately by the number of the group. For example, `*Icon2` would load the default icon from the second icon group.

***n (variation):** Specify for **n** 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 each pixel's color. For example, `*2` would allow two shades of variation. This parameter is helpful if the coloring of the image varies slightly or if *ImageFile* uses a format such as GIF or JPG that does not accurately represent an image on the screen. If you specify 255 shades of variation, all colors will match. The default is 0 shades.

***TransN:** This option makes it easier to find a match by specifying one color within the image that will match any color on the screen. It is most commonly used to find PNG, GIF, and TIF files that have some transparent areas (however, icons do not need this option because their transparency is automatically supported). For GIF files, `*TransWhite` might be most likely to work. For PNG and TIF files, `*TransBlack` might be best. Otherwise, specify for **N** some other color name or RGB value (see the [color chart](#) for guidance, or use [PixelGetColor](#) in its RGB mode). Examples: `*TransBlack`, `*TransFFFAA`, `*Trans0xFFFAA`.

***wn** and ***hn:** Width and height to which to scale the image (this width and height also determines which icon to load from a multi-icon .ICO file). If both these options are omitted, icons loaded from ICO, DLL, or

EXE files are scaled to the system's default small-icon size, which is usually 16 by 16 (you can force the actual/internal size to be used by specifying `*w0 *h0`). Images that are not icons are loaded at their actual size. 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.

[v1.1.23+]: A [bitmap or icon handle](#) can be used instead of a filename. For example, `HBITMAP: *%handle%` .

ErrorLevel

[v1.1.04+]: This command is able to throw an exception if there was a problem while searching. For more information, see [Runtime Errors](#).

[ErrorLevel](#) is set to 0 if the image 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 (such as failure to open the image file or a badly formatted option).

Remarks

ImageSearch can be used to detect graphical objects on the screen that either lack text or whose text cannot be easily retrieved. For example, it can be used to discover the position of picture buttons, icons, web page links, or game objects. Once located, such objects can be clicked via [Click](#).

A strategy that is sometimes useful is to search for a small clipping from an image rather than the entire image. This can improve reliability in cases where the image as a whole varies, but certain parts within it are always the same. One way to extract a clipping is to:

1. Press `Alt` `PrintScreen` visible in the active window. This places a screenshot on the clipboard.
2. Open an image processing program such as Paint.
3. Paste the contents of the clipboard (that is, the screenshot).
4. Select a region that does not vary and that is unique to the image.

5. Copy and paste that region to a new image document.
6. Save it as a small file for use with ImageSearch.

To be a match, an image on the screen must be the same size as the one loaded via the *ImageFile* parameter and its options.

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. By contrast, images that lie partially beneath the mouse cursor can usually be detected. The exception to this is game cursors, which in most cases will obstruct any images beneath them.

Since the search starts at the top row of the region and moves downward, if there is more than one match, the one closest to the top will be found.

Icons containing a transparent color automatically allow that color to match any color on the screen. Therefore, the color of what lies behind the icon does not matter.

ImageSearch supports 8-bit color screens (256-color) or higher.

The search behavior may vary depending on the display adapter's color depth (especially for GIF and JPG files). Therefore, if a script will run under multiple color depths, it is best to test it on each depth setting. You can use the shades-of-variation option (*n) to help make the behavior consistent across multiple color depths.

If the image on the screen is translucent, ImageSearch will probably fail to find it. To work around this, try the shades-of-variation option (*n) or make the window temporarily opaque via `WinSet, Transparent, Off`.

Related

[PixelSearch](#), [PixelGetColor](#), [CoordMode](#), [MouseGetPos](#)

Examples

```
ImageSearch, FoundX, FoundY, 40,40, 300, 300,  
C:\My Images\test.bmp
```

CoordMode Pixel ; Interprets the coordinates below as relative to the screen rather than the active window.

```
ImageSearch, FoundX, FoundY, 0, 0, A_ScreenWidth, A_ScreenHeight, *Icon3
```

```
%A_ProgramFiles%\SomeApp\SomeApp.exe
```

```
if ErrorLevel = 2
```

```
    MsgBox Could not conduct the search.
```

```
else if ErrorLevel = 1
```

```
    MsgBox Icon could not be found on the screen.
```

```
else
```

```
    MsgBox The icon was found at
```

```
%FoundX%x%FoundY%.
```

PixelGetColor

Retrieves the color of the pixel at the specified x,y coordinates.

```
PixelGetColor, OutputVar, X, Y [, Alt|Slow|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 intensity of 80 for its blue and red components but an intensity of 00 for its green component.

X, Y

The X and Y coordinates of the pixel, which can be [expressions](#). Coordinates are relative to the active window unless [CoordMode](#) was used to change that.

Alt|Slow|RGB

This parameter may contain zero or more of the following words. If more than one word is present, separate each from the next with a space (e.g. `Alt RGB`).

Alt [v1.0.43.10+]: Uses an alternate method to retrieve the color, which should be used when the normal method produces invalid or inaccurate colors for a particular type of window. This method is about 10% slower than the normal method.

Slow [v1.0.43.10+]: Uses a more elaborate method to retrieve the color, which may work in certain full-screen applications when the other methods fail. This method is about three times slower than the normal method. Note: *Slow* takes precedence over *Alt*, so there is no need to specify *Alt* in this case.

RGB: Retrieves the color 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

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[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. By contrast, pixels beneath the mouse cursor can usually be detected. The exception to this is game cursors, which in most cases will hide any pixels beneath them.

Use Window Spy (available in tray icon menu) or the example at the bottom of this page to determine the colors currently on the screen.

Known limitations:

- A window that is [partially transparent](#) or that has one of its colors marked invisible ([TransparentColor](#)) typically yields colors for the window behind itself rather than its own.
- `PixelGetColor` might not produce accurate results for certain applications. If this occurs, try specifying the word *Alt* or *Slow* in the last parameter.

Related

[PixelSearch](#), [ImageSearch](#), [CoordMode](#), [MouseGetPos](#)

Example

```
^!z:: ; Control+Alt+Z hotkey.  
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.

```
PixelSearch, OutputVarX, OutputVarY, X1, Y1, X2, Y2,  
ColorID [, Variation, Fast|RGB]
```

Parameters

OutputVarX/Y

The names of the variables in which to store the X and Y coordinates of the first pixel that matches *ColorID* (if no match is found, the variables are made blank). Coordinates are relative to the active window unless [CoordMode](#) was used to change that.

Either or both of these parameters may be left blank, in which case *ErrorLevel* (see below) can be used to determine whether a match was found.

X1, Y1

The X and Y coordinates of the upper left corner of the rectangle to search, which can be [expressions](#). **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, which can be [expressions](#). 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, which can be an [expression](#). Color IDs can be determined using Window Spy (accessible from the tray menu) or via [PixelGetColor](#). For 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 (can be an [expression](#)). This parameter is helpful if the color sought is not always exactly the same shade. If you specify 255 shades of variation, all colors will match. The default is 0 shades.

Fast|RGB

This parameter may contain the word Fast, RGB, or both (if both are present, separate them with a space; that is, *Fast RGB*).

Fast: Uses a faster searching method that in most cases dramatically reduces the amount of CPU time used by the search. Although color depths as low as 8-bit (256-color) are supported, the fast mode performs much better in 24-bit or 32-bit color. If the screen's color depth is 16-bit or lower, the *Variation* parameter might behave slightly differently in fast mode than it does in slow mode. Finally, the fast mode searches the screen row by row (top down) instead of column by column. Therefore, it might find a different pixel than that of the slow mode if there is more than one matching pixel.

RGB: Causes *ColorID* to be interpreted as an RGB value instead of BGR. In other words, the red and blue components are swapped.

ErrorLevel

[v1.1.04+]: This command is able to throw an exception if there was a problem while searching. For more information, see [Runtime Errors](#).

[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. By contrast, pixels

beneath the mouse cursor can usually be detected. The exception to this is cursors in games, which in most cases will hide any pixels beneath them.

Slow mode only: By default, the search starts at the upper-left pixel of the region and checks all pixels vertically beneath it for a match. If no match is found there, the search continues to the right, column by column, until it finds a matching pixel. The default left-to-right search order can be inverted by swapping *X1* and *X2* in the parameter list. In other words, if *X1* is greater than *X2*, the search will be conducted from right to left, starting at column *X1*. Similarly, if *Y1* is greater than *Y2*, each column of pixels to be searched starting at the bottom rather than the top. Finally, 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](#), [ImageSearch](#), [CoordMode](#), [MouseGetPos](#)

Example

```
PixelSearch, Px, Py, 200, 200, 300, 300, 0x9d6346,  
3, Fast  
if ErrorLevel  
    MsgBox, That color was not found in the  
    specified region.  
else  
    MsgBox, A color within 3 shades of variation  
    was found at X%Px% Y%Py%.
```

SoundBeep

Emits a tone from the PC speaker.

```
SoundBeep [ , Frequency, Duration ]
```

Parameters

Frequency

The frequency of the sound, which can be an [expression](#). It should be a number between 37 and 32767. If omitted, the frequency will be 523.

Duration

The duration of the sound, in milliseconds (can be an [expression](#)). If omitted, the duration will be 150.

Remarks

The script waits for the sound to finish before continuing. In addition, system responsiveness might be reduced during sound production.

If the computer lacks a sound card, a standard beep is played through the PC speaker.

To produce the standard system sounds instead of beeping the PC Speaker, see the asterisk mode of [SoundPlay](#).

Related

[SoundPlay](#)

Example

```
SoundBeep ; Play the default pitch and duration.
```

```
SoundBeep, 750, 500 ; Play a higher pitch for  
half a second.
```

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 *ControlTypes* 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 [SetFormat](#).

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, ANALOG, HEADPHONES, or N/A. 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 *ComponentType* (two of type LINE, for example), usually the first contains the playback settings and the second contains the recording settings. To access an instance other than the first, append a colon and a number to this parameter. For example: `Analog:2` is the second instance of the analog component.

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, EQUALIZER, or the number of a valid control type (see [soundcard analysis script](#)). If the specified *ComponentType* lacks the specified *ControlType*, *ErrorLevel* will indicate the problem.

Note: Sound devices usually support only VOLUME (or VOL) and MUTE, although others may be available depending on Windows and the sound device.

DeviceNumber

A number between 1 and the total number of supported devices. If this parameter is omitted, it defaults to 1 (the first sound device), or on Windows Vista or above, the system's default device for playback. This parameter can be an [expression](#). The [soundcard analysis script](#) may help determine which number to use.

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

Support for Windows Vista and later was added in v1.1.10.

To discover the capabilities of the sound devices (mixers) installed on the system -- such as the available component types and control types -- run the [soundcard analysis script](#).

For more functionality and finer grained control over audio, consider using the [VA library](#).

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  
    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

Retrieves 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

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[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](#).

[v1.1.10+]: On Windows Vista and later, this command is equivalent to [SoundGet](#) with *ComponentType* set to `Wave` and *ControlType* set to `Volume`.

Related

[SoundSetWaveVolume](#), [SoundSet](#), [SoundGet](#), [SoundPlay](#)

Example

```
SoundGetWaveVolume, OutputVar  
MsgBox, The current wave output volume level is  
%OutputVar%`%.
```

SoundPlay

Plays 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.

To produce standard system sounds, specify an asterisk followed by a number as shown below. Note: the *Wait* parameter has no effect in this mode.

***-1:** Simple beep. If the sound card is not available, the sound is generated using the speaker.

***16:** Hand (stop/error)

***32:** Question

***48:** Exclamation

***64:** Asterisk (info)

Wait

If omitted, the script's [current thread](#) will move on to the next command(s) while the file is playing. To avoid this, specify 1 or the word WAIT, which causes the current thread to wait until the file is finished playing before continuing. Even while waiting, new [threads](#) can be launched via [hotkey](#), [custom menu item](#), or [timer](#).

Known limitation: If the WAIT parameter is omitted, the OS might consider the playing file to be "in use" until the script closes or until another file is played (even a nonexistent file).

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[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.

If a file is playing and the current script plays a second file, the first file will be stopped so that the second one can play. On some systems, certain file types might stop playing even when an entirely separate script plays a new file.

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 exited, any currently-playing file that it started will stop.

Related

[SoundBeep](#), [SoundGet](#), [SoundSet](#), [SoundGetWaveVolume](#), [SoundSetWaveVolume](#), [MsgBox](#), [Threads](#)

Example

```
SoundPlay, %A_WinDir%\Media\ding.wav
SoundPlay *-1 ; Simple beep. If the sound card is
not available, the sound is generated using the
speaker.
```

SoundSet

Changes various settings of a sound device (master mute, master volume, etc.)

```
SoundSet, NewSetting [, ComponentType, ControlType,  
DeviceNumber ]
```

Parameters

NewSetting

Percentage number between -100 and 100 inclusive (it can be a floating point number or [expression](#)). 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 *NewSetting*.

For *ControlTypes* 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).

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, ANALOG, HEADPHONES, or N/A. 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 *ComponentType* (two of type LINE, for example), usually the first contains the playback settings and

the second contains the recording settings. To access an instance other than the first, append a colon and a number to this parameter. For example: `Analog : 2` is the second instance of the analog component.

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, EQUALIZER, or the number of a valid control type (see [soundcard analysis script](#)). If the specified *ComponentType* lacks the specified *ControlType*, *ErrorLevel* will indicate the problem.

Note: Sound devices usually support only VOLUME (or VOL) and MUTE, although others may be available depending on Windows and the sound device.

DeviceNumber

A number between 1 and the total number of supported devices. If this parameter is omitted, it defaults to 1 (the first sound device), or on Windows Vista or above, the system's default device for playback. This parameter can be an [expression](#). The [soundcard analysis script](#) may help determine which number to use.

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[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

[v1.1.10+]: This command supports Windows Vista and later.

For more functionality and finer grained control over audio, consider using the [VA library](#).

An alternative way to adjust the volume is to have the script send volume-control keystrokes to change the master volume for the entire system, such as in the example below:

```
Send {Volume_Up} ; Raise the master volume by 1 interval (typically 5%).  
Send {Volume_Down 3} ; Lower the master volume by 3 intervals.  
Send {Volume_Mute} ; Mute/unmute the master volume.
```

To discover the capabilities of the sound devices (mixers) installed on the system -- such as the available component types and control types -- run the [soundcard analysis script](#).

Windows 2000/XP/2003: When SoundSet changes the volume of a component, all of that component's channels (e.g. left and right) are set to the same level. In other words, any off-center "balance" that may have been set previously is lost. This can be avoided for the WAVE component by using [SoundSetWaveVolume](#) instead, which attempts to preserve the existing balance when changing the volume level.

[v1.1.10+]: On Windows Vista and later, SoundSet attempts to preserve the existing balance when changing the volume level.

Use [SoundGet](#) to retrieve the current value of a setting.

Related

[SoundGet](#), [SoundGetWaveVolume](#), [SoundSetWaveVolume](#), [SoundPlay](#)

Examples

```
; BASIC EXAMPLES:
SoundSet, 50 ; Set the master volume to 50%
SoundSet +10 ; Increase master volume by 10%
SoundSet -10 ; Decrease master volume by 10%
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
    MsgBox, The BASS setting is not supported for
MASTER.
```

Soundcard Analysis

Use the following script to discover your soundcard's capabilities (component types and control types). It displays the results in a simple ListView. Alternatively, a script for Windows Vista and later which provides more detail (such as display names of devices) can be downloaded from the following forum topic: <http://www.autohotkey.com/board/topic/90877-/>

```
SetBatchLines -1
SplashTextOn,,, Gathering Soundcard Info...

; Most of the pure numbers below probably don't
exist in any mixer, but they're queried for
completeness.
; The numbers correspond to the following items
(in order): CUSTOM, BOOLEANMETER, SIGNEDMETER,
PEAKMETER,
; UNSIGNEDMETER, BOOLEAN, BUTTON, DECIBELS,
SIGNED, UNSIGNED, PERCENT, SLIDER, FADER,
SINGLESELECT, MUX,
```

```
; MULTIPLESELECT, MIXER, MICROTIME, MILLITIME
ControlTypes =
VOLUME, ONOFF, MUTE, MONO, LOUDNESS, STEREOENH, BASSBOOS
T, PAN, QSOUNDPAN, BASS, TREBLE, EQUALIZER, 0x00000000,
0x10010000, 0x10020000, 0x10020001, 0x10030000, 0x2001
0000, 0x21010000, 0x30040000, 0x30020000, 0x30030000, 0
x30050000, 0x40020000, 0x50030000, 0x70010000, 0x70010
001, 0x71010000, 0x71010001, 0x60030000, 0x61030000
```

```
ComponentTypes =
MASTER, HEADPHONES, DIGITAL, LINE, MICROPHONE, SYNTH, CD
, TELEPHONE, PCSPEAKER, WAVE, AUX, ANALOG, N/A
```

```
; Create a ListView and prepare for the main loop:
Gui, Add, ListView, w400 h400 vMyListView,
Component Type|Control Type|Setting|Mixer
LV_ModifyCol(4, "Integer")
SetFormat, Float, 0.2 ; Limit number of decimal
places in percentages to two.
```

```
Loop ; For each mixer number that exists in the
system, query its capabilities.
{
    CurrMixer := A_Index
    SoundGet, Setting,,, %CurrMixer%
    if ErrorLevel = Can't Open Specified Mixer ;
Any error other than this indicates that the mixer
exists.
        break

    ; For each component type that exists in this
mixer, query its instances and control types:
    Loop, parse, ComponentTypes, ` ,
    {
        CurrComponent := A_LoopField
        ; First check if this component type even
exists in the mixer:
```

```

        SoundGet, Setting, %CurrComponent%, ,
%CurrMixer%
        if ErrorLevel = Mixer Doesn't Support This
Component Type
            continue ; Start a new iteration to
move on to the next component type.
        Loop ; For each instance of this
component type, query its control types.
        {
            CurrInstance := A_Index
            ; First check if this instance of this
instance even exists in the mixer:
            SoundGet, Setting,
%CurrComponent%:%CurrInstance%, , %CurrMixer%
            ; Checking for both of the following
errors allows this script to run on older
versions:
            if ErrorLevel in Mixer Doesn't Have
That Many of That Component Type,Invalid Control
Type or Component Type
                break ; No more instances of this
component type.
            ; Get the current setting of each
control type that exists in this instance of this
component:
            Loop, parse, ControlTypes, `,
            {
                CurrControl := A_LoopField
                SoundGet, Setting,
%CurrComponent%:%CurrInstance%, %CurrControl%,
%CurrMixer%
                ; Checking for both of the
following errors allows this script to run on
older versions:
                if ErrorLevel in Component Doesn't
Support This Control Type,Invalid Control Type or
Component Type

```

```

        continue
        if ErrorLevel ; Some other error,
which is unexpected so show it in the results.
        Setting := ErrorLevel
        ComponentString := CurrComponent
        if CurrInstance > 1
            ComponentString =
%ComponentString%:%CurrInstance%
            LV_Add("", ComponentString,
CurrControl, Setting, CurrMixer)
        } ; For each control type.
    } ; For each component instance.
} ; For each component type.
} ; For each mixer.

Loop % LV_GetCount("Col") ; Auto-size each column
to fit its contents.
    LV_ModifyCol(A_Index, "AutoHdr")

SplashTextOff
Gui, Show
return

GuiClose:
ExitApp

```

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 or an [expression](#)). 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

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[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](#).

[v1.1.10+]: On Windows Vista and later, this command is equivalent to [SoundSet](#) with *ComponentType* set to `Wave` and *ControlType* set to `Volume`.

Both commands attempt to preserve the existing balance between channels.

Windows 2000/XP/2003: Unlike [SoundSet](#), this command attempts to preserve the existing balance between channels (e.g. left and right) when changing the volume level.

Related

[SoundGetWaveVolume](#), [SoundSet](#), [SoundGet](#), [SoundPlay](#)

Examples

```
SoundSetWaveVolume, 50 ; Set the volume to its  
half-way point.  
SoundSetWaveVolume, -10 ; Decrease the current  
level by 10 (e.g. 80 would become 70).  
SoundSetWaveVolume, +20 ; Increase the current  
level by 20.
```

Asc()

Returns the numeric value of the first byte or UTF-16 code unit in the specified string.

```
Number := Asc(String)
```

Parameters

String

The string whose numeric value is retrieved.

Return Value

This function returns a numeric value in the range 0 to 255 (for ANSI) or 0 to 0xFFFF (for Unicode). See [Unicode vs ANSI](#) for details. If *String* is empty, it returns 0.

Remarks

This function is equivalent to `Transform, OutputVar, Asc`.

To allow for Unicode supplementary characters, use [Ord\(String\)](#) instead.

Related

[Transform](#), [Ord\(\)](#), [Chr\(\)](#)

Examples

```
; Both message boxes below show 116, because only  
the first character is considered.  
MsgBox, % Asc("t")  
MsgBox, % Asc("test")
```

Chr()

Returns the string (usually a single character) corresponding to the character code indicated by the specified number.

```
String := Chr(Number)
```

Parameters

Number

If Unicode is supported, *Number* is a Unicode character code between 0 and 0x10FFFF (or 0xFFFF prior to [v1.1.21]); otherwise it is an ANSI character code between 0 and 255.

Return Value

This function returns a string corresponding to *Number*. If *Number* is not in the valid range of character codes, an empty string is returned.

Remarks

This function supersedes [Transform](#), [OutputVar](#), [Chr](#).

The meaning of character codes greater than 127 depends on the [string encoding](#) in use, which in turn depends on whether a [Unicode or ANSI](#) executable is in use.

Common character codes include 9 (tab), 10 (linefeed), 13 (carriage return), 32 (space), 48-57 (the digits 0-9), 65-90 (uppercase A-Z), and 97-122 (lowercase a-z).

Related

[Transform](#), [Ord\(\)](#), [Asc\(\)](#)

Examples

```
MsgBox, % Chr(116) ; Shows "t".
```

Format [v1.1.17+]

Formats a variable number of input values according to a format string.

```
String := Format(FormatStr [, Values...])
```

Parameters

FormatStr

A format string composed of literal text and placeholders of the form `{Index:Format}`.

Index is an integer indicating which input value to use, where 1 is the first value.

Format is an optional format specifier, as described below.

Omit the index to use the next input value in the sequence (even if it has been used earlier in the string). For example, `"{2:i} {:i}"` formats the second and third input values as decimal integers, separated by a space. If *Index* is omitted, *Format* must still be preceded by `:`. Specify empty braces to use the next input value with default formatting: `{}`

Use `{{}}` and `{}}` to include literal braces in the string. Any other invalid placeholders are included in the result as is.

Whitespace inside the braces is not permitted (except as a flag).

Values

Input values to be formatted and inserted into the final string. Each value is a separate parameter. The first value has an index of 1.

To pass an array of values, use a [variadic function call](#):

```
arr := [13, 240]
```

```
MsgBox % Format("{2:x}{1:02x}", arr*)
```

Format Specifiers

Each format specifier can include the following components, in this order (without the spaces):

Flags Width .Precision ULT Type

Flags which affect output justification and prefixes: `-` `+` `0` `space` `#`

Width: a decimal integer which controls the minimum width of the formatted value, in characters. By default, values are right-aligned and spaces are used for padding. This can be overridden by using the `-` (left-align) and `0` (zero prefix) flags.

.Precision: a decimal integer which controls the maximum number of string characters, decimal places, or significant digits to output, depending on the output type. It must be preceded by a decimal point. Specifying a precision may cause the value to be truncated or rounded.

- `f`, `e`, `E`: *Precision* specifies the number of digits after the decimal point. The default is 6.
- `g`, `G`: *Precision* specifies the maximum number of significant digits. The default is 6.
- `s`: *Precision* specifies the maximum number of characters to be printed. Characters in excess of this are not printed.
- For the integer types (`d`, `i`, `u`, `x`, `X`, `o`), *Precision* acts like *Width* with the `0` prefix and a default of 1.

ULT [v1.1.20+]: specifies a case transformation to apply to a string value -- **U**pper, **L**ower or **T**itle. Valid only with the `s` type. For example `{:U}` or `{:.20Ts}`. Lower-case `l` and `t` are also supported, but `u` is reserved for unsigned integers.

Type: a character indicating how the input value should be interpreted. If omitted, it defaults to `s`.

Flag	Meaning	Default
-	Left align the result within the given field width (insert spaces to the right if needed).	Right align.
+	Use a sign (+ or -) to prefix the output value if it is of a signed type.	Sign appears only for negative signed values (-).
0	If <i>width</i> is prefixed by 0, leading zeros are added until the minimum width is reached. If both 0 and - appear, the 0 is ignored. If 0 is specified as an integer format (i, u, x, X, o, d) and a precision specification is also present - for example, { :04.d } - the 0 is ignored.	No padding.
space	Use a single space to prefix the output value with a space if it is signed and positive. The space is ignored if both the space and + flags appear.	No space appears.
#	<p>When it's used with the o, x, or X format, the # flag uses 0, 0x, or 0X, respectively, to prefix any nonzero output value.</p> <p>When it's used with the e, E, f, a or A format, the # flag forces the output value to contain a decimal point.</p> <p>When it's used with the g or G format, the # flag forces the output value to contain a decimal point and prevents the truncation of trailing zeros.</p> <p>Ignored when used with c, d, i, u, or s.</p>	

Type Character	Argument	Output format

<code>d</code> or <code>i</code>	Integer	Signed decimal integer.
<code>u</code>	Integer	Unsigned decimal integer.
<code>x</code> or <code>X</code>	Integer	Unsigned hexadecimal integer; uses "abcdef" or "ABCDEF" depending on the case of <code>x</code> . The <code>0x</code> prefix is not included unless the <code>#</code> flag is used, as in <code>{:#x}</code> . For hexadecimal formatting consistent with SetFormat , use <code>0x{:x}</code> or similar.
<code>o</code>	Integer	Unsigned octal integer.
<code>f</code>	Floating-point	Signed value that has the form <code>[-]dddd.dddd</code> , where <code>dddd</code> is one or more decimal digits. The number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the requested precision.
<code>e</code>	Floating-point	Signed value that has the form <code>[-]d.dddd e [sign]dd[d]</code> where <code>d</code> is one decimal digit, <code>dddd</code> is one or more decimal digits, <code>dd[d]</code> is two or three decimal digits depending on the output format and size of the exponent, and <code>sign</code> is <code>+</code> or <code>-</code> .
<code>E</code>	Floating-point	Identical to the <code>e</code> format except that <code>E</code> rather than <code>e</code> introduces the exponent.
<code>g</code>	Floating-point	Signed values are displayed in <code>f</code> or <code>e</code> format, whichever is more compact for the given value and precision. The <code>e</code> format is used only when the exponent of the value is less than <code>-4</code> or greater than or equal to the <i>precision</i> argument. Trailing zeros are truncated, and the decimal point appears only if one or more digits follow it.
<code>G</code>	Floating-point	Identical to the <code>g</code> format, except that <code>E</code> , rather than <code>e</code> , introduces the exponent

		(where appropriate).
a	Floating-point	Signed hexadecimal double-precision floating-point value that has the form <code>[?]<i>0xh.hhhh p±dd</i></code> , where <i>h.hhhh</i> are the hex digits (using lower case letters) of the mantissa, and <i>dd</i> are one or more digits for the exponent. The precision specifies the number of digits after the point.
A	Floating-point	Identical to the a format, except that <i>P</i> , rather than <i>p</i> , introduces the exponent.
p	Integer	Displays the argument as a memory address in hexadecimal digits.
s	String	Specifies a string. If the input value is numeric, it is automatically converted to a string using the script's current number format before the <i>Width</i> and <i>Precision</i> arguments are applied.
c	Character code	Specifies a single character by its ordinal value, similar to <code>Chr(n)</code> . If the input value is outside the expected range, it wraps around.

Remarks

Unlike `printf`, size specifiers are not supported. All integers and floating-point input values are 64-bit.

Related

[SetFormat](#), [FormatTime](#)

Example

```
; Simple substitution
```

```
s .= Format("{2}, {1}!`r`n", "World", "Hello")
; Padding with spaces
s .= Format("|{: -10}|`r`n|{:10}|`r`n", "Left",
"Right")
; Hexadecimal
s .= Format("{1:#x} {2:X} 0x{3:x}`r`n",
3735928559, 195948557, 0)
; Floating-point
s .= Format("{1:0.3f} {1:.10f}", 4*ATan(1))
```

ListVars ; Use AutoHotkey's main window to display monospaced text.

```
WinWaitActive ahk_class AutoHotkey
ControlSetText Edit1, %s%
WinWaitClose
```

FormatTime

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. For 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 M must be capitalized: M/d/yyyy h: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
dddd	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. For 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)
HH	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. For 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. Consequently, 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:

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).

0x10: Add marks for left-to-right reading order layout.

0x20: Add marks for right-to-left reading order layout.

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.

12: Combination of the above two.

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.

On a related note, addition and subtraction of dates and times can be performed with [EnvAdd](#) and [EnvSub](#).

Related

To convert in the reverse direction -- that is, *from* a formatted date/time *to* `YYYYMMDDHH24MISS` format -- see

www.autohotkey.com/forum/topic20405.html

See also: [Gui DateTime control](#), [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%
```

```

; Change the date-time stamp of a file:
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%.

```

```

; The following function converts the specified
number of seconds into the corresponding
; number of hours, minutes, and seconds (hh:mm:ss
format).

MsgBox % FormatSeconds(7384) ; 7384 = 2 hours + 3
minutes + 4 seconds. It yields: 2:03:04

FormatSeconds(NumberOfSeconds) ; Convert the
specified number of seconds to hh:mm:ss format.
{
    time = 19990101 ; *Midnight* of an arbitrary
date.
    time += %NumberOfSeconds%, seconds
    FormatTime, mmss, %time%, mm:ss
    return NumberOfSeconds//3600 ":" mmss
    /*
; Unlike the method used above, this would not
support more than 24 hours worth of seconds:
    FormatTime, hmmmss, %time%, h:mm:ss
    return hmmmss
    */
}

```

InStr()

Searches for a given occurrence of a string, from the left or the right.

```
FoundPos := InStr(Haystack, Needle [, CaseSensitive := false, StartingPos := 1, Occurrence := 1])
```

Parameters

Haystack

▮ The string whose content is searched.

Needle

▮ The string to search for.

CaseSensitive

▮ If the parameter *CaseSensitive* is omitted or false, the search is not case sensitive (the method of insensitivity depends on [StringCaseSense](#)); otherwise, the case must match exactly.

StartingPos

▮ If *StartingPos* is omitted, it defaults to 1 (the beginning of *Haystack*). Otherwise, specify 2 to start at the second character, 3 to start at the third, and so on.

▮ If *StartingPos* is beyond the length of *Haystack*, 0 is returned. [AHK_L 57+]: If *StartingPos* is 0 or negative, the search is conducted in reverse (right-to-left) beginning at that offset from the end.

▮ Regardless of the value of *StartingPos*, the return value is always relative to the first character of *Haystack*. For example, the position of "abc" in "123abc789" is always 4.

Occurrence [AHK_L 57+]

▮ If *Occurrence* is omitted, it defaults to the first match of the *Needle* in

Haystack. Specify 2 for *Occurrence* to return the position of the second match, 3 for the third match, etc.

Return Value

This function returns the position of an occurrence of the string *Needle* in the string *Haystack*. Position 1 is the first character; this is because 0 is synonymous with "false", making it an intuitive "not found" indicator.

Remarks

This function is a combination of [IfInString](#) and [StringGetPos](#).

[RegExMatch\(\)](#) can be used to search for a pattern (regular expression) within a string, making it much more flexible than [InStr\(\)](#). However, [InStr\(\)](#) is generally faster than [RegExMatch\(\)](#) when searching for a simple substring.

Related

[RegExMatch\(\)](#), [StringGetPos](#), [IfInString](#), [StringCaseSense](#), [if var in/contains](#), [MatchList](#), [if var between](#), [if var is type](#)

Example

```
; Example 1
MsgBox % InStr("123abc789","abc") ; Returns 4
```

```
; Example 2
Haystack := "The Quick Brown Fox Jumps Over the
Lazy Dog"
Needle := "Fox"
If InStr(Haystack, Needle)
    MsgBox, The string was found.
Else
    MsgBox, The string was not found.
```

```
; Example 3
```

```
Haystack := "The Quick Brown Fox Jumps Over the  
Lazy Dog"  
Needle := "the"  
MsgBox % InStr(Haystack, Needle, false, 1, 2) ;  
case insensitive search, return start position of  
second occurrence  
MsgBox % InStr(Haystack, Needle, true) ; case  
sensitive search, return start position of first  
occurrence, same result as above
```

Ord()

 [v1.1.21+]

Returns the ordinal value (numeric character code) of the first character in the specified string.

```
Number := Ord(String)
```

Parameters

String

The string whose ordinal value is retrieved.

Return Value

This function returns the ordinal value of *String*, or 0 if *String* is empty. If *String* begins with a Unicode supplementary character, this function returns the corresponding Unicode character code (a number between 0x10000 and 0x10FFFF). Otherwise it returns a value in the range 0 to 255 (for ANSI) or 0 to 0xFFFF (for Unicode). See [Unicode vs ANSI](#) for details.

Remarks

Apart from the Unicode supplementary character detection, this function is identical to [Asc\(\)](#).

Related

[Asc\(\)](#), [Chr\(\)](#)

Examples

```
; Both message boxes below show 116, because only  
the first character is considered.  
MsgBox, % Ord("t")
```

MsgBox, % Ord("test")

Regular Expressions (RegEx) - Quick Reference

Fundamentals

Match anywhere: By default, a regular expression matches a substring *anywhere* inside the string to be searched. For example, the regular expression **abc** matches `abc123`, `123abc`, and `123abcxyz`. To require the match to occur only at the beginning or end, use an [anchor](#).

Escaped characters: Most characters like `abc123` can be used literally inside a regular expression. However, the characters `\.*?+[]|()^$` must be preceded by a backslash to be seen as literal. For example, `\.` is a literal period and `\\` is a literal backslash. Escaping can be avoided by using `\Q...\E`. For example: `\QLiteral Text\E`.

Case-sensitive: By default, regular expressions are case-sensitive. This can be changed via the `"i"` option. For example, the pattern `i)abc` searches for `"abc"` without regard to case. See below for other modifiers.

Options (case sensitive)

At the very beginning of a regular expression, specify zero or more of the following options followed by a close-parenthesis. For example, the pattern "**im**)abc" would search for *abc* with the case-insensitive and multiline options (the parenthesis may be omitted when there are no options). Although this syntax breaks from tradition, it requires no special delimiters (such as forward-slash), and thus there is no need to escape such delimiters inside the pattern. In addition, performance is improved because the options are easier to parse.

i	Case-insensitive matching, which treats the letters A through Z as identical to their lowercase counterparts.
m	<p>Multiline. Views <i>Haystack</i> as a collection of individual lines (if it contains newlines) rather than as a single continuous line. Specifically, it changes the following:</p> <ol style="list-style-type: none">1) Circumflex (^) matches immediately after all internal newlines -- as well as at the start of <i>Haystack</i> where it always matches (but it does not match after a newline <i>at the very end</i> of <i>Haystack</i>).2) Dollar-sign (\$) matches before any newlines in <i>Haystack</i> (as well as at the very end where it always matches). <p>For example, the pattern "m)^abc\$" would not match the <i>Haystack</i>"xyz`r`nabc" unless the "m" option is present.</p> <p>The "D" option is ignored when "m" is present.</p>
s	DotAll. This causes a period (.) to match all characters including newlines (normally, it does not match newlines). However, when the newline character is at its default of CRLF (`r`n), two dots are required to match it (not one). Regardless of this option, a negative class such as [^a] always matches newlines.
	Ignores whitespace characters in the pattern except when escaped or inside a character class. The characters `n and `t are among those ignored

x	because by the time they get to PCRE, they are already raw/literal whitespace characters (by contrast, <code>\n</code> and <code>\t</code> are not ignored because they are PCRE escape sequences). The x option also ignores characters between a non-escaped <code>#</code> outside a character class and the next newline character, inclusive. This makes it possible to include comments inside complicated patterns. However, this applies only to data characters; whitespace may never appear within special character sequences such as <code>(?()</code> , which begins a conditional subpattern.
A	Forces the pattern to be anchored; that is, it can match only at the start of <i>Haystack</i> . Under most conditions, this is equivalent to explicitly anchoring the pattern by means such as <code>^</code> .
D	Forces dollar-sign (<code>\$</code>) to match at the very end of <i>Haystack</i> , even if <i>Haystack's</i> last item is a newline. Without this option, <code>\$</code> instead matches right before the final newline (if there is one). Note: This option is ignored when the "m" option is present.
J	Allows duplicate named subpatterns . This can be useful for patterns in which only one of a collection of identically-named subpatterns can match. Note: If more than one instance of a particular name matches something, only the leftmost one is stored. Also, variable names are not case-sensitive.
U	Ungreedy. Makes the quantifiers <code>*+?{ }</code> consume only those characters absolutely necessary to form a match, leaving the remaining ones available for the next part of the pattern. When the "U" option is not in effect, an individual quantifier can be made non-greedy by following it with a question mark. Conversely, when "U" is in effect, the question mark makes an individual quantifier greedy.
X	<code>PCRE_EXTRA</code> . Enables PCRE features that are incompatible with Perl. Currently, the only such feature is that any backslash in a pattern that is followed by a letter that has no special meaning causes the match to fail and <code>ErrorLevel</code> to be set accordingly. This option helps reserve unused backslash sequences for future use. Without this option, a backslash followed by a letter with no special meaning is treated as a literal (e.g. <code>\g</code> and <code>g</code> are both recognized as a literal <code>g</code>). Regardless of this option, non-alphabetic backslash sequences that have no special meaning are always treated as literals (e.g. <code>\/</code> and <code>/</code> are both recognized as forward-slash).

P	Position mode. This causes RegExMatch() to yield the position and length of the match and its subpatterns rather than their matching substrings. For details, see OutputVar .
S	Studies the pattern to try improve its performance. This is useful when a particular pattern (especially a complex one) will be executed many times. If PCRE finds a way to improve performance, that discovery is stored alongside the pattern in the cache for use by subsequent executions of the same pattern (subsequent uses of that pattern should also specify the S option because finding a match in the cache requires that the option letters exactly match, including their order).
C	Enables the auto-callout mode. See Regular Expression Callouts for more info.
\n	Switches from the default newline character (<code>\r\n</code>) to a solitary linefeed (<code>\n</code>), which is the standard on UNIX systems. The chosen newline character affects the behavior of anchors (^ and \$) and the dot/period pattern .
\r	Switches from the default newline character (<code>\r\n</code>) to a solitary carriage return (<code>\r</code>).
\a	[v1.0.46.06+]: <code>\a</code> recognizes any type of newline, namely <code>\r</code> , <code>\n</code> , <code>\r\n</code> , <code>\v</code> /VT/vertical tab/ <code>chr(0xB)</code> , <code>\f</code> /FF/formfeed/ <code>chr(0xC)</code> , and NEL/next-line/ <code>chr(0x85)</code> . [v1.0.47.05+]: Newlines can be restricted to only CR, LF, and CRLF by instead specifying (<code>*ANYCRLF</code>) in uppercase at the beginning of the pattern (after the options); e.g. <code>im (*ANYCRLF)^abc\$</code> .

Note: Spaces and tabs may optionally be used to separate each option from the next.

Commonly Used Symbols and Syntax

.	<p>By default, a dot matches any single character which is not part of a newline (<code>\r\n</code>) sequence, but this can be changed by using the DotAll (s), linefeed (\n), carriage return (\r), <code>\a</code> or (*ANYCRLF) options. For example, <code>ab.</code> matches <code>abc</code> and <code>abz</code> and <code>ab_</code>.</p>
*	<p>An asterisk matches zero or more of the preceding character, class, or subpattern. For example, <code>a*</code> matches <code>ab</code> and <code>aaab</code>. It also matches at the very beginning of any string that contains no "a" at all.</p> <p>Wildcard: The dot-star pattern <code>.*</code> is one of the most permissive because it matches zero or more occurrences of <i>any</i> character (except newline: <code>\r</code> and <code>\n</code>). For example, <code>abc.*123</code> matches <code>abcAnything123</code> as well as <code>abc123</code>.</p>
?	<p>A question mark matches zero or one of the preceding character, class, or subpattern. Think of this as "the preceding item is optional". For example, <code>colou?r</code> matches both <code>color</code> and <code>colour</code> because the "u" is optional.</p>
+	<p>A plus sign matches one or more of the preceding character, class, or subpattern. For example <code>a+</code> matches <code>ab</code> and <code>aaab</code>. But unlike <code>a*</code> and <code>a?</code>, the pattern <code>a+</code> does not match at the beginning of strings that lack an "a" character.</p>
{min,max}	<p>Matches between <i>min</i> and <i>max</i> occurrences of the preceding character, class, or subpattern. For example, <code>a{1,2}</code> matches <code>ab</code> but only the first two a's in <code>aaab</code>.</p> <p>Also, <code>{3}</code> means exactly 3 occurrences, and <code>{3,}</code> means 3 or more occurrences. Note: The specified numbers must be less than 65536, and the first must be less than or equal to the second.</p>

<p>[...]</p>	<p>Classes of characters: The square brackets enclose a list or range of characters (or both). For example, [abc] means "any single character that is either a, b or c". Using a dash in between creates a range; for example, [a-z] means "any single character that is between lowercase a and z (inclusive)". Lists and ranges may be combined; for example [a-zA-Z0-9_] means "any single character that is alphanumeric or underscore".</p> <p>A character class may be followed by *, ?, +, or {min,max}. For example, [0-9]+ matches one or more occurrence of any digit; thus it matches xyz123 but not abcxyz.</p> <p>The following POSIX named sets are also supported via the form [[:xxx:]], where xxx is one of the following words: alnum, alpha, ascii (0-127), blank (space or tab), cntrl (control character), digit (0-9), xdigit (hex digit), print, graph (print excluding space), punct, lower, upper, space (whitespace), word (same as <code>\w</code>).</p> <p>Within a character class, characters do not need to be escaped except when they have special meaning inside a class; e.g. [^a], [a\b], [a\]], and [\\a].</p>
<p>[^...]</p>	<p>Matches any single character that is not in the class. For example, [^/]* matches zero or more occurrences of any character that is <i>not</i> a forward-slash, such as <code>http://</code>. Similarly, [^0-9xyz] matches any single character that isn't a digit and isn't the letter x, y, or z.</p>
<p>\d</p>	<p>Matches any single digit (equivalent to the class [0-9]). Conversely, capital <code>\D</code> means "any <i>non</i>-digit". This and the other two below can also be used inside a class; for example, [\d.-] means "any single digit, period, or minus sign".</p>
<p>\s</p>	<p>Matches any single whitespace character, mainly space, tab, and newline (<code>\r</code> and <code>\n</code>). Conversely, capital <code>\S</code> means "any <i>non</i>-whitespace character".</p>

\w	Matches any single "word" character, namely alphanumeric or underscore. This is equivalent to [a-zA-Z0-9_] . Conversely, capital \W means "any <i>non</i> -word character".
^ \$	<p>Circumflex (^) and dollar sign (\$) are called <i>anchors</i> because they don't consume any characters; instead, they tie the pattern to the beginning or end of the string being searched.</p> <p>^ may appear at the beginning of a pattern to require the match to occur at the very beginning of a line. For example, ^abc matches abc123 but not 123abc.</p> <p>\$ may appear at the end of a pattern to require the match to occur at the very end of a line. For example, abc\$ matches 123abc but not abc123.</p> <p>The two anchors may be combined. For example, ^abc\$ matches only abc (i.e. there must be no other characters before or after it).</p> <p>If the text being searched contains multiple lines, the anchors can be made to apply to each line rather than the text as a whole by means of the "m" option. For example, m)^abc\$ matches 123`r`nabc`r`n789. But without the "m" option, it wouldn't match.</p>
\b	\b means "word boundary", which is like an anchor because it doesn't consume any characters. It requires the current character's status as a word character (\w) to be the opposite of the previous character's. It is typically used to avoid accidentally matching a word that appears inside some other word. For example, \bcat\b doesn't match catfish, but it matches cat regardless of what punctuation and whitespace surrounds it. Capital \B is the opposite: it requires that the current character <i>not</i> be at a word boundary.
	The vertical bar separates two or more alternatives. A match occurs if <i>any</i> of the alternatives is satisfied. For example, gray grey matches both gray and grey . Similarly, the pattern gr(a e)y

does the same thing with the help of the parentheses described below.

Items enclosed in parentheses are most commonly used to:

- Determine the order of evaluation. For example, **(Sun|Mon|Tues|Wednes|Thurs|Fri|Sat)day** matches the name of any day.
- Apply *****, **?**, **+**, or **{min,max}** to a *series* of characters rather than just one. For example, **(abc)+** matches one or more occurrences of the string "abc"; thus it matches **abcabc123** but not **ab123** or **bc123**.
- Capture a subpattern such as the dot-star in **abc(.*)xyz**. For example, **RegexMatch()** stores the substring that matches each subpattern in its **output array**. Similarly, **RegexReplace()** allows the substring that matches each subpattern to be reinserted into the result via **backreferences** like **\$1**. To use the parentheses without the side-effect of capturing a subpattern, specify **?:** as the first two characters inside the parentheses; for example: **(?:.*)**
- Change **options** on-the-fly. For example, **(?im)** turns on the case-insensitive and multiline options for the remainder of the pattern (or subpattern if it occurs inside a subpattern). Conversely, **(?-im)** would turn them both off. All options are supported except **DPS`r`n`a**.

(...)

These escape sequences stand for special characters. The most common ones are **\t** (tab), **\r** (carriage return), and **\n** (linefeed). In AutoHotkey, an accent (`) may optionally be used in place of the backslash in these cases. Escape sequences in the form **\xhh** are also supported, in which **hh** is the hex code of any ANSI character between 00 and FF.

\t
\r
etc.

[v1.0.46.06+]: **\R** means "any single newline of any type", namely those listed at the **`a option** (however, **\R** inside a **character class** is merely the letter "R"). [v1.0.47.05+]: **\R** can

	<p>be restricted to CR, LF, and CRLF by specifying (*BSR_ANYCRLF) in uppercase at the beginning of the pattern (after the options); e.g. im)(*BSR_ANYCRLF)abc\Rxyz</p>
<p>\p{xx} \P{xx} \X</p>	<p>[AHK_L 61+]: Unicode character properties. Not supported on ANSI builds. \p{xx} matches a character with the xx property while \P{xx} matches any character <i>without</i> the xx property. For example, \pL matches any letter and \p{Lu} matches any uppercase letter. \X matches any number of characters that form an extended Unicode sequence.</p> <p>For a full list of supported property names and other details, search for "\p{xx}" at www.pcre.org/pcre.txt.</p>
<p>(*UCP)</p>	<p>[AHK_L 61+]: For performance, \d, \D, \s, \S, \w, \W, \b and \B recognize only ASCII characters by default, even on Unicode builds. If the pattern begins with (*UCP), Unicode properties will be used to determine which characters match. For example, \w becomes equivalent to [\p{L}\p{N}_] and \d becomes equivalent to \p{Nd}.</p>

Greed: By default, *****, **?**, **+**, and **{min,max}** are greedy because they consume all characters up through the *last* possible one that still satisfies the entire pattern. To instead have them stop at the *first* possible character, follow them with a question mark. For example, the pattern **<.+>** (which lacks a question mark) means: "search for a **<**, followed by one or more of any character, followed by a **>**". To stop this pattern from matching the *entire* string **text**, append a question mark to the plus sign: **<.+?>**. This causes the match to stop at the first **'>'** and thus it matches only the first tag ****.

Look-ahead and look-behind assertions: The groups **(?=...)**, **(?!...)**, **(?<=...)**, and **(?<!...)** are called *assertions* because they demand a condition to be met but don't consume any characters. For example, **abc(?!.*xyz)** is a look-ahead assertion that requires the string xyz to exist somewhere to the right of the string

abc (if it doesn't, the entire pattern is not considered a match). `(?=...)` is called a *positive* look-ahead because it requires that the specified pattern exist. Conversely, `(?!...)` is a *negative* look-ahead because it requires that the specified pattern *not* exist. Similarly, `(?<=...)` and `(?<!...)` are positive and negative look-*behinds* (respectively) because they look to the *left* of the current position rather than the right. Look-behinds are more limited than look-aheads because they do not support quantifiers of varying size such as `*`, `?`, and `+`. The escape sequence `\K` is similar to a look-behind assertion because it causes any previously-matched characters to be omitted from the final matched string. For example, `foo\Kbar` matches "foobar" but reports that it has matched "bar".

Related: Regular expressions are supported by [RegexMatch\(\)](#), [RegexReplace\(\)](#), and [SetTitleMatchMode](#).

Final note: Although this page touches upon most of the commonly-used RegEx features, there are quite a few other features you may want to explore such as conditional subpatterns. The complete PCRE manual is at www.pcre.org/pcre.txt

RegExMatch() [v1.0.45+]

Determines whether a string contains a pattern (regular expression).

```
FoundPos := RegExMatch(Haystack, NeedleRegEx [,  
OutputVar, StartingPosition := 1])
```

Parameters

Haystack

The string whose content is searched.

NeedleRegEx

The pattern to search for, which is a Perl-compatible regular expression (PCRE). The pattern's [options](#) (if any) must be included at the beginning of the string followed by a close-parenthesis. For example, the pattern "**i**abc.*123" would turn on the case-insensitive option and search for "abc", followed by zero or more occurrences of any character, followed by "123". If there are no options, the ")" is optional; for example, ")abc" is equivalent to "abc".

OutputVar

Mode 1 (default): Specify a variable in which to store the part of *Haystack* that matched the entire pattern. If the pattern is not found (that is, if the function returns 0), this variable and all array elements below are made blank.

If any [capturing subpatterns](#) are present inside *NeedleRegEx*, their matches are stored in a [pseudo-array](#) whose base name is *OutputVar*. For example, if the variable's name is *Match*, the substring that matches the first subpattern would be stored in *Match1*, the second would be stored in *Match2*, and so on. The exception to this is [named subpatterns](#): they are stored by name instead of number. For example, the substring that matches the named subpattern "(?P<Year>\d{4})" would be stored in *MatchYear*. If a particular subpattern does not match anything (or if the

function returns zero), the corresponding variable is made blank.

Within a [function](#), to create a pseudo-array that is global instead of local, [declare](#) the base name of the pseudo-array (e.g. `Match`) as a global variable prior to using it. The converse is true for [assume-global](#) functions. However, it is often also necessary to declare each element, due to a [common source of confusion](#).

Mode 2 (position-and-length): If a capital P is present in the RegEx's options -- such as "**P**abc.*123" -- the *length* of the entire-pattern match is stored in *OutputVar* (or 0 if no match). If any [capturing subpatterns](#) are present, their positions and lengths are stored in two [pseudo-arrays](#): *OutputVarPos* and *OutputVarLen*. For example, if the variable's base name is *Match*, the one-based *position* of the first subpattern's match would be stored in *MatchPos1*, and its length in *MatchLen1* (zero is stored in both if the subpattern was not matched or the function returns 0). The exception to this is [named subpatterns](#): they are stored by name instead of number (e.g. *MatchPosYear* and *MatchLenYear*).

Mode 3 (match object) [v1.1.05+]: If a capital O is present in the RegEx's options -- such as "**O**abc.*123" -- a [match object](#) is stored in *OutputVar*. This object can be used to retrieve the position, length and value of the overall match and of each [captured subpattern](#), if present.

StartingPosition

If *StartingPosition* is omitted, it defaults to 1 (the beginning of *Haystack*). Otherwise, specify 2 to start at the second character, 3 to start at the third, and so on. If *StartingPosition* is beyond the length of *Haystack*, the search starts at the empty string that lies at the end of *Haystack* (which typically results in no match).

If *StartingPosition* is less than 1, it is considered to be an offset from the end of *Haystack*. For example, 0 starts at the last character and -1 starts at the next-to-last character. If *StartingPosition* tries to go beyond the left end of *Haystack*, all of *Haystack* is searched.

Regardless of the value of *StartingPosition*, the return value is always relative to the first character of *Haystack*. For example, the position of "abc" in "123abc789" is always 4.

Return Value

This function returns the position of the leftmost occurrence of *NeedleRegEx* in the string *Haystack*. Position 1 is the first character. Zero is returned if the pattern is not found. If an error occurs (such as a syntax error inside *NeedleRegEx*), an empty string is returned and `ErrorLevel` is set to one of the values [below](#) instead of 0.

ErrorLevel

[v1.1.04+]: This function is able to throw an exception on failure (not to be confused with "no match found"). For more information, see [Runtime Errors](#).

`ErrorLevel` is set to one of the following:

- 0, which means that no error occurred.
- A string in the following form: *Compile error N at offset M: description*. In that string, *N* is the PCRE error number, *M* is the position of the offending character inside the regular expression, and *description* is the text describing the error.
- A negative number, which means an error occurred during the *execution* of the regular expression. Although such errors are rare, the ones most likely to occur are "too many possible empty-string matches" (-22), "recursion too deep" (-21), and "reached match limit" (-8). If these happen, try to redesign the pattern to be more restrictive, such as replacing each `*` with a `?`, `+`, or a limit like `{0,3}` wherever feasible.

Options

See [Options](#) for modifiers such as `"i)abc"`, which turns off case-sensitivity in the pattern `"abc"`.

Match Object [v1.1.05+]

If a capital O is present in the RegEx's options, a match object is stored in *OutputVar*. This object has the following properties:

Match.Pos(N): Returns the position of the overall match or a captured

subpattern.

Match.Len(N): Returns the length of the overall match or a captured subpattern.

Match.Value(N): Returns the overall match or a captured subpattern.

Match.Name(N): Returns the name of the given subpattern, if it has one.

Match.Count(): Returns the overall number of subpatterns.

Match.Mark(): Returns the *NAME* of the last encountered `(*MARK : NAME)`, when applicable.

Match[N]: If *N* is 0 or a valid subpattern number or name, this is equivalent to `Match.Value(N)`. Otherwise, *N* can be the name of one of the above properties. For example, `Match["Pos"]` and `Match.POS` are equivalent to `Match.Pos()` unless a subpattern named "Pos" exists, in which case they are equivalent to `Match.Value("Pos")`.

Match.N: Same as above, except that *N* is an unquoted name or number.

For all of the above properties, *N* can be any of the following:

- 0 for the overall match.
- The number of a subpattern, even one that also has a name.
- The name of a subpattern.

Brackets [] may be used in place of parentheses () if *N* is specified.

Performance

To search for a simple substring inside a larger string, use `InStr()` because it is faster than `RegexMatch()`.

To improve performance, the 100 most recently used regular expressions are kept cached in memory (in compiled form).

The **study option (S)** can sometimes improve the performance of a regular expression that is used many times (such as in a loop).

Remarks

A subpattern may be given a name such as the word *Year* in the pattern "(?P<Year>\d{4})". Such names may consist of up to 32 alphanumeric characters and underscores. The following limitation does not apply to the "O" (match object) mode: Although named subpatterns are also available by their numbers during the RegEx operation itself (e.g. \1 is a backreference to the string that actually matched the first capturing subpattern), they are stored in the [output pseudo-array](#) *only* by name (not by number). For example, if "Year" is the first subpattern, *OutputVarYear* would be set to the matching substring, but *OutputVar1* would not be changed at all (it would retain its previous value, if any). However, if an [unnamed subpattern](#) occurs after "Year", it would be stored in *OutputVar2*, not *OutputVar1*.

Most characters like abc123 can be used literally inside a regular expression. However, the characters `\.*?+[{|^$` must be preceded by a backslash to be seen as literal. For example, `\.` is a literal period and `\\` is a literal backslash. Escaping can be avoided by using `\Q...\E`. For example: `\QLiteral Text\E`.

Within a regular expression, special characters such as tab and newline can be escaped with either an accent (```) or a backslash (`\`). For example, ``t` is the same as `\t` except when the `x` option is used.

To learn the basics of regular expressions (or refresh your memory of pattern syntax), see the [RegEx Quick Reference](#).

AutoHotkey's regular expressions are implemented using Perl-compatible Regular Expressions (PCRE) from www.pcre.org.

[AHK_L 31+]: Within an [expression](#), the `a ~= b` can be used as shorthand for `RegExMatch(a, b)`.

Related

[RegExReplace\(\)](#), [RegEx Quick Reference](#), [Regular Expression Callouts](#), [InStr\(\)](#), [IfInString](#), [StringGetPos](#), [SubStr\(\)](#), [SetTitleMatchMode RegEx](#), [Global matching and Grep \(forum link\)](#)

Common sources of text data: [FileRead](#), [UrlDownloadToFile](#), [Clipboard](#), [GUI](#)

Edit controls

Examples

```
FoundPos := RegExMatch("xxxabc123xyz", "abc.*xyz")  
; Returns 4, which is the position where the match  
was found.
```

```
FoundPos := RegExMatch("abc123123", "123$") ;  
Returns 7 because the $ requires the match to be  
at the end.
```

```
FoundPos := RegExMatch("abc123", "i)^ABC") ;  
Returns 1 because a match was achieved via the  
case-insensitive option.
```

```
FoundPos := RegExMatch("abcXYZ123", "abc(.*)123",  
SubPat) ; Returns 1 and stores "XYZ" in SubPat1.
```

```
FoundPos := RegExMatch("abc123abc456", "abc\d+",  
"", 2) ; Returns 7 instead of 1 due to  
StartingPosition 2 vs. 1.
```

```
; For general RegEx examples, see the RegEx Quick  
Reference.
```

RegexReplace() [v1.0.45+]

Replaces occurrences of a pattern (regular expression) inside a string.

```
NewStr := RegexReplace(Haystack, NeedleRegex [,  
Replacement := "", OutputVarCount := "", Limit :=  
-1, StartingPosition := 1])
```

Parameters

Haystack

The string whose content is searched and replaced.

NeedleRegex

The pattern to search for, which is a Perl-compatible regular expression (PCRE). The pattern's [options](#) (if any) must be included at the beginning of the string followed by a close-parenthesis. For example, the pattern `"i)abc.*123"` would turn on the case-insensitive option and search for "abc", followed by zero or more occurrences of any character, followed by "123". If there are no options, the ")" is optional; for example, `")abc"` is equivalent to `"abc"`.

Replacement

The string to be substituted for each match, which is plain text (not a regular expression). It may include backreferences like `$1`, which brings in the substring from *Haystack* that matched the first [subpattern](#). The simplest backreferences are `$0` through `$9`, where `$0` is the substring that matched the entire pattern, `$1` is the substring that matched the first subpattern, `$2` is the second, and so on. For backreferences above 9 (and optionally those below 9), enclose the number in braces; e.g. `${10}`, `${11}`, and so on. For [named subpatterns](#), enclose the name in braces; e.g. `${SubpatternName}`. To specify a literal `$`, use `$$` (this is the only character that needs such special treatment; backslashes are never needed to escape anything).

To convert the case of a subpattern, follow the \$ with one of the following characters: U or u (uppercase), L or l (lowercase), T or t (title case, in which the first letter of each word is capitalized but all others are made lowercase). For example, both \$U1 and \$U{1} transcribe an uppercase version of the first subpattern.

Nonexistent backreferences and those that did not match anything in *Haystack* -- such as one of the subpatterns in "(abc)|(xyz)" -- are transcribed as empty strings.

OutputVarCount

Specify a variable in which to store the number of replacements that occurred (0 if none).

Limit

If *Limit* is omitted, it defaults to -1, which replaces **all** occurrences of the pattern found in *Haystack*. Otherwise, specify the maximum number of replacements to allow. The part of *Haystack* to the right of the last replacement is left unchanged.

StartingPosition

If *StartingPosition* is omitted, it defaults to 1 (the beginning of *Haystack*). Otherwise, specify 2 to start at the second character, 3 to start at the third, and so on. If *StartingPosition* is beyond the length of *Haystack*, the search starts at the empty string that lies at the end of *Haystack* (which typically results in no replacements).

If *StartingPosition* is less than 1, it is considered to be an offset from the end of *Haystack*. For example, 0 starts at the last character and -1 starts at the next-to-last character. If *StartingPosition* tries to go beyond the left end of *Haystack*, all of *Haystack* is searched.

Regardless of the value of *StartingPosition*, the return value is always a complete copy of *Haystack* -- the only difference is that more of its left side might be unaltered compared to what would have happened with a *StartingPosition* of 1.

Return Value

This function returns a version of *Haystack* whose contents have been replaced by the operation. If no replacements are needed, *Haystack* is returned unaltered. If an error occurs (such as a syntax error inside *NeedleRegex*), *Haystack* is returned unaltered (except in versions prior to 1.0.46.06, which return "") and `ErrorLevel` is set to one of the values [below](#) instead of 0.

ErrorLevel

[v1.1.04+]: This function is able to throw an exception on failure (not to be confused with "no match found"). For more information, see [Runtime Errors](#).

`ErrorLevel` is set to one of the following:

- 0, which means that no error occurred.
- A string in the following form: *Compile error N at offset M: description*. In that string, *N* is the PCRE error number, *M* is the position of the offending character inside the regular expression, and *description* is the text describing the error.
- A negative number, which means an error occurred during the *execution* of the regular expression. Although such errors are rare, the ones most likely to occur are "too many possible empty-string matches" (-22), "recursion too deep" (-21), and "reached match limit" (-8). If these happen, try to redesign the pattern to be more restrictive, such as replacing each `*` with a `?`, `+`, or a limit like `{0,3}` wherever feasible.

Options

See [Options](#) for modifiers such as `"i)abc"`, which turns off case-sensitivity in the pattern `"abc"`.

Performance

To replace simple substrings, use [StringReplace](#) because it is faster than `RegexReplace()`.

If you know what the maximum number of replacements will be, specifying that for the *Limit* parameter improves performance because the search can be stopped early (this might also reduce the memory load on the system during the operation). For example, if you know there can be only one match near the

beginning of a large string, specify a limit of 1.

To improve performance, the 100 most recently used regular expressions are kept cached in memory (in compiled form).

The [study option \(S\)](#) can sometimes improve the performance of a regular expression that is used many times (such as in a loop).

Remarks

Most characters like abc123 can be used literally inside a regular expression. However, the characters `\.*?+[{|^$` must be preceded by a backslash to be seen as literal. For example, `\.` is a literal period and `\\` is a literal backslash. Escaping can be avoided by using `\Q...\E`. For example: `\QLiteral Text\E`.

Within a regular expression, special characters such as tab and newline can be escaped with either an accent (`^`) or a backslash (`\`). For example, `^t` is the same as `\t`.

To learn the basics of regular expressions (or refresh your memory of pattern syntax), see the [RegEx Quick Reference](#).

Related

[RegExMatch\(\)](#), [RegEx Quick Reference](#), [Regular Expression Callouts](#), [StringReplace](#), [InStr\(\)](#)

Common sources of text data: [FileRead](#), [UrlDownloadToFile](#), [Clipboard](#), [GUI Edit controls](#)

Examples

```
NewStr := RegExReplace("abc123123", "123$", "xyz")  
; Returns "abc123xyz" because the $ allows a match  
only at the end.  
NewStr := RegExReplace("abc123", "i)^ABC") ;  
Returns "123" because a match was achieved via the  
case-insensitive option.
```

```
NewStr := RegExReplace("abcXYZ123", "abc(.*)123",  
"aaa$1zzz") ; Returns "aaaXYZzzz" by means of the  
$1 backreference.
```

```
NewStr := RegExReplace("abc123abc456", "abc\d+",  
"", ReplacementCount) ; Returns "" and stores 2  
in ReplacementCount.
```

; For general RegEx examples, see the [RegEx Quick Reference](#).

SetEnv (Var = Value)

Assigns the specified value to a [variable](#).

Deprecated: This command or a legacy assignment is not recommended for use in new scripts. Use [expression assignments](#) like `Var := Value` instead.

```
SetEnv, Var, Value  
Var = Value
```

Parameters

Var

The name of the [variable](#) in which to store *Value*.

Value

The string or number to store. If the string is long, it can be broken up into several shorter lines by means of a [continuation section](#), which might improve readability and maintainability.

Remarks

By default, any spaces or tabs at the beginning and end of *Value* are omitted from *Var*. To prevent this, use the methods described at [AutoTrim Off](#).

The name "SetEnv" is misleading and is a holdover from AutoIt v2. Unlike AutoIt v2, AutoHotkey does not store its variables in the environment. This is because performance would be worse and also because the OS limits the size of each environment variable to 32 KB. 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 a [pseudo-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%i% = 123`. See [Arrays](#) for more details.

Related

[AutoTrim](#), [EnvSet](#), [EnvAdd](#), [EnvSub](#), [EnvMult](#), [EnvDiv](#), [If](#), [Arrays](#)

Example

```
Var1 = This is a string.  
Color2 = 450  
Color3 = %Var1%  
Array%i% = %A_TickCount%
```

Sort

Arranges a variable's contents in alphabetical, numerical, or random order (optionally removing duplicates).

```
Sort, VarName [, Options]
```

Parameters

VarName

The name of the variable whose contents will be sorted. This cannot be an expression.

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). If both **C** and **CL** are omitted, the uppercase letters A-Z are considered identical to their lowercase counterparts for the purpose of the sort.

CL [v1.0.43.03+]: Case insensitive sort based on the current user's locale. For example, most English and Western European locales treat the letters A-Z and ANSI letters like Å and Ü as identical to their lowercase counterparts. This method also uses a "word sort", which treats hyphens and apostrophes in such a way that words like "coop" and "co-op" stay together. Depending on the content of the items being sorted, the performance will be 1 to 8 times worse than the default method of insensitivity.

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`), which correctly sorts *VarName* if its lines end in either LF (``n`) or CR+LF

(`r`n).

F MyFunction [v1.0.47+]: Uses custom sorting according to the criteria in *MyFunction* (though sorting takes much longer). Specify the letter "F" followed by optional spaces/tabs followed by the name of a [function](#) to be used for comparing any two items in the list. The function must accept two or three parameters. When the function deems the first parameter to be greater than the second, it should return a positive integer; when it deems the two parameters to be equal, it should return 0, "", or nothing; otherwise, it should return a negative integer. If a decimal point is present in the returned value, that part is ignored (i.e. 0.8 is the same as 0). If present, the third parameter receives the offset (in characters) of the second item from the first as seen in the original/unsorted list (see examples). Finally, the function uses the same global settings (e.g. [StringCaseSense](#)) as the Sort command that called it.

Note: The **F** option causes all other options except **D**, **Z**, and **U** to be ignored (though **N**, **C**, and **CL** still affect how [duplicates](#) are detected). Also, sorting does not occur when the specified function: 1) does not exist; 2) accepts fewer than two parameters; or 3) the first or second parameter is [ByRef](#).

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 do not start with a number are considered to be zero for the purpose of the sort. Numbers are treated as 64-bit floating point values so that the decimal portion of each number (if any) is taken into account.

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**, **Z**, and **U** to be ignored (though **N**, **C**, and **CL** still affect how duplicates are detected). Examples:

```
Sort, MyVar, Random  
Sort, MyVar, Random Z D |
```

U: Removes duplicate items from the list so that every item is unique.

ErrorLevel is set to the number of items removed (0 if none). If the **C** option is in effect, the case of items must match for them to be considered identical. If the **N** option is in effect, an item such as 2 would be considered a duplicate of 2.0. If either the **Pn** or **** (backslash) option is in effect, the entire item must be a duplicate, not just the substring that is used for sorting. If the **Random** or **F/Function** option is in effect, duplicates are removed only if they appear adjacent to each other as a result of the sort. For example, when "A|B|A" is sorted randomly, the result could contain either one or two A's.

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 **Z**, the last `n (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).

\: 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), such as the example below, in which the AAA.txt line is sorted above the BBB.txt line because their directories are ignored for the purpose of the sort:

```
C:\BBB\AAA.txt  
C:\AAA\BBB.txt
```

Note: Options **N** and **P** are ignored when the backslash option is present.

Remarks

This command is typically used to sort a variable that contains a list of lines, with each line ending in a linefeed character (`\n`). One way to get a list of lines into a variable is to load an entire file via [FileRead](#).

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.

[ErrorLevel](#) is changed by this command only when the **U** option is in effect.

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

[FileRead](#), [file-reading loop](#), [parsing loop](#), [StringSplit](#), [RegisterCallback\(\)](#), [clipboard](#), [#MaxMem](#)

Examples

```
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

; The following example sorts the contents of a
file:
FileRead, Contents, C:\Address List.txt
if not ErrorLevel ; Successfully loaded.
{
    Sort, Contents
    FileDelete, C:\Address List (alphabetical).txt
```

```

    FileAppend, %Contents%, C:\Address List
(alphabetical).txt
    Contents = ; Free the memory.
}

; The following 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
    return
Sort Clipboard
MsgBox Ready to be pasted:`n%Clipboard%
return

; The following examples demonstrate custom
sorting via a callback function.
MyVar = def`nabc`nmno`nFGH`nco-
op`ncoop`ncop`ncon`n
Sort, MyVar, F StringSort
StringSort(a1, a2)
{
    return a1 > a2 ? 1 : a1 < a2 ? -1 : 0 ; Sorts
alphabetically based on the setting of
StringCaseSense.
}

MyVar = 5,3,7,9,1,13,999,-4
Sort, MyVar, F IntegerSort D,
IntegerSort(a1, a2)
{
    return a1 - a2 ; Sorts in ascending numeric
order. This method works only if the difference

```

is never so large as to overflow a signed 64-bit integer.

}

MyVar = 1,2,3,4

Sort, MyVar, F ReverseDirection D, ; Reverses the list so that it contains 4,3,2,1

ReverseDirection(a1, a2, offset)

{

 return offset ; Offset is positive if a2 came after a1 in the original list; negative otherwise.

}

StringCaseSense

Determines whether string comparisons are case sensitive (default is "not case sensitive").

`StringCaseSense, On|Off|Locale`

Parameters

On|Off|Locale

On: String comparisons are case sensitive. This setting also makes the [expression equal sign operator \(=\)](#) and the case-insensitive mode of [InStr\(\)](#) use the *locale* method described below.

Off (starting default): The letters A-Z are considered identical to their lowercase counterparts. This is the starting default for all scripts due to backward compatibility and performance (*Locale* is 1 to 8 times slower than *Off* depending on the nature of the strings being compared).

Locale [v1.0.43.03+]: String comparisons are case **insensitive** according to the rules of the current user's locale. For example, most English and Western European locales treat not only the letters A-Z as identical to their lowercase counterparts, but also ANSI letters like Ä and Ü as identical to theirs.

Remarks

This setting applies to:

- [Expression comparison operators](#) (except ==). However, since the [equal-sign operator \(=\)](#) is always case-insensitive, it uses the *Locale* mode when *StringCaseSense* is *On*, as does the case-insensitive mode of [InStr\(\)](#).
- [IfInString](#), [StringGetPos](#), and [InStr\(\)](#). However, [InStr\(\)](#) is not affected when its *CaseSensitive* parameter is *true*.
- [StringReplace](#)
- [if var in/contains MatchList](#), [if var between](#), and [IfEqual and its family](#).

- [AHK_L 42+]: [if var is \[not\] type](#) respects the system locale only if the *Locale* mode is in effect.

The built-in variable **A_StringCaseSense** contains the current setting (the word On, Off, or Locale).

Every newly launched [thread](#) (such as a [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 Locale
```

StringGetPos

Retrieves the position of the specified substring within a string.

Deprecated: This command is not recommended for use in new scripts. Use the [InStr](#) function instead.

```
StringGetPos, OutputVar, InputVar, SearchText [,  
L#|R#, Offset]
```

Parameters

OutputVar

The name of the variable in which to store the retrieved position relative to the first character of *InputVar*. Position 0 is the first character for `StringGetPos` and position 1 is the first character for `InStr()`.

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.

SearchText

The string to search for. Matching is not case sensitive unless [StringCaseSense](#) has been turned on.

L#|R#

This affects which occurrence will be found if *SearchText* 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.

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.

Offset

The number of characters on the leftmost or rightmost side (depending on the parameter above) to skip over. If omitted, the default is 0. For example, the following would start searching at the tenth character from the left: `StringGetPos, OutputVar, InputVar, abc, , 9`. This parameter can be an [expression](#).

ErrorLevel

[ErrorLevel](#) is set to 1 if the specified occurrence of *SearchText* could not be found within *InputVar*, or 0 otherwise.

Remarks

Unlike [StringMid](#) and [InStr\(\)](#), 0 is defined as the position of the first character for `StringGetPos`.

The retrieved position is always relative to the first character of *InputVar*, regardless of whether *L#|R#* and/or *Offset* are specified. For example, if the string "abc" is found in 123abc789, its reported position will always be 3 regardless of the method by which it was found.

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. They are useful when searching for spaces and tabs alone or at the beginning or end of *SearchText*.

Related

[InStr\(\)](#), [RegExMatch\(\)](#), [IfInString](#), [if var in/contains MatchList](#),

StringCaseSense, StringReplace, SplitPath, StringLeft, StringRight, StringMid, StringTrimLeft, StringTrimRight, StringLen, StringLower, StringUpper, 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 += 1 ; 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
            break
    }
```

```
length := pos_prev - pos - 1
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%
    }
}
```

StringLeft / StringRight

Retrieves a number of characters from the left or right-hand side of a string.

Deprecated: These commands are not recommended for use in new scripts. Use the [SubStr](#) function instead.

```
StringLeft, OutputVar, InputVar, Count  
StringRight, OutputVar, InputVar, Count
```

Parameters

OutputVar

The name of the variable in which to store the substring extracted from *InputVar*.

InputVar

The name of the variable whose contents will be extracted from. Do not enclose the name in percent signs unless you want the *contents* of the variable to be used as the name.

Count

The number of characters to extract, which can be an [expression](#). If *Count* is less than or equal to zero, *OutputVar* will be made empty (blank). If *Count* exceeds the length of *InputVar*, *OutputVar* will be set equal to the entirety of *InputVar*.

Remarks

For this and all other commands, *OutputVar* is allowed to be the same variable as an *InputVar*.

Related

SubStr(), StringMid, StringTrimLeft, StringTrimRight, IfInString, StringGetPos, StringLen, StringLower, StringUpper, StringReplace

Example

```
String = This is a test.  
StringLeft, OutputVar, String, 4 ; Stores the  
string "This" in OutputVar.  
StringRight, OutputVar, String, 5 ; Stores the  
string "test." in OutputVar.
```

StrLen() / StringLen

Retrieves the count of how many characters are in a string.

StrLen()

```
Length := StrLen(String)
```

Parameters

String

- The string to be measured.

Return Value

This function returns the length of the string.

Examples

```
InputVar := "The Quick Brown Fox Jumps Over the  
Lazy Dog"  
MsgBox % "The length of InputVar is " .  
StrLen(InputVar) ; Result: 43
```

StringLen

Deprecated: This command is not recommended for use in new scripts. Use the [StrLen](#) function instead.

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 *contents* of the variable to be used as the name.

Examples

```
StringLen, Length, InputVar  
MsgBox, The length of InputVar is %Length%.
```

Remarks

If *String* or *InputVar* is a variable to which [ClipboardAll](#) was previously assigned, `StrLen()` or `StringLen` will report its total size.

Related

[IfInString](#), [StringGetPos](#), [StringMid](#), [StringTrimLeft](#), [StringTrimRight](#), [StringLeft](#), [StringRight](#), [StringLower](#), [StringUpper](#), [StringReplace](#)

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 *contents* 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*.

[v1.1.20+]: [Format\(\)](#) can also be used for case conversions, as shown below:

```
MsgBox % Format("{:U}, {:L} and {:T}", "upper",  
"LOWER", "title")
```

Related

[Format\(\)](#), [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 one or more characters from the specified position in a string.

Deprecated: This command is not recommended for use in new scripts. Use the [SubStr](#) function instead.

```
StringMid, OutputVar, InputVar, StartChar [, Count, L]
```

Parameters

OutputVar

The name of the variable in which to store the substring extracted from *InputVar*.

InputVar

The name of the variable from whose contents the substring will be extracted. Do not enclose the name in percent signs unless you want the *contents* of the variable to be used as the name.

StartChar

The position of the first character to be extracted, which can be an [expression](#). Unlike [StringGetPos](#), 1 is the first character. If *StartChar* is less than 1, it will be assumed to be 1. If *StartChar* is beyond the end of the string, *OutputVar* is made empty (blank).

Count

[v1.0.43.10+]: This parameter may be omitted or left blank, which is the same as specifying an integer large enough to retrieve all characters from the string.

Otherwise, specify the number of characters to extract, which can be an [expression](#). If *Count* is less than or equal to zero, *OutputVar* will be made

empty (blank). If *Count* exceeds the length of *InputVar* measured from *StartChar*, *OutputVar* will be set equal to the entirety of *InputVar* starting at *StartChar*.

L

The letter L can be used to extract characters that lie to the left of *StartChar* rather than to the right. In the following example, *OutputVar* will be set to *Red*:

```
InputVar = The Red Fox  
StringMid, OutputVar, InputVar, 7, 3, L
```

If the L option is present and *StartChar* is less than 1, *OutputVar* will be made blank. If *StartChar* is beyond the length of *InputVar*, only those characters within reach of *Count* will be extracted. For example, the below will set *OutputVar* to *Fox*:

```
InputVar = The Red Fox  
StringMid, OutputVar, InputVar, 14, 6, L
```

Remarks

For this and all other commands, *OutputVar* is allowed to be the same variable as *InputVar*.

Related

[SubStr\(\)](#), [StringLeft](#), [StringRight](#), [StringTrimLeft](#), [StringTrimRight](#), [IfInString](#), [StringGetPos](#), [StringLen](#), [StringLower](#), [StringUpper](#), [StringReplace](#)

Example

```
Source = Hello this is a test.  
StringMid, the_word_this, Source, 7, 4
```

StrReplace() / StringReplace

Replaces the specified substring with a new string.

StrReplace() [v1.1.21+]

```
ReplacedStr := StrReplace(Haystack, SearchText [,  
ReplaceText, OutputVarCount, Limit := -1])
```

Parameters

Haystack

The string whose content is searched and replaced.

SearchText

The string to search for. Matching is not case sensitive unless [StringCaseSense](#) has been turned on.

ReplaceText

SearchText will be replaced with this text. If omitted or blank, *SearchText* will be replaced with blank (empty). In other words, it will be omitted from *OutputVar*.

OutputVarCount

Specify a variable in which to store the number of replacements that occurred (0 if none).

Limit

If *Limit* is omitted, it defaults to -1, which replaces **all** occurrences of the pattern found in *Haystack*. Otherwise, specify the maximum number of replacements to allow. The part of *Haystack* to the right of the last replacement is left unchanged.

Return Value

This function returns a version of *Haystack* whose contents have been replaced by the operation. If no replacements are needed, *Haystack* is returned unaltered.

Examples

```
; Remove all CR+LF's from the clipboard contents:
Clipboard := StrReplace(Clipboard, "`r`n")

; Replace all spaces with pluses:
NewStr := StrReplace(OldStr, A_Space, "+")

; Remove all blank lines from the text in a
variable:
Loop
{
    MyString := StrReplace(MyString, "`r`n`r`n",
"`r`n", Count)
    if Count = 0 ; No more replacements needed.
        break
}
```

StringReplace

Deprecated: This command is not recommended for use in new scripts. Use the [StrReplace](#) function instead.

```
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 *contents* of the variable to be used as the name.

SearchText

The string to search for. Matching is not case sensitive unless [StringCaseSense](#) has been turned on.

ReplaceText

SearchText will be replaced with this text. If omitted or blank, *SearchText* will be replaced with blank (empty). In other words, it will be omitted from *OutputVar*.

ReplaceAll?

If omitted, only the first occurrence of *SearchText* will be replaced. But if this parameter is 1, A, or All, all occurrences will be replaced.

Specify the word **UseErrorLevel** to store in ErrorLevel the number of occurrences replaced (0 if none). UseErrorLevel implies "All".

ErrorLevel

When the last parameter is *UseErrorLevel*, **ErrorLevel** is given the number occurrences replaced (0 if none). Otherwise, **ErrorLevel** is set to 1 if *SearchText* is not found within *InputVar*, or 0 if it is found.

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. They are useful when searching for spaces and tabs alone or at the beginning or end of *SearchText*.

[v1.0.45+]: The AllSlow option became obsolete due to improvements to performance and memory utilization. Although it may still be specified, it has no effect.

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

; Remove all blank lines from the text in a
variable:
Loop
{
    StringReplace, MyString, MyString, `r`n`r`n,
`r`n, UseErrorLevel
    if ErrorLevel = 0 ; No more replacements
needed.
        break
}
```

Related

[RegexReplace\(\)](#), [IfInString](#), [StringCaseSense](#), [StringLeft](#), [StringRight](#), [StringMid](#), [StringTrimLeft](#), [StringTrimRight](#), [StringLen](#), [StringLower](#), [StringUpper](#), [StringGetPos](#), [if var is type](#)

StrSplit() / StringSplit

Separates a string into an array of substrings using the specified delimiters.

StrSplit() [v1.1.13+]

```
Array := StrSplit(String [, Delimiters, OmitChars])  
Array := StrSplit(String [, Delimiters, OmitChars,  
MaxParts := -1]) ; [v1.1.28+]
```

Parameters

String

A string to split.

Delimiters

If this parameter is blank or omitted, each character of the input string will be treated as a separate substring.

Otherwise, *Delimiters* can be either a single string or an array of strings, each of which is used to determine where the boundaries between substrings occur. Since the delimiters are not considered to be part of the substrings themselves, they are never included in the returned array. Also, if there is nothing between a pair of delimiters within the input string, the corresponding array element will be blank.

For example: `" , "` 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 the input string.

OmitChars

An optional list of characters (case sensitive) to exclude from the beginning and end of each array element. For example, if *OmitChars* is `" `t "`, spaces and tabs will be removed from the beginning and end (but not the middle) of every element.

If *Delimiters* is blank, *OmitChars* indicates which characters should be excluded from the array.

MaxParts [v1.1.28+]

The maximum number of substrings to return. If non-zero, the string is split a maximum of *MaxParts*-1 times and the remainder of the string is returned in the last substring (excluding any leading or trailing *OmitChars*). If omitted or -1, there is no limit.

Return Value

This function returns an [array](#) (object) of strings.

Examples

```
TestString := "This is a test."
word_array := StrSplit(TestString, A_Space, ".")
; Omits periods.
MsgBox % "The 4th word is " word_array[4]

Colors := "red,green,blue"
ColorArray := StrSplit(Colors, ",")
Loop % ColorArray.MaxIndex()
{
    this_color := ColorArray[A_Index]
    MsgBox, Color number %A_Index% is
    %this_color%.
}
```

StringSplit

Deprecated: This command is not recommended for use in new scripts. Use the [StrSplit](#) function instead.

```
StringSplit, OutputArray, InputVar [, Delimiters, OmitChars]
```

Parameters

OutputArray

The name of the [pseudo-array](#) in which to store each substring extracted from *InputVar*. For example, if *MyArray* is specified, the command will put the number of substrings produced (0 if none) into *MyArray0*, the first substring into *MyArray1*, the second into *MyArray2*, and so on.

Within a [function](#), to create a pseudo-array that is global instead of local, [declare](#) *MyArray0* as a global variable inside the function (the converse is true for [assume-global](#) functions). However, it is often also necessary to declare each element, due to a [common source of confusion](#). For more details, see [Functions](#).

InputVar

The name of the variable whose contents will be analyzed. Do not enclose the name in percent signs unless you want the *contents* of the variable to be used as the name.

Note: *InputVar* must not be one of the variables in *OutputArray*.

Delimiters

If this parameter is blank or omitted, each character of *InputVar* will be treated as a separate substring.

Otherwise, *Delimiters* contains one or more characters (case sensitive), each of which is used to determine where the boundaries between substrings occur in *InputVar*. Since the delimiter characters are not considered to be part of the substrings themselves, they are never copied into *OutputArray*. Also, if there is nothing between a pair of delimiters within *InputVar*, the corresponding array element will be blank.

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 *InputVar*.

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:

```
StringReplace, NewHTML, HTMLString, <br>,
` , All ; Replace each <br> with an
accent.
StringSplit, MyArray, NewHTML, ` ` ; Split
the string based on the accent character.
```

OmitChars

An optional list of characters (case sensitive) to exclude from the beginning and end of each array element. For example, if *OmitChars* is `%A_Space%%A_Tab%`, spaces and tabs will be removed from the beginning and end (but not the middle) of every element.

If *Delimiters* is blank, *OmitChars* indicates which characters should be excluded from the array.

Unlike the last parameter of most other commands, commas in *OmitChars* must be escaped (`` ,`).

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, it is safest to use the zero element (*MyArray0*) to determine how many items were actually produced by the command.

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%
{
    this_color := ColorArray%A_Index%
    MsgBox, Color number %A_Index% is
%this_color%.
}
```

Remarks

Whitespace characters such as spaces and tabs will be preserved unless those characters are themselves delimiters or included in *OmitChars*. Tabs and spaces can be trimmed from both ends of any variable by calling the [Trim](#) function. For example: `MyArray1 := Trim(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.

To arrange the fields in a different order prior to splitting them, use the [Sort](#) command.

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. For example:

```
Colors = red,green,blue
Loop, parse, Colors, `,
    MsgBox Color number %A_Index% is
%A_LoopField%.
```

Related

[Parsing loop](#), [Arrays](#), [Sort](#), [SplitPath](#), [IfInString](#), [StringGetPos](#), [StringMid](#), [StringTrimLeft](#), [StringTrimRight](#), [StringLen](#), [StringLower](#), [StringUpper](#), [StringReplace](#)

StringTrimLeft / StringTrimRight

Removes a number of characters from the left or right-hand side of a string.

Deprecated: These commands are not recommended for use in new scripts. Use the [SubStr](#) function instead.

```
StringTrimLeft, OutputVar, InputVar, Count  
StringTrimRight, OutputVar, InputVar, Count
```

Parameters

OutputVar

The name of the variable in which to store the shortened version of *InputVar*.

InputVar

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.

Count

The number of characters to remove, which can be an [expression](#). If *Count* is less than or equal to zero, *OutputVar* will be set equal to the entirety of *InputVar*. If *Count* exceeds the length of *InputVar*, *OutputVar* will be made empty (blank).

Remarks

For this and all other commands, *OutputVar* is allowed to be the same variable as an *InputVar*.

Related

SubStr(), StringMid, StringLeft, StringRight, IfInString, StringGetPos,
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.
```

SubStr() [v1.0.46+]

Retrieves one or more characters from the specified position in a string.

```
NewStr := SubStr(String, StartingPos [, Length])
```

Parameters

String

The string whose content is copied.

StartingPos

Specify 1 to start at the first character, 2 to start at the second, and so on (if *StartingPos* is beyond *String*'s length, an empty string is returned). If *StartingPos* is less than 1, it is considered to be an offset from the end of the string. For example, 0 extracts the last character and -1 extracts the two last characters (but if *StartingPos* tries to go beyond the left end of the string, the extraction starts at the first character).

Length

If this parameter is omitted, it defaults to "all characters". Otherwise, specify the maximum number of characters to retrieve (fewer than the maximum are retrieved whenever the remaining part of the string is too short). You can also specify a negative *Length* to omit that many characters from the end of the returned string (an empty string is returned if all or too many characters are omitted).

Return Value

This function returns the requested substring of *String*.

Remarks

Functionally, the **SubStr** function is almost the same as the [StringMid](#) command. However, it's recommended to use `SubStr`, because it is more flexible and future-

proofed than StringMid.

Related

[RegExMatch\(\)](#), [StringMid](#), [StringLeft/Right](#), [StringTrimLeft/Right](#)

Examples

```
; Example 1
MsgBox % SubStr("123abc789", 4, 3) ; Returns abc

; Example 2
String := "The Quick Brown Fox Jumps Over the Lazy
Dog"
MsgBox % SubStr(String, 1, 19) ; Returns "The
Quick Brown Fox"
MsgBox % SubStr(String, -7) ; Returns "Lazy Dog"
```

Trim [AHK_L 31+]

Trims characters from the beginning and/or end of a string.

```
Result := Trim(String, OmitChars := " `t")
Result := LTrim(String, OmitChars := " `t")
Result := RTrim(String, OmitChars := " `t")
```

Parameters

String

Any string value or variable. Numbers are not supported.

OmitChars

An optional list of characters (case sensitive) to exclude from the beginning and/or end of *String*. If omitted, spaces and tabs will be removed.

Examples

```
text := "  text  "
MsgBox % "No trim:`t '" text "'
    . "`nTrim:`t '" Trim(text) "'
    . "`nLTrim:`t '" LTrim(text) "'
    . "`nRTrim:`t '" RTrim(text) "'
MsgBox % LTrim("00000123", "0")
```

Control

Makes a variety of changes to a control.

```
Control, Cmd [, Value, Control, WinTitle, WinText,  
ExcludeTitle, ExcludeText]
```

Parameters

Cmd, Value

See list below.

Control

Can be either ClassNN (the classname and instance number of the control) or the control's text, both of which can be determined via Window Spy. When using text, the matching behavior is determined by [SetTitleMatchMode](#). If this parameter is blank, the target window's topmost control will be used.

To operate upon a control's HWND (window handle), leave the *Control* parameter blank and specify `ahk_id %ControlHWND%` for the *WinTitle* parameter (this also works on hidden controls even when [DetectHiddenWindows](#) is Off). The HWND of a control is typically retrieved via [ControlGet Hwnd](#), [MouseGetPos](#), or [DllCall](#).

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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.

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. If you additionally want to prevent a control's shortcut key (underlined letter) from working, disable the control via **Control Disable**.

Style, N or ExStyle, N: Changes the style or extended style of a control, respectively. If the first character of **N** is a plus or minus sign, the style(s) in **N** are added or removed, respectively. If the first character is a caret (^), the style(s) in **N** are each toggled to the opposite state. If the first character is a digit, the control's style is overwritten completely; that is, it becomes **N**. **ErrorLevel** is set to 1 if the target window/control is not found or the style is not allowed to be applied.

Certain style changes require that the entire window be redrawn using **WinSet Redraw**. Also, the [styles table](#) lists some of the style numbers. For example:

```
Control, Style, ^0x800000, Edit1, WinTitle ;  
Set the WS_BORDER style to its opposite state.
```

ShowDropDown: Drops a ComboBox so that its choices become visible.

HideDropDown: Reverses the above.

TabLeft [, Count] and **TabRight [, Count]:** Moves left or right by one or more tabs in a SysTabControl32. *Count* is assumed to be 1 if omitted or blank. To instead select a tab directly by number, replace the number 5 below with one less than the tab number you wish to select. In other words, 0 selects the first tab, 1 selects the second, and so on:

```
SendMessage, 0x1330, 5,, SysTabControl321,  
WinTitle ; 0x1330 is TCM_SETCURFOCUS.  
Sleep 0 ; This line and the next are necessary  
only for certain tab controls.  
SendMessage, 0x130C, 5,, SysTabControl321,  
WinTitle ; 0x130C is TCM_SETCURSEL.
```

Add, String: Adds *String* as a new entry at the bottom of a ListBox, ComboBox (and possibly other types).

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. To select or deselect all items in a *multi-select* listbox, follow this example:

```
PostMessage, 0x185, 1, -1, ListBox1, WinTitle ;  
Select all listbox items. 0x185 is LB_SETSEL.
```

ChooseString, String: Sets the selection (choice) in a ListBox or ComboBox to be the first entry whose leading part matches *String*. The search is not case sensitive. For example, if a ListBox/ComboBox contains the item "UNIX Text", specifying the word unix (lowercase) would be enough to select it.

EditPaste, String: Pastes *String* at the caret/insert position in an Edit control (this does not affect the contents of the [clipboard](#)).

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[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 (except for *Style* and *ExStyle*). That delay can be changed via [SetControlDelay](#).

To discover the ClassNN or HWND of the control that the mouse is currently hovering over, use [MouseGetPos](#).

Window titles and text are case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[SetControlDelay](#), [ControlGet](#), [GuiControl](#), [ControlGetText](#), [ControlSetText](#), [ControlMove](#), [ControlGetPos](#), [ControlClick](#), [ControlFocus](#), [ControlSend](#), [WinSet](#)

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 control's text, both of which can be determined via Window Spy. When using text, the matching behavior is determined by [SetTitleMatchMode](#). If this parameter is blank or omitted, the target window's topmost control will be used.

To operate upon a control's HWND (window handle), leave the *Control* parameter blank and specify `ahk_id %ControlHwnd%` for the *WinTitle* parameter (this also works on hidden controls even when [DetectHiddenWindows](#) is Off). The HWND of a control is typically retrieved via [ControlGet Hwnd](#), [MouseGetPos](#), or [DllCall](#).

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

Remarks

To be effective, the control's window generally must not be minimized or hidden.

To improve reliability, a delay is done automatically after every use of this command. That delay can be changed via [SetControlDelay](#).

To discover the ClassNN or HWND of the control that the mouse is currently hovering over, use [MouseGetPos](#).

Window titles and text are 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

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 *Cmd*.

Cmd, Value

See list below.

Control

Can be either ClassNN (the classname and instance number of the control) or the control's text, both of which can be determined via Window Spy. When using text, the matching behavior is determined by [SetTitleMatchMode](#). If this parameter is blank, the target window's topmost control will be used.

To operate upon a control's HWND (window handle), leave the *Control* parameter blank and specify `ahk_id %ControlHwnd%` for the *WinTitle* parameter (this also works on hidden controls even when [DetectHiddenWindows](#) is Off). The HWND of a control is typically retrieved via [ControlGet Hwnd](#), [MouseGetPos](#), or [DllCall](#).

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility).

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.

Cmd, Value

The *Cmd* and *Value* parameters are dependent upon each other and their usage is described below.

List: Retrieves a list of items from a ListView, ListBox, ComboBox, or DropDownList.

ListView: The syntax for ListView retrieval is:

```
ControlGet, OutputVar, List, Options,  
SysListView321, WinTitle, WinText
```

If the *Options* parameter is blank or omitted, all the text in the control is retrieved. Each row except the last will end with a linefeed character (`\n`). Within each row, each field (column) except the last will end with a tab character (`\t`).

Specify for *Options* zero or more of the following words, each separated from the next with a space or tab:

Selected: Retrieves only the selected (highlighted) rows rather than all rows. If none, *OutputVar* is made blank.

Focused: Retrieves only the focused row. If none, *OutputVar* is made blank.

Col4: Retrieves only the fourth column (field) rather than all columns (replace 4 with a number of your choice).

Count: Retrieves a single number that is the total number of rows in the control.

Count Selected: Retrieves the number of selected (highlighted) rows.

Count Focused: Retrieves the row number (position) of the focused row (0 if none).

Count Col: Retrieves the number of columns in the control (or -1 if the count

cannot be determined).

Note: Some applications store their ListView text privately, which prevents their text from being retrieved. In these cases, ErrorLevel will usually be set to 0 (indicating success) but all the retrieved fields will be empty. Also note that ListView text retrieval is not restricted by #MaxMem.

Upon success, ErrorLevel is set to 0. Upon failure, it is set to 1 and *OutputVar* is made blank. Failure occurs when: 1) the target window or control does not exist; 2) the target control is not of type SysListView32; 3) the process owning the ListView could not be opened, perhaps due to a lack of user permissions or because it is locked; 4) the *ColN* option specifies a nonexistent column.

To extract the individual rows and fields out of a ListView, use a [parsing loop](#) as in this example:

```
ControlGet, List, List, Selected,
SysListView321, WinTitle
Loop, Parse, List, `n ; Rows are delimited by
linefeeds (`n).
{
    RowNumber := A_Index
    Loop, Parse, A_LoopField, %A_Tab% ; Fields
(columns) in each row are delimited by tabs
(A_Tab).
        MsgBox Row #%RowNumber% Col #%A_Index%
is %A_LoopField%.
}
```

On a related note, the columns in a ListView can be resized via [SendMessage](#) as shown in this example:

```
SendMessage, 4126, 0, 80, SysListView321,
WinTitle ; 4126 is the message
LVM_SETCOLUMNWIDTH.
; In the above, 0 indicates the first column
```

```
(specify 1 for the second, 2 for the third, etc.) Also, 80 is the new width.  
; Replace 80 with -1 to autosize the column.  
Replace it with -2 to do the same but also take into account the header text width.
```

ListBox, ComboBox, DropDownList: All the text is retrieved from the control (that is, the ListView options above such as *Count* and *Selected* are not supported).

Each item except the last will be terminated by a linefeed character (`n). To access the items individually, use a [parsing loop](#) as in this example:

```
ControlGet, List, List,, ComboBox1, WinTitle  
Loop, Parse, List, `n  
    MsgBox Item number %A_Index% is  
    %A_LoopField%.
```

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. To instead discover how many tabs (pages) exist in a tab control, follow this example:

```
SendMessage, 0x1304,,, SysTabControl321,  
WinTitle ; 0x1304 is TCM_GETITEMCOUNT.  
TabCount = %ErrorLevel%
```

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<<32>>32 ; Convert UInt  
to Int to have -1 if there is no item selected.  
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 (text insertion 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`). If the specified line number is blank or does not exist, **ErrorLevel** is set to 1 and *OutputVar* is made blank.

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: Retrieves an 8-digit hexadecimal number representing the style of the control. See the [styles table](#) for a listing of some styles.

ExStyle: Retrieves an 8-digit hexadecimal number representing the extended style of the control.

Hwnd [v1.0.43.06+]: Retrieves the window handle (HWND) of the specified control. For example: `ControlGet, OutputVar, Hwnd,, Edit1, WinTitle`. A control's HWND is often used with [PostMessage](#), [SendMessage](#), and [DllCall](#). On a related note, a control's HWND can also be retrieved via [MouseGetPos](#). Finally, a control's HWND can be used directly as an [ahk_id WinTitle](#) (this also works on hidden controls even when [DetectHiddenWindows](#) is Off).

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

Upon success, [ErrorLevel](#) is set to 0. If a problem occurred -- such as a nonexistent window or control -- [ErrorLevel](#) is set to 1 and *OutputVar* is made blank.

Remarks

Unlike commands that change a control, `ControlGet` does not have an automatic delay; that is, [SetControlDelay](#) does not affect it.

To discover the ClassNN or HWND of the control that the mouse is currently hovering over, use [MouseGetPos](#). To retrieve a list of all controls in a window, use [WinGet ControlList](#).

Window titles and text are case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

Control, GuiControlGet, ControlMove, ControlGetText, ControlSetText, ControlGetPos, ControlClick, ControlFocus, ControlSend, WinGet

Example

```
ControlGet, OutputVar, Line, 1, Edit1, Some Window  
Title
```

```
ControlGet, WhichTab, Tab, , SysTabControl321,  
Some Window Title  
if ErrorLevel  
    MsgBox There was a problem.  
else  
    MsgBox Tab #%WhichTab% is active.
```

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

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[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.

Prior to [v1.1.19.03], if `ControlGetFocus` was executed repeatedly at a high frequency (i.e. every 500 ms or faster), it usually disrupted the user's ability to double-click. This has been fixed.

Window titles and text are 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
    MsgBox, The target window doesn't exist or
    none of its controls has input focus.
else
    MsgBox, Control with focus = %OutputVar%
```

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 (in pixels) of *Control's* upper left corner. These coordinates are relative to the target window's upper-left corner 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 *Control's* width and height (in pixels). If omitted, the corresponding values will not be stored.

Control

Can be either ClassNN (the classname and instance number of the control) or the control's text, both of which can be determined via Window Spy. When using text, the matching behavior is determined by [SetTitleMatchMode](#). If this parameter is blank, the target window's topmost control will be used.

To operate upon a control's HWND (window handle), leave the *Control* parameter blank and specify `ahk_id %ControlHwnd%` for the *WinTitle* parameter (this also works on hidden controls even when [DetectHiddenWindows](#) is Off). The HWND of a control is typically retrieved via [ControlGet Hwnd](#), [MouseGetPos](#), or [DllCall](#).

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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

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 [ClassNN](#) or [HWND](#) of the control that the mouse is currently hovering over, use [MouseGetPos](#). To retrieve a list of all controls in a window, use [WinGet ControlList](#).

Window titles and text are 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) or the control's text, both of which can be determined via Window Spy. When using text, the matching behavior is determined by [SetTitleMatchMode](#). If this parameter is blank or omitted, the target window's topmost control will be used.

To operate upon a control's HWND (window handle), leave the *Control* parameter blank and specify `ahk_id %ControlHwnd%` for the *WinTitle* parameter (this also works on hidden controls even when [DetectHiddenWindows](#) is Off). The HWND of a control is typically retrieved via [ControlGet Hwnd](#), [MouseGetPos](#), or [DllCall](#).

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

Remarks

Note: To retrieve text from a `ListView`, `ListBox`, or `ComboBox`, use [ControlGet List](#) instead.

If the retrieved text appears to be truncated (incomplete), try using `VarSetCapacity(OutputVar, 55)` prior to `ControlGetText` [replace 55 with a size that is considerably longer than the truncated text]. This is necessary because some applications do not respond properly to the `WM_GETTEXTLENGTH` message, which causes `AutoHotkey` to make the output variable too small to fit all the text.

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 = .`

Text retrieved from most control types uses carriage return and linefeed (`\r\n`) rather than a solitary linefeed (`\n`) to mark the end of each line.

It is not necessary to do `SetTitleMatchMode Slow` because `ControlGetText` always retrieves the text using the slow method (since it works on a broader range of control types).

To retrieve a list of all controls in a window, use [WinGet ControlList](#).

Window titles and text are case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[ControlSetText](#), [WinGetText](#), [Control](#), [ControlGet](#), [ControlMove](#), [ControlFocus](#), [ControlClick](#), [ControlSend](#), [#MaxMem](#)

Example

```
ControlGetText, OutputVar, Edit1, Untitled -
```

ControlMove

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 control's text, both of which can be determined via Window Spy. When using text, the matching behavior is determined by [SetTitleMatchMode](#). If this parameter is blank, the target window's topmost control will be used.

To operate upon a control's HWND (window handle), leave the *Control* parameter blank and specify `ahk_id %ControlHWND%` for the *WinTitle* parameter (this also works on hidden controls even when [DetectHiddenWindows](#) is Off). The HWND of a control is typically retrieved via [ControlGet Hwnd](#), [MouseGetPos](#), or [DllCall](#).

X, Y

The X and Y coordinates (in pixels) of the upper left corner of *Control's* new location, which can be [expressions](#). If either coordinate is blank, *Control's* position in that dimension will not be changed. The coordinates are relative to the upper-left corner of the *Control's* parent window; [ControlGetPos](#) or Window Spy can be used to determine them.

Width, Height

The new width and height of *Control* (in pixels), which can be [expressions](#). If either parameter is blank or omitted, *Control's* size in that dimension will not be changed.

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[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 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.

```
ControlSetText [, Control, NewText, WinTitle, WinText, ExcludeTitle, ExcludeText]
```

Parameters

Control

Can be either ClassNN (the classname and instance number of the control) or the control's text, both of which can be determined via Window Spy. When using text, the matching behavior is determined by [SetTitleMatchMode](#). If this parameter is blank, the target window's topmost control will be used.

To operate upon a control's HWND (window handle), leave the *Control* parameter blank and specify `ahk_id %ControlHwnd%` for the *WinTitle* parameter (this also works on hidden controls even when [DetectHiddenWindows](#) is Off). The HWND of a control is typically retrieved via [ControlGet Hwnd](#), [MouseGetPos](#), or [DllCall](#).

NewText

The new text to set into the control. If blank or omitted, the control is made blank.

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

Remarks

Most control types use carriage return and linefeed (`\r\n`) rather than a solitary linefeed (`\n`) to mark the end of each line. To translate a block of text containing `\n` characters, follow this example:

```
StringReplace, MyVar, MyVar, \n, \r\n, All
```

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 case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[SetControlDelay](#), [ControlGetFocus](#), [ControlGetText](#), [ControlGet](#), [Control](#), [ControlMove](#), [ControlGetPos](#), [ControlClick](#), [ControlFocus](#), [ControlSend](#)

Example

```
ControlSetText, Edit1, New Text Here, Untitled -
```

PostMessage / SendMessage

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,  
Timeout]
```

Parameters

Msg

The message number to send, which can be an [expression](#). See the [message list](#) to determine the number.

wParam

The first component of the message, which can be an [expression](#). If blank or omitted, 0 will be sent.

lParam

The second component of the message, which can be an [expression](#). If blank or omitted, 0 will be sent.

Control

If this parameter is blank or omitted, the message will be sent directly to the target window rather than one of its controls. Otherwise, this parameter can be either ClassNN (the classname and instance number of the control) or the control's text, both of which can be determined via Window Spy. When using text, the matching behavior is determined by [SetTitleMatchMode](#).

To operate upon a control's HWND (window handle), leave the *Control*

parameter blank and specify `ahk_id %ControlHwnd%` for the `WinTitle` parameter (this also works on hidden controls even when `DetectHiddenWindows` is Off). The HWND of a control is typically retrieved via `ControlGet Hwnd`, `MouseGetPos`, or `DllCall`.

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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.

Timeout

[AHK_L 42+]: The maximum number of milliseconds to wait for the target window to process the message. If omitted, it defaults to 5000 (milliseconds), which is also the default behaviour in older versions of AutoHotkey which did not support this parameter. If the message is not processed within this time, the command finishes and sets `ErrorLevel` to the word FAIL. This parameter can be an [expression](#).

ErrorLevel

[v1.1.04+]: These commands are able to throw an exception on failure. For more information, see [Runtime Errors](#).

PostMessage: [ErrorLevel](#) is set to 1 if there was a problem such as the target window or control not existing. Otherwise, it is set to 0.

SendMessage: [ErrorLevel](#) is set to the word FAIL if there was a problem or the command timed out. Otherwise, it is set to the numeric result of the message,

which might sometimes be a "reply" depending on the nature of the message and its target window.

The range of possible values depends on the target window and the version of AutoHotkey that is running. When using a 32-bit version of AutoHotkey, or if the target window is 32-bit, the result is a 32-bit unsigned integer between 0 and 4294967295. When using the 64-bit version of AutoHotkey with a 64-bit window, the result is a 64-bit signed integer between -9223372036854775808 and 9223372036854775807.

If the result is intended to be a 32-bit signed integer (a value from -2147483648 to 2147483648), it can be truncated to 32-bit and converted to a signed value as follows:

```
MsgReply := ErrorLevel << 32 >> 32
```

This conversion may be necessary even on AutoHotkey 64-bit, because results from 32-bit windows are zero-extended. For example, a result of -1 from a 32-bit window is seen as 0xFFFFFFFF on any version of AutoHotkey, whereas a result of -1 from a 64-bit window is seen as 0xFFFFFFFF on AutoHotkey 32-bit and -1 on AutoHotkey 64-bit.

Remarks

These commands should be used with caution because sending a message to the wrong window (or sending an invalid message) might cause unexpected behavior or even crash the target application. This is because most applications are not designed to expect certain types of messages from external sources.

PostMessage places the message in the message queue associated with the target window. It does not wait for acknowledgement or reply. By contrast, SendMessage waits for the target window to process the message, up until the timeout period expires.

The *wParam* and *lParam* parameters should be integers. If AutoHotkey or the target window is 32-bit, only the low 32 bits are used; that is, the value should be between -2147483648 and 4294967295 (0xFFFFFFFF). If AutoHotkey and the target window are both 64-bit, any integer value [supported by AutoHotkey](#) can

be used. As with all integer values in AutoHotkey, a prefix of 0x indicates a hex value. For example, 0xFF is equivalent to 255.

A string may be sent via *wParam* or *lParam* by specifying the address of a variable. The following example uses the [address operator \(&\)](#) to do this:

```
SendMessage, 0xC, 0, &MyVar, ClassNN, WinTitle  
; 0XC is WM_SETTEXT
```

[v1.0.43.06+]: A string put into MyVar by the receiver of the message is properly recognized without the need for extra steps. However, this works only if the parameter's first character is an ampersand (&); for example, `5+&MyVar` would not work but `&MyVar` or `&MyVar+5` would work.

A quoted/literal string may also be sent as in the following working example (the & operator should not be used in this case):

```
Run Notepad  
WinWait Untitled - Notepad  
SendMessage, 0xC, 0, "New Notepad Title" ; 0XC  
is WM_SETTEXT
```

To send a message to all windows in the system, including those that are hidden or disabled, specify `ahk_id 0xFFFF` for *WinTitle* (0xFFFF is `HWND_BROADCAST`). This technique should be used only for messages intended to be broadcast, such as the following example:

```
SendMessage, 0x1A,,,, ahk_id 0xFFFF ; 0x1A is  
WM_SETTINGCHANGE
```

To have a script receive a message, use [OnMessage\(\)](#).

See the [Message Tutorial](#) for an introduction to using these commands.

Window titles and text are case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[Message List](#), [Message Tutorial](#), [OnMessage\(\)](#), [Automating Winamp](#), [DllCall](#), [ControlSend](#), [WinMenuSelectItem](#)

Examples

```
#0:: ; Win+0 hotkey that turns off the monitor.
Sleep 1000 ; Give user a chance to release keys
(in case their release would wake up the monitor
again).
; Turn Monitor Off:
SendMessage, 0x112, 0xF170, 2,, Program Manager ;
0x112 is WM_SYSCOMMAND, 0xF170 is SC_MONITORPOWER.
; Note for the above: Use -1 in place of 2 to turn
the monitor on.
; Use 1 in place of 2 to activate the monitor's
low-power mode.
return

; Start the user's chosen screen saver:
SendMessage, 0x112, 0xF140, 0,, Program Manager ;
0x112 is WM_SYSCOMMAND, and 0xF140 is
SC_SCREENSAVE.

; Scroll up by one line (for a control that has a
vertical scroll bar):
ControlGetFocus, control, A
SendMessage, 0x115, 0, 0, %control%, A

; Scroll down by one line:
ControlGetFocus, control, A
SendMessage, 0x115, 1, 0, %control%, A

; Switch the active window's keyboard
layout/language to English:
```

```
PostMessage, 0x50, 0, 0x4090409,, A ; 0x50 is  
WM_INPUTLANGCHANGEREQUEST.
```

```
; This example asks Winamp which track number is  
currently active:
```

```
SetTitleMatchMode, 2
```

```
SendMessage, 1024, 0, 120,, - Winamp
```

```
if ErrorLevel <> FAIL
```

```
{
```

```
    ErrorLevel++ ; Winamp's count starts at "0",  
so adjust by 1.
```

```
    MsgBox, Track #%ErrorLevel% is active or  
playing.
```

```
}
```

```
; See Automating Winamp for more information.
```

```
; To find the process ID of an AHK script (an  
alternative to "WinGet PID"): 
```

```
SetTitleMatchMode, 2
```

```
DetectHiddenWindows, On
```

```
SendMessage, 0x44, 0x405, 0, , SomeOtherScript.ahk  
- AutoHotkey v
```

```
MsgBox %ErrorLevel% is the process id.
```

SetControlDelay

Sets the delay that will occur after each control-modifying command.

SetControlDelay, Delay

Parameters

Delay

Time in milliseconds, which can be an [expression](#). Use -1 for no delay at all and 0 for the smallest possible delay. If unset, the default delay is 20.

Remarks

A short delay (sleep) is done automatically 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.

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

Every newly launched [thread](#) (such as a [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
```

WinMenuSelectItem

Invokes a menu item from the menu bar of the specified window.

```
WinMenuSelectItem, WinTitle, WinText, Menu [,  
SubMenu1, SubMenu2, SubMenu3, SubMenu4, SubMenu5,  
SubMenu6, ExcludeTitle, ExcludeText ]
```

Parameters

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). Hidden text elements are detected if [DetectHiddenText](#) is ON.

Menu

The name of the top-level menu item, 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.

[v1.1.28+]: Specify `0&` to use the window's [system menu](#).

SubMenu1

The name of the menu item to select or its position (see above).

This can be omitted if the top-level item does not contain a menu (rare).

SubMenu2

If *SubMenu1* itself contains a menu, this is the name of the menu item inside, or its position.

SubMenu3
SubMenu4
SubMenu5
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.

ErrorLevel

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

[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 case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

System Menu [v1.1.28+]

Menu can be `0&` to select an item within the window's system menu, which typically appears when the user presses `Alt` `Space` on the icon in the window's title bar. For example:

```
; Paste a command into cmd.exe without
activating the window.
Clipboard := "echo Hello, world!`r"
WinMenuItem ahk_exe cmd.exe,, 0&, Edit,
Paste
```

Caution: Use this only on windows which have custom items in their system menu.

If the window does not already have a custom system menu, a copy of the standard system menu will be created and assigned to the target window as a side effect. This copy is destroyed by the system when the script exits, leaving other scripts unable to access it. Therefore, avoid using `0&` for the standard items which appear on all windows. Instead, post the [WM_SYSCOMMAND](#) message directly. For example:

```
; Like "WinMinimize A", but also play the system
sound for minimizing.
WM_SYSCOMMAND := 0x112
SC_MINIMIZE := 0xF020
PostMessage WM_SYSCOMMAND, SC_MINIMIZE, 0,, A
```

Related

ControlSend, PostMessage

Example

```
; This will select File->Open in Notepad:
```

```
WinMenuSelectItem, Untitled - Notepad, , File,  
Open
```

```
; Same as above except it's done by position vs.  
name:
```

```
WinMenuSelectItem, 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.

ErrorLevel [AHK_L 54+]

[ErrorLevel](#) is set to 1 if no window was found to activate or 0 otherwise.

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. Group names are not case sensitive.

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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 group (or this *window specification* prior to AHK_L 54) 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.

Remarks

Each use of this command adds a new rule to a group. In other words, a group consists of a set of criteria rather than a fixed list of windows. Later, when a group is used by a command such as [GroupActivate](#), each window on the desktop is checked against each of these criteria. If a window matches one of the criteria in the group, it is considered a match.

Although [SetTitleMatchMode](#) and [DetectHiddenWindows](#) do not directly affect the behavior of this command, they do affect the other group commands such as [GroupActivate](#) and [GroupClose](#). They also affect the use of `ahk_group` in any other command's [WinTitle](#).

A window group is typically used to bind together a collection of related windows, which is useful for tasks that involve many related windows, or an application that owns many subwindows. For example, if you frequently work with many instances of a graphics program or text editor, you can use [GroupActivate](#) on a hotkey to visit each instance of that program, one at a time, 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 (top part of the script). 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, AllWindows
```

All windowing commands can operate upon a window group by specifying `ahk_group MyGroupName` for the *WinTitle* parameter. The commands [WinMinimize](#), [WinMaximize](#), [WinRestore](#), [WinHide](#), [WinShow](#), [WinClose](#), and [WinKill](#) will act upon **all** the group's windows. To instead act upon only the topmost window, follow this example:

```
WinHide % "ahk_id " . WinExist("ahk_group  
MyGroup")
```

By contrast, the other window commands such as [WinActivate](#) and [IfWinExist](#) will operate only upon the topmost window of the group.

Related

[GroupActivate](#), [GroupDeactivate](#), [GroupClose](#)

Examples

```
; In the autoexecute section at the top of the
script:
GroupAdd, MSIE, ahk_class IEFramE ; Add only
Internet Explorer windows to this group.
return ; End of autoexecute 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, Sear&ch 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  
visit each Outlook window, one at a time.
```

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. This is the same effect as `WinClose ahk_group GroupName`.

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.
```

The *WinTitle* Parameter & the Last Found Window

Many commands and a few functions have a *WinTitle* parameter, used to identify which window (or windows) to operate on. This parameter can be the title or partial title of the window, and/or any other criteria described on this page.

Quick Reference:

<i>Title</i>	Matching Behaviour
A	The Active Window
ahk_class	Window Class
ahk_id	Unique ID/HWND
ahk_pid	Process ID
ahk_exe	Process Name/Path
ahk_group	Window Group
	Multiple Criteria
(All empty)	Last Found Window

Matching Behaviour

[SetTitleMatchMode](#) controls how a partial or complete title is compared against the title of each window. Depending on the setting, *WinTitle* can be an exact title, or it can contain a prefix, a substring which occurs anywhere in the title, or a [RegEx pattern](#). This setting also controls whether [ahk_class](#) is interpreted as an exact class name or a RegEx pattern.

Window titles are case sensitive, except when using [the i\) modifier](#) in a RegEx pattern.

Hidden windows are only detected if [DetectHiddenWindows](#) is On, except with [WinShow](#), which always detects hidden windows.

If multiple windows match the *WinTitle* and any other criteria, the topmost matching window is used. If the active window matches the criteria, it usually takes precedence since it is usually above all other windows. However, if an [always-on-top](#) window also matches (and the active window is not always-on-top), it may be used instead.

Active Window (A)

If *WinTitle* is the letter **A** and the other 3 window parameters (*WinText*, *ExcludeTitle* and *ExcludeText*) are blank or omitted, the active window will be used.

```
; Retrieve the ID/HWND of the active window
id := WinExist("A")
MsgBox % id

; Press Win+↑ to maximize the active window
#Up::WinMaximize, A
```

ahk_class Window Class

A window class is a set of attributes that the system uses as a template to create a window. In other words, the class name of the window identifies what *type* of window it is. To use a window class, specify `ahk_class ExactClassName` as shown by Window Spy. *ExactClassName* can be retrieved by [WinGetClass](#).

If the RegEx [title matching mode](#) is active, `ahk_class` accepts a [regular expression](#).

```
; Activate a console window (e.g. cmd.exe)
WinActivate, ahk_class ConsoleWindowClass
```

ahk_id Unique ID / HWND

Each window or control has a unique ID, also known as a HWND (short for handle to window). This ID can be used to identify the window or control even if its title changes. The ID of a window is typically retrieved via [WinExist\(\)](#) or [WinGet](#). The ID of a control is typically retrieved via [ControlGet Hwnd](#), [MouseGetPos](#), or [DllCall](#). Also, ahk_id will operate on controls even if they are hidden; that is, the setting of [DetectHiddenWindows](#) does not matter for controls.

```
WinActivate, ahk_id %VarContainingID%
```

ahk_pid Process ID

Use `ahk_pid` to identify a window belonging to a specific process. The process identifier (PID) is typically retrieved by [WinGet](#), [Run](#) or [Process](#).

```
WinActivate, ahk_pid %VarContainingPID%
```

ahk_exe Process Name/Path [v1.1.01+]

Use ahk_exe to identify a window belonging to any process with the given name or path.

While ahk_pid is limited to one specific process, ahk_exe considers all processes with name or full path matching a given string. If the RegEx [title matching mode](#) is active, ahk_exe accepts a [regular expression](#) which must match the full path of the process. Otherwise, ahk_exe accepts a case-insensitive name or full path; for example, ahk_exe notepad.exe covers ahk_exe C:\Windows\notepad.exe, ahk_exe C:\Windows\System32\notepad.exe and other variations.

```
; Activate an existing notepad.exe window, or
open a new one
if WinExist("ahk_exe notepad.exe")
    WinActivate, ahk_exe notepad.exe
else
    Run, notepad.exe
```

```
SetTitleMatchMode RegEx
WinActivate ahk_exe i)\notepad\.exe$ ; Match
the name part of the full path.
```

ahk_group Window Group

Use `ahk_group` to identify a window or windows matching the rules contained by a previously defined [window group](#).

The commands [WinMinimize](#), [WinMaximize](#), [WinRestore](#), [WinHide](#), [WinShow](#), [WinClose](#), and [WinKill](#) will act on **all** the group's windows. By contrast, the other window commands such as [WinActivate](#) and [IfWinExist](#) will operate only on the topmost window of the group.

```
; Define the group: Windows Explorer windows
GroupAdd, Explorer, ahk_class ExploreWClass ;
Unused on Vista and later
GroupAdd, Explorer, ahk_class CabinetWClass

; Activate any window matching the above
criteria
WinActivate, ahk_group Explorer
```

Multiple Criteria

By contrast with [ahk_group](#) (which broadens the search), the search may be narrowed by specifying more than one criterion within the WinTitle parameter. In the following example, the script waits for a window whose title contains *My File.txt* **and** whose class is *Notepad*:

```
WinWait My File.txt ahk_class Notepad
WinActivate ; Activate the window it found.
```

When using this method, the text of the title (if any is desired) should be listed first, followed by one or more additional criteria. Criteria beyond the first should be separated from the previous with exactly one space or tab (any other spaces or tabs are treated as a literal part of the previous criterion).

[ahk_id](#) can be combined with other criteria to test a window's title, class or other attributes:

```
MouseGetPos,,, id
if WinExist("ahk_class Notepad ahk_id " id)
    MsgBox The mouse is over Notepad.
```

The "Last Found" Window

This is the window most recently found by `IfWin[Not]Exist`, `WinExist()`, `IfWin[Not]Active`, `WinActive()`, `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 has 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.

If the last found window is a hidden `Gui window`, it can be used even when `DetectHiddenWindows` is Off. This is often used in conjunction with `Gui +LastFound`.

```
Run Notepad
WinWait Untitled - Notepad
WinActivate ; Uses the last found window.

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
}
```

#WinActivateForce

Skips the gentle method of activating a window and goes straight to the forceful method.

```
#WinActivateForce
```

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.

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: Hidden text is not detected.

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 [thread](#) (such as a [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 turned 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](#). By contrast, [WinShow](#) will always unhide a hidden window even if hidden windows are not being detected.

Turning on DetectHiddenWindows is not necessary when accessing a control or child window via the [ahk_id method](#) or as the [last-found-window](#). It is also not necessary when accessing GUI windows via `Gui +LastFound`.

The built-in variable **A_DetectHiddenWindows** contains the current setting (On or Off).

Every newly launched [thread](#) (such as a [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

[DetectHiddenText](#)

Example

```
DetectHiddenWindows, On
```

SetTitleMatchMode

Sets the matching behavior of the *WinTitle* parameter in commands such as [WinWait](#).

```
SetTitleMatchMode, MatchMode  
SetTitleMatchMode, Fast|Slow
```

Parameters

MatchMode

One of the following digits or the word `RegExp`:

- 1: A window's title must start with the specified *WinTitle* to be a match.
- 2: A window's title can contain *WinTitle* anywhere inside it to be a match.
- 3: A window's title must exactly match *WinTitle* to be a match.

RegExp [v1.0.45+]: Changes *WinTitle*, *WinText*, *ExcludeTitle*, and *ExcludeText* to accept [regular expressions](#). Do not enclose such expressions in quotes when using them with commands. For example: `WinActivate Untitled.*Notepad`.

Note:

- `RegExp` also applies to `ahk_class` and `ahk_exe`; for example, `ahk_class IEFrame` searches for any window whose class name contains *IEFrame* anywhere (this is because by default, regular expressions find a match *anywhere* in the target string).
- For *WinTitle*, each component is separate. For example, in `i)^untitled ahk_class i)^notepad$ ahk_pid %mypid%, i)^untitled` and `i)^notepad$` are separate regex patterns and `%mypid%` is always compared numerically (it is not a regex pattern).
- For *WinText*, each text element (i.e. each control's text) is matched against the `RegExp` separately. Therefore, it is not possible to have a

match span more than one text element.

The modes above also affect *ExcludeTitle* in the same way as *WinTitle*. For example, mode 3 requires that a window's title exactly match *ExcludeTitle* for that window to be excluded.

Fast|Slow

One of the following words to specify how the *WinText* and *ExcludeText* parameters should be matched:

Fast: This is the default behavior. Performance may be substantially better than *Slow*, but certain types of controls are not detected. For instance, text is typically detected within Static and Button controls, but not Edit controls, unless they are owned by the script.

Slow: Can be much slower, but works with all controls which respond to the [WM_GETTEXT](#) message.

Remarks

This command affects the behavior of all windowing commands, e.g. [WinExist](#) and [WinActivate](#). [WinGetText](#) is affected in the same way as other commands, but it always uses the *Slow* method to retrieve text.

If unspecified, *TitleMatchMode* defaults to 1 and *fast*.

If a [window group](#) is used, the current title match mode applies to each individual rule in the group.

Generally, the *slow* mode should be used only 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".

Window Spy has an option for *Slow TitleMatchMode* 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.

Regardless of the current TitleMatchMode, *WinTitle*, *WinText*, *ExcludeTitle* and *ExcludeText* are case sensitive. The only exception is the [case-insensitive option](#) of the RegEx mode; for example: `i)untitled - notepad`.

Every newly launched [thread](#) (such as a [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](#), [RegExMatch\(\)](#)

Example

```
SetTitleMatchMode 2  
; OR:  
SetTitleMatchMode RegEx  
  
SetTitleMatchMode Slow ; Slow/Fast can be set  
independently of all the other modes.
```

SetWinDelay

Sets the delay that will occur after each windowing command, such as [WinActivate](#).

SetWinDelay, Delay

Parameters

Delay

Time in milliseconds, which can be an [expression](#). Use -1 for no delay at all and 0 for the smallest possible delay. If unset, the default delay is 100.

Remarks

A short delay (sleep) is done automatically after every windowing command except [IfWinActive](#) and [IfWinExist](#). 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.

The built-in variable `A_WinDelay` contains the current setting.

Every newly launched [thread](#) (such as a [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](#), [SendMode](#)

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, which can be an [expression](#). Default 1, which is usually the part that contains the text of interest.

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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

[v1.1.04+]: This command is able to throw an exception on failure. For more

information, see [Runtime Errors](#).

[ErrorLevel](#) is set to 1 if there is a problem or 0 otherwise. If there was a problem, *OutputVar* is also made blank.

Remarks

This command 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, in which case the text cannot be retrieved.

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 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.

To instead wait for the bar's text to *change*, either use [StatusBarGetText](#) in a loop, or use the RegEx example at the bottom of this page.

Seconds

The number of seconds (can contain a decimal point or be an [expression](#)) 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, which can be an [expression](#). Default 1, which is usually the part that contains the text of interest.

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility).

Hidden text elements are detected if [DetectHiddenText](#) is ON.

Interval

How often the status bar should be checked while the command is waiting (in milliseconds), which can be an [expression](#). 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.

ErrorLevel

[v1.1.04+]: This command is able to throw an exception if the status bar couldn't be accessed. For more information, see [Runtime Errors](#).

[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

StatusBarWait attempts to read the first *standard* status bar on a window (class `msctls_statusbar32`). Some programs use their own status bars or special versions of the MS common control. Such bars are not supported.

Rather than using [StatusBarGetText](#) in a loop, it is usually more efficient to use StatusBarItem because it contains optimizations that avoid the overhead that repeated calls to [StatusBarGetText](#) would incur.

StatusBarItem determines its target window before it begins waiting for a match. If that target window is closed, the command will stop waiting even if there is another window matching the specified *WinTitle* and *WinText*.

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 case sensitive. Hidden windows are not detected

unless [DetectHiddenWindows](#) has been turned on.

Related

[StatusBarGetText](#), [WinGetTitle](#), [WinGetText](#), [ControlGetText](#)

Example

```
; The following example enters a new search
pattern into an existing Explorer/Search window.
IfWinExist, Search Results ; Sets the Last Found
window to simplify the below.
{
    WinActivate
    Send, {tab 2}!o*.txt{enter} ; In the Search
window, enter the pattern to search for.
    Sleep, 400 ; Give the status bar time to
change to "Searching".
    StatusBarWait, found, 30
    if ErrorLevel
        MsgBox, The command timed out or there was
a problem.
    else
        MsgBox, The search successfully completed.
}
```

```
; The following example waits for the status bar
of the active window to change. This example
requires [v1.0.46.06+].
SetTitleMatchMode RegEx
IfWinExist A ; Set the last-found window to be
the active window (for use below).
{
    StatusBarGetText, OrigText
```

```
StatusBarWait, ^(?!\Q%OrigText%\E$) ; This  
regular expression waits for any change to the  
text.  
}
```

WinActivate

Activates the specified window.

```
WinActivate [, WinTitle, WinText, ExcludeTitle,  
ExcludeText]
```

Parameters

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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

When an inactive window becomes active, the operating system also makes it foremost (brings it to the top of the stack). This does not occur if the window is already active.

If the window is minimized and inactive, it is automatically restored prior to being activated. [v1.1.20+]: If *WinTitle* is the letter "A" and the other parameters are omitted, the active window is restored. [v1.1.28.02+]: The window is restored even if it was already active.

Six attempts will be made to activate the target window over the course of 60ms. Thus, it is usually unnecessary to follow WinActivate with [WinWaitActive](#) or [IfWinNotActive](#).

In general, if more than one window matches, the topmost matching window (typically the one most recently used) will be activated. If the window is already active, it will be kept active rather than activating any other matching window beneath it. However, if the active window is moved to the bottom of the stack with [WinSet Bottom](#), some other window may be activated even if the active window is a match.

[WinActivateBottom](#) activates the bottommost matching window (typically the one least recently used).

[GroupActivate](#) activates the next window that matches criteria specified by a window group.

[v1.1.20+]: If the active window is hidden and [DetectHiddenWindows](#) is turned off, it is never considered a match. Instead, a visible matching window is activated if one exists.

When a window is activated immediately after the activation of some other window, task bar buttons might start flashing on some systems (depending on OS and settings). To prevent this, use [#WinActivateForce](#).

Window titles and text are case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Known issue: If the script is running on a computer or server being accessed via remote desktop, WinActivate may hang if the remote desktop client is minimized. One workaround is to use commands which don't require window activation, such as [ControlSend](#) and [ControlClick](#). Another possible workaround is to apply the following registry setting on the local/client computer:

```
; Change HKCU to HKLM to affect all users on
this system.
RegWrite REG_DWORD, HKCU,
Software\Microsoft\Terminal Server Client
, RemoteDesktop_SuppressWhenMinimized, 2
```

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 that it activates the bottommost matching window rather than the topmost.

```
WinActivateBottom [ , WinTitle, WinText, ExcludeTitle, ExcludeText ]
```

Parameters

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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

The bottommost window is typically the one least recently used, except when windows have been reordered, such as with [WinSet Bottom](#).

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.

If the window is minimized and inactive, it is automatically restored prior to being activated. [v1.1.20+]: If *WinTitle* is the letter "A" and the other parameters are omitted, the active window is restored. [v1.1.28.02+]: The window is restored even if it was already active.

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 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:
#i::
SetTitleMatchMode, 2
WinActivateBottom, - Microsoft Internet Explorer
return
```

WinClose

Closes the specified window.

```
WinClose [, WinTitle, WinText, SecondsToWait, ExcludeTitle, ExcludeText]
```

Parameters

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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 or be an [expression](#)) 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.

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 topmost (most recently used) will be closed.

This command operates only upon the topmost matching window except when *WinTitle* is [ahk_group GroupName](#), in which case all windows in the group are affected.

WinClose sends a WM_CLOSE message to the target window, which is a somewhat forceful method of closing it. An alternate method of closing is to send the following message. It might produce different behavior because it is similar in effect to pressing  clicking the window's close button in its title bar:

```
PostMessage, 0x112, 0xF060,,, WinTitle, WinText  
; 0x112 = WM_SYSCOMMAND, 0xF060 = SC_CLOSE
```

If a window does not close via WinClose, you can force it to close with [WinKill](#).

Window titles and text are 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

Retrieves the specified window's unique ID, process ID, process name, or a list of its controls. It can also retrieve a list of all windows matching the specified criteria.

```
WinGet, OutputVar [, Cmd, WinTitle, WinText,  
ExcludeTitle, ExcludeText]
```

Parameters

OutputVar

The name of the variable in which to store the result of *Cmd*.

Cmd

See list below.

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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.

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 (HWND/handle) of a window. If there is no matching window, *OutputVar* is made blank. The functions [WinExist\(\)](#) and [WinActive\(\)](#) can also be used to retrieve the ID of a window; for example, `WinExist("A")` is a fast way to get the ID of the active window. To discover the HWND of a control (for use with [Post/SendMessage](#) or [DllCall](#)), use [ControlGet Hwnd](#) or [MouseGetPos](#).

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 \(PID\)](#) of a window.

ProcessName: Retrieves the name of the process (e.g. notepad.exe) that owns a window. If there are no matching windows, *OutputVar* is made blank.

ProcessPath [v1.1.01+]: Similar to *ProcessName*, but retrieves the full path and name of the process instead of just the name.

Count: Retrieves the number of existing windows that match the specified *WinTitle*, *WinText*, *ExcludeTitle*, and *ExcludeText* (0 if none). To count all windows on the system, omit all four title/text parameters. Hidden windows are included only if [DetectHiddenWindows](#) has been turned on.

List: Retrieves the unique ID numbers of all existing windows that match the specified *WinTitle*, *WinText*, *ExcludeTitle*, and *ExcludeText* (to retrieve all windows on the entire system, omit all four title/text parameters). Each ID number is stored in a variable whose name begins with *OutputVar*'s own name (to form a [pseudo-array](#)), 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. Windows are retrieved in order from topmost to bottommost (according to how they are stacked on the desktop). Hidden windows are included only if [DetectHiddenWindows](#) has been turned on. Within a [function](#), to create a pseudo-array that is global instead of local, [declare](#) *MyArray* as a global variable prior to using this command (the converse is true for [assume-global](#) functions). However, it is often also necessary to declare each variable in the set, due to a [common source of confusion](#).

MinMax: Retrieves the minimized/maximized state for a 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 controls in a window. If no matching window exists or there are no controls in the window, *OutputVar* is made blank. Otherwise, each control name consists of its class name followed immediately by its sequence number (ClassNN), 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 the navigation order via  the window supports tabbing.

The control currently under the mouse cursor can be retrieved via [MouseGetPos](#).

ControlListHwnd [v1.0.43.06+]: Same as above except it retrieves the [window handle \(HWND\)](#) of each control rather than its ClassNN.

Transparent: Retrieves the degree of transparency of a window (see [WinSet](#) for how to set transparency). *OutputVar* is made blank if: 1) the OS is older than Windows XP; 2) there are no matching windows; 3) the window has no transparency level; or 4) other conditions (caused by OS behavior) such as the window having been minimized, restored, and/or resized since it was made transparent. Otherwise, a number between 0 and 255 is stored, where 0 indicates an invisible window and 255 indicates an opaque window. For example:

```
MouseGetPos,,, MouseWin
WinGet, Transparent, Transparent, ahk_id
%MouseWin% ; Transparency of window under the
mouse cursor.
```

TransparentColor: Retrieves the color that is marked transparent in a window (see

[WinSet](#) for how to set the TransColor). *OutputVar* is made blank if: 1) the OS is older than Windows XP; 2) there are no matching windows; 3) the window has no transparent color; or 4) other conditions (caused by OS behavior) such as the window having been minimized, restored, and/or resized since it was made transparent. Otherwise, a six-digit hexadecimal RGB color is stored, e.g. 0x00CC99. For example:

```
MouseGetPos,,, MouseWin
WinGet, TransColor, TransColor, ahk_id
%MouseWin% ; TransColor of the window under the
mouse cursor.
```

Style or **ExStyle**: Retrieves an 8-digit hexadecimal number representing style or extended style (respectively) of a 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, Style, My Window Title
if (Style & 0x80000000) ; 0x80000000 is
WS_DISABLED.
... 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, ExStyle, My Window Title
if (ExStyle & 0x8) ; 0x8 is WS_EX_TOPMOST.
... the window is always-on-top, so perform
appropriate action.
```

See the [styles table](#) for a partial listing of 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 integers, 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 integers.

Window titles and text are case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[WinGetClass](#), [Process](#), [WinGetTitle](#), [MouseGetPos](#), [ControlGet](#), [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%
{
    this_id := id%A_Index%
    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 the specified 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

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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

Only the class name is retrieved (the prefix "ahk_class" is not included in *OutputVar*).

Window titles and text are 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 the specified 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

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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

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 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 the specified window.

```
WinGetText, OutputVar [, WinTitle, WinText,  
ExcludeTitle, ExcludeText]
```

Parameters

OutputVar

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

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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

[v1.1.04+]: This command is able to throw an exception on failure. For more information, see [Runtime Errors](#).

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

Remarks

The text retrieved is generally the same as what Window Spy shows for that window. However, if [DetectHiddenText](#) has been turned off, hidden text is omitted from *OutputVar*.

Each text element ends with a carriage return and linefeed (CR+LF), which can be represented in the script as ``r`n`. To extract individual lines or substrings, use commands such as [StringGetPos](#) and [StringMid](#). A [parsing loop](#) can also be used to examine each line or word one by one.

If the retrieved text appears to be truncated (incomplete), try using `VarSetCapacity(OutputVar, 55)` prior to `WinGetText` [replace 55 with a size that is considerably longer than the truncated text]. This is necessary because some applications do not respond properly to the `WM_GETTEXTLENGTH` message, which causes AutoHotkey to make the output variable too small to fit all the text.

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. In any case, a variable's memory can be freed later by assigning it to nothing, i.e. `OutputVar = .`

To retrieve a list of all controls in a window, follow this example: `WinGet, OutputVar, ControlList, WinTitle`

Window titles and text are case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[ControlGetText](#), [WinGetActiveStats](#), [WinGetActiveTitle](#), [WinGetTitle](#), [WinGetPos](#), [#MaxMem](#)

Example

```
Run, Calc.exe  
WinWait, Calculator  
WinGetText, text ; The window found above will be  
used.  
MsgBox, The text is:`n%text%
```

WinGetTitle

Retrieves the title of the specified window.

```
WinGetTitle, OutputVar [, WinTitle, WinText,  
ExcludeTitle, ExcludeText]
```

Parameters

OutputVar

The name of the variable in which to store the retrieved title.

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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

To discover the name of the window that the mouse is currently hovering over, use [MouseGetPos](#).

Window titles and text are 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 the specified window.

```
WinHide [, WinTitle, WinText, ExcludeTitle,  
ExcludeText]
```

Parameters

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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

Use [WinShow](#) to unhide a hidden window ([DetectHiddenWindows](#) can be either On or Off to do this).

This command operates only upon the topmost matching window except when *WinTitle* is [ahk_group GroupName](#), in which case all windows in the group are affected.

The Explorer taskbar may be hidden/shown as follows:

```
WinHide ahk_class Shell_TrayWnd  
WinShow ahk_class Shell_TrayWnd
```

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 the specified window to close.

```
WinKill [, WinTitle, WinText, SecondsToWait,  
ExcludeTitle, ExcludeText]
```

Parameters

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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 or be an [expression](#)) 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.

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 topmost (most recently used) will be closed.

This command operates only upon the topmost matching window except when *WinTitle* is [ahk_group GroupName](#), in which case all windows in the group are affected.

Window titles and text are 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 the specified window to its maximum size.

```
WinMaximize [, WinTitle, WinText, ExcludeTitle,  
ExcludeText]
```

Parameters

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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

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
```

This command operates only upon the topmost matching window except when *WinTitle* is `ahk_group GroupName`, in which case all windows in the group are affected.

Window titles and text are 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 the specified window into a button on the task bar.

```
WinMinimize [, WinTitle, WinText, ExcludeTitle,  
ExcludeText]
```

Parameters

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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

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
```

This command operates only upon the topmost matching window except when *WinTitle* is `ahk_group GroupName`, in which case all windows in the group are affected.

Window titles and text are case sensitive. Hidden windows are not detected unless `DetectHiddenWindows` has been turned on.

Related

[WinRestore](#), [WinMaximize](#), [WinMinimizeAll](#)

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[Undo]

Minimizes or unminimizes all windows.

```
WinMinimizeAll  
WinMinimizeAllUndo
```

On most systems, this is equivalent to Explorer's `D` keys.



Related

[WinMinimize](#), [GroupAdd](#)

Example

```
WinMinimizeAll  
WinMinimizeAllUndo
```

WinMove

Changes the position and/or size of the specified window.

```
WinMove, X, Y  
WinMove, WinTitle, WinText, X, Y [, Width, Height,  
ExcludeTitle, ExcludeText]
```

Parameters

X, Y

The X and Y coordinates (in pixels) of the upper left corner of the target window's new location, which can be [expressions](#). The upper-left pixel of the screen is at 0, 0.

If these are the only parameters given with the command, the [Last Found Window](#) will be used as the target window.

Otherwise, X and/or Y can be omitted, in which case the current position is used.

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#). See also the [known limitation](#) below.

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). Hidden text elements are detected if [DetectHiddenText](#) is ON. See also the [known limitation](#) below.

Width, Height

The new width and height of the window (in pixels), which can be [expressions](#). 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.

Remarks

If *Width* and *Height* are small (or negative), most windows with a title bar will generally go no smaller than 112 x 27 pixels (however, some types of windows may have a different minimum size). If *Width* and *Height* are large, most windows 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.

The speed of WinMove is affected by [SetWinDelay](#).

Window titles and text are case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Known limitation: If *WinTitle* or *WinText* contains (, [or { , but not the closing counterpart, such as `WinMove, KEDIT - [`, the parameter is automatically interpreted as an [expression](#), resulting in an error message. To avoid this, you can use a [leading percent sign](#) to force a literal string instead, such as `WinMove, % "KEDIT - ["`.

Related

[ControlMove](#), [WinGetPos](#), [WinHide](#), [WinMinimize](#), [WinMaximize](#), [WinSet](#)

Example

```
Run, calc.exe
```

```
WinWait, Calculator
WinMove, 0, 0 ; Move the window found by WinWait
to the upper-left corner of the screen.
```

```
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
```

```
; The following function centers the specified
window on the screen:
```

```
CenterWindow(WinTitle)
```

```
{
    WinGetPos,,, Width, Height, %WinTitle%
    WinMove, %WinTitle%,, (A_ScreenWidth/2)-
    (Width/2), (A_ScreenHeight/2)-(Height/2)
}
```

WinRestore

Unminimizes or unmaximizes the specified window if it is minimized or maximized.

```
WinRestore [, WinTitle, WinText, ExcludeTitle,  
ExcludeText]
```

Parameters

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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

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
```

This command operates only upon the topmost matching window except when

WinTitle is [ahk_group GroupName](#), in which case all windows in the group are affected.

Window titles and text are case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[WinMinimize](#), [WinMaximize](#)

Example

```
WinRestore, Untitled - Notepad
```

WinSet

Makes a variety of changes to the specified window, such as "always on top" and transparency.

```
WinSet, Attribute, Value [, WinTitle, WinText,  
ExcludeTitle, ExcludeText]
```

Parameters

Attribute, Value

See list below.

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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.

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` `WinSet, Bottom,, WinTitle`.

Top: Brings a window to the top of the stack without explicitly [activating](#) it. However, the system default settings will probably cause it to activate in most cases. In addition, this command may have no effect due to the operating system's protection against applications that try to steal focus from the user (it may depend on factors such as what type of window is currently active and what the user is currently doing). One possible work-around is to make the window briefly [AlwaysOnTop](#), then turn off `AlwaysOnTop`.

Disable or **Enable:** Disables or enables a window (respectively). When a window is disabled, the user cannot move it or interact with its controls. In addition, disabled windows are omitted from the alt-tab list.

Redraw: Attempts to update the appearance/contents of a window by informing the OS that the window's rectangle needs to be redrawn. If this method does not work for a particular window, try [WinMove](#). If that does not work, try the following:

```
WinHide, WinTitle
WinShow, WinTitle
```

Style, N or **ExStyle, N:** Changes the style or extended style of a window, respectively. If the first character of **N** is a plus or minus sign, the style(s) in **N** are added or removed, respectively. If the first character is a caret (^), the style(s) in **N** are each toggled to the opposite state. If the first character is a digit, the window's style is overwritten completely; that is, it becomes **N**.

[ErrorLevel](#) is set to 1 upon failure and 0 upon success. Failure occurs if the target window is not found or the style is not allowed to be applied.

After applying certain style changes to a visible window, it might be necessary to redraw the window using `WinSet Redraw` (see below). Finally, the [styles table](#) lists some of the common style numbers. Examples:

```
WinSet, Style, -0xC00000, A ; Remove the active
window's title bar (WS_CAPTION).
WinSet, ExStyle, ^0x80, WinTitle ; Toggle the
WS_EX_TOOLWINDOW attribute, which removes/adds
the window from the alt-tab list.
```

WinSet, Region [, Options, WinTitle, ...]

Changes the shape of a window to be the specified rectangle, ellipse, or polygon. If the *Options* parameter is blank, the window is restored to its original/default display area. Otherwise, one or more of the following options can be specified, each separated from the others with space(s):

Wn: Width of rectangle or ellipse. For example: `w200` .

Hn: Height of rectangle or ellipse. For example: `h300` .

X-Y: Each of these is a pair of X/Y coordinates. For example, `200-0` would use 200 for the X coordinate and 0 for the Y.

E: Makes the region an ellipse rather than a rectangle. This option is valid only when **W** and **H** are present.

R[w-h]: Makes the region a rectangle with rounded corners. For example, `R30-30` would use a 30x30 ellipse for each corner. If **w-h** is omitted, 30-30 is used.

R is valid only when **W** and **H** are present.

Rectangle or ellipse: If the **W** and **H** options are present, the new display area will be a rectangle whose upper left corner is specified by the first (and only) pair of **X-Y** coordinates. However, if the **E** option is also present, the new display area will be an ellipse rather than a rectangle. For example: `WinSet, Region, 50-0 w200 H250 E, WinTitle` .

Polygon: When the **W** and **H** options are absent, the new display area will be a polygon determined by multiple pairs of **X-Y** coordinates (each pair of coordinates is a point inside the window relative to its upper left corner). For example, if three pairs of coordinates are specified, the new display area will be a triangle in most cases. The order of the coordinate pairs with respect to each other is sometimes important. In addition, the word **Wind** maybe be present in *Options* to use the winding method instead of the alternating method to determine the polygon's region.

[ErrorLevel](#) is set to 1 upon failure and 0 upon success. Failure occurs when: 1) the target window does not exist; 2) one or more *Options* are invalid; 3) more than 2000 pairs of coordinates were specified; or 4) the specified region is invalid or could not be applied to the target window.

See the bottom of this page for examples of how to use this command.

WinSet, Transparent, N, WinTitle

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. 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 usage of system resources.

Known Limitations for Transparent and [TransColor](#):

- Setting "Transparent" to 255 prior to turning it off might avoid window redrawing problems such as a black background. If the window still fails to be redrawn correctly, see [Redraw](#) for a possible workaround.
- To change a window's existing [TransColor](#), it may be necessary to turn off transparency before making the change.
- On [v1.1.24.04] and earlier, they have no effect on a window that lacks a caption (title bar) and lacks the [always-on-top](#) property. This was fixed in [v1.1.24.05].

Tip: 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 the Start Menu's submenus transparent,
also include the script below.
```

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
```

WinSet, TransColor, Color [N], WinTitle

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). For example: `WinSet, TransColor, EEAA99 150, WinTitle`.

TransColor is often used to create on-screen displays and other visual effects. There is an example of an on-screen display [at the bottom of the Gui page](#).

The word OFF may be specified to completely turn off transparency for a window. Both of the following are identical in function:

```
WinSet, Transparent, Off, WinTitle
WinSet, TransColor, Off, WinTitle
```

Known Limitations: See the list [above](#).

Remarks

[ErrorLevel](#) is not changed by this command except where indicated above.

Although transparency is supported on Windows 2000/XP or later, retrieving the current transparency settings of a window is possible only on Windows XP or later (via [WinGet](#)).

A script's [SplashText](#) window can be made non-AlwaysOnTop via:

```
WinSet, AlwaysOnTop, Off, My Splash Window Title
```

Window titles and text are case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[WinGet](#), [WinHide](#), [WinSetTitle](#), [WinMove](#), [WinActivate](#), [Control](#)

Examples

```
WinSet, Transparent, 200, Untitled - Notepad ;  
Make the window a little bit transparent.  
WinSet, TransColor, White, Untitled - Notepad ;  
Make all white pixels invisible.  
WinSet, AlwaysOnTop, toggle, Calculator ; Toggle  
the always-on-top status of Calculator.
```

```
; Longer Example:  
; Here are some hotkeys that demonstrate the  
effects of "Transparent" and  
; "TransColor". NOTE: If you press one of the  
hotkeys while the mouse cursor  
; is hovering over a pixel that is invisible as a  
result of TransColor, the window  
; visible beneath that pixel will be acted upon  
instead! Also, Win+G will  
; have an effect only on Windows XP because
```

```
retrieval of transparency settings  
; is not supported by Windows 2000:
```

```
#t:: ; Press Win+T to make the color under the  
mouse cursor invisible.
```

```
MouseGetPos, MouseX, MouseY, MouseWin  
PixelGetColor, MouseRGB, %MouseX%, %MouseY%, RGB  
; In seems necessary to turn off any existing  
transparency first:  
WinSet, TransColor, Off, ahk_id %MouseWin%  
WinSet, TransColor, %MouseRGB% 220, ahk_id  
%MouseWin%  
return
```

```
#o:: ; Press Win+O to turn off transparency for  
the window under the mouse.
```

```
MouseGetPos,,, MouseWin  
WinSet, TransColor, Off, ahk_id %MouseWin%  
return
```

```
#g:: ; Press Win+G to show the current settings  
of the window under the mouse.
```

```
MouseGetPos,,, MouseWin  
WinGet, Transparent, Transparent, ahk_id  
%MouseWin%  
WinGet, TransColor, TransColor, ahk_id %MouseWin%  
ToolTip  
Translucency: `t%Transparent%`nTransColor: `t%TransC  
olor%  
return
```

```
; Examples of "WinSet Region":
```

```
WinSet, Region, 50-0 W200 H250, WinTitle ; Make  
all parts of the window outside this rectangle  
invisible.
```

```
WinSet, Region, 50-0 W200 H250 R40-40, WinTitle ;  
Same as above but with corners rounded to 40x40.
```

```
WinSet, Region, 50-0 W200 H250 E, WinTitle ; An  
ellipse instead of a rectangle.  
WinSet, Region, 50-0 250-0 150-250, WinTitle ; A  
triangle with apex pointing down.  
WinSet, Region,, WinTitle ; Restore the window to  
its original/default display area.
```

```
; Here is a region with a more complex area. It  
creates a see-through rectangular hole inside a  
window.
```

```
; There are two rectangles specified below: an  
outer and an inner. Each rectangle consists of 5  
pairs
```

```
; of X/Y coordinates because the first pair is  
repeated at the end to "close off" each rectangle.
```

```
WinSet, Region, 0-0 300-0 300-300 0-300 0-0 100-  
100 200-100 200-200 100-200 100-100, WinTitle
```

WinSetTitle

Changes the title of the specified 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

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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

A change to a window's title might be merely temporary if the application that owns the window frequently changes the title.

Window titles and text are 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
```

```
WinWaitActive, Untitled - Notepad
```

```
WinSetTitle, This is a new title ; Uses the  
window found above by WinWaitActive
```

WinShow

Unhides the specified window.

```
WinShow [, WinTitle, WinText, ExcludeTitle,  
ExcludeText]
```

Parameters

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). 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

By default, WinShow is the only command that can always detect hidden windows. Other commands can detect them only if [DetectHiddenWindows](#) has been turned on.

This command operates only upon the topmost matching window except when *WinTitle* is [ahk_group GroupName](#), in which case all windows in the group are affected.

Related

WinHide, SetTitleMatchMode, DetectHiddenWindows, Last Found Window

Example

```
Run, notepad.exe
WinWait, Untitled - Notepad
Sleep, 500
WinHide ; Because its parameter is omitted, it
uses the window found above.
Sleep, 1000
WinShow
```

WinWait

Waits until the specified window exists.

```
WinWait [ , WinTitle, WinText, Seconds, ExcludeTitle, ExcludeText ]
```

Parameters

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinTitle may be blank only when *WinText*, *ExcludeTitle*, or *ExcludeText* is present.

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). Hidden text elements are detected if [DetectHiddenText](#) is ON.

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. This parameter can be an [expression](#).

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 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.

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

If another [thread](#) changes the contents of any variable(s) that were used for this command's parameters, the command will not see the change -- it will continue to use the title and text that were originally present in the variables when the command first started waiting.

Window titles and text are case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[WinWaitActive](#), [WinWaitClose](#), [IfWinExist](#), [IfWinActive](#), [Process](#), [SetTitleMatchMode](#), [DetectHiddenWindows](#)

Example

```
Run, notepad.exe
WinWait, Untitled - Notepad, , 3
if ErrorLevel
{
    MsgBox, WinWait timed out.
    return
}
else
    WinMinimize ; Minimize the window found by
WinWait.
```

WinWait[Not]Active

Waits until the specified window is active or not active.

```
WinWaitActive [, WinTitle, WinText, Seconds,  
ExcludeTitle, ExcludeText]  
WinWaitNotActive [, WinTitle, WinText, Seconds,  
ExcludeTitle, ExcludeText]
```

Parameters

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). Hidden text elements are detected if [DetectHiddenText](#) is ON.

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. This parameter can be an [expression](#).

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 the command timed out or 0 otherwise.

Remarks

If a matching window satisfies the command's expectation, the command will not wait for *Seconds* to expire. Instead, it will immediately set [ErrorLevel](#) to 0 and the script will continue executing.

Both `WinWaitActive` and `WinWaitNotActive` will update the [Last Found Window](#) if a qualified window is active when the command begins. In addition, `WinWaitActive` will update the Last Found Window if a qualified window becomes active before the command times out.

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

If another [thread](#) changes the contents of any variable(s) that were used for this command's parameters, the command will not see the change -- it will continue to use the title and text that were originally present in the variables when the command first started waiting.

Window titles and text are case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

Related

[WinWait](#), [WinWaitClose](#), [IfWinExist](#), [IfWinActive](#), [SetTitleMatchMode](#), [DetectHiddenWindows](#)

Example

```
Run, notepad.exe
WinWaitActive, Untitled - Notepad, , 2
if ErrorLevel
{
    MsgBox, WinWait timed out.
    return
}
else
    WinMinimize ; minimize the window found by
```

WinWaitActive.

WinWaitClose

Waits until the specified window does not exist.

```
WinWaitClose [, WinTitle, WinText, Seconds,  
ExcludeTitle, ExcludeText]
```

Parameters

WinTitle

A window title or other criteria identifying the target window. See [WinTitle](#).

WinText

If present, this parameter must be a substring from a single text element of the target window (as revealed by the included Window Spy utility). Hidden text elements are detected if [DetectHiddenText](#) is ON.

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. This parameter can be an [expression](#).

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 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](#).

If another [thread](#) changes the contents of any variable(s) that were used for this command's parameters, the command will not see the change -- it will continue to use the title and text that were originally present in the variables when the command first started waiting.

Window titles and text are case sensitive. Hidden windows are not detected unless [DetectHiddenWindows](#) has been turned on.

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.
```

#AllowSameLineComments

[v1.1.09+]: **This directive was removed.** AutoIt scripts are no longer supported.

Only for AutoIt v2 (.aut) scripts: Allows a comment to appear on the same line as a command.

#AllowSameLineComments

Specifying this directive at the **top** of any AutoIt v2 (.aut) script will enable the use of same-line comments, which are normally disabled for compatibility reasons. If not used at the top of the script, same-line comments are not supported above the point where the directive occurs.

Example

```
#AllowSameLineComments  
Sleep, 1 ; This comment is a same-line comment.
```

#ClipboardTimeout

Changes how long the script keeps trying to access the clipboard when the first attempt fails.

```
#ClipboardTimeout Milliseconds
```

Parameters

Milliseconds

The length of the interval in milliseconds. Specify -1 to have it keep trying indefinitely. Specify 0 to have it try only once. Scripts that do not contain this directive use a 1000 ms timeout.

Remarks

Some applications keep the clipboard open for long periods of time, perhaps to write or read large amounts of data. In such cases, increasing this setting causes the script to wait longer before giving up and displaying an error message.

This settings applies to all [clipboard](#) operations, the simplest of which are the following examples: `Var = %Clipboard%` and `Clipboard = New Text`.

Whenever the script is waiting for the clipboard to become available, new [threads](#) cannot be launched and [timers](#) will not run. However, if the user presses a [hotkey](#), selects a [custom menu item](#), or performs a [GUI action](#) such as pressing a button, that event will be buffered until later; in other words, its subroutine will be performed after the clipboard finally becomes available.

This directive does **not** cause the reading of clipboard data to be reattempted if the first attempt fails. Prior to [v1.1.16], it did cause the script to wait until the timeout expired, but in doing so prevented any further data from being retrieved.

Related

Clipboard, Thread

Example

```
#ClipboardTimeout 2000
```

#CommentFlag

Changes the script's comment symbol from semicolon to some other string.

Note: This directive is not recommended for use in new scripts because a) it does more harm than good and b) it is no longer available in [AutoHotkey v2](#).

```
#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 are not loaded into memory when a script is launched, so they do not affect performance).

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++ comment style.
```

#ErrorStdOut

Sends any syntax error that prevents a script from launching to **stderr** rather than displaying a dialog.

#ErrorStdOut

[AHK_L 42+]: Errors are written to stderr instead of stdout. The command prompt and fancy editors usually display both. This change was undocumented until after [v1.1.19.01].

This allows fancy editors such as TextPad, SciTE, Crimson, and EditPlus to jump to the offending line when a syntax error occurs. Since the #ErrorStdOut directive would have to be added to every script, it is usually better to set up your editor to use the [command line switch](#) /ErrorStdOut when launching any AutoHotkey script (see further below for setup instructions).

Because AutoHotkey is not a console program, errors will not appear at the command prompt directly. Instead, such output can be captured via piping or redirection. For example:

```
"C:\Program Files\AutoHotkey\AutoHotkey.exe"  
/ErrorStdOut "My Script.ahk" 2>&1 |more  
"C:\Program Files\AutoHotkey\AutoHotkey.exe"  
/ErrorStdOut "My Script.ahk" 2>"Syntax-Error  
Log.txt"
```

You can also pipe the output directly to the clipboard by downloading [cb.zip](#) (4 KB) and then following this example:

```
"C:\Program Files\AutoHotkey\AutoHotkey.exe"  
/ErrorStdOut "My Script.ahk" 2>&1 |cb.exe
```

Note: `2>&1` causes stderr to be redirected to stdout, while `2>Filename` redirects only stderr to a file.

Instructions for specific editors

EditPlus:

From the menu bar, select Tools > Configure User Tools.

Press button: Add Tool > Program

Menu Text: Your choice

Command: C:\Program Files\AutoHotkey\AutoHotkey.exe

Argument: /ErrorStdOut "\$(FilePath)"

Initial directory: \$(FileDir)

Capture output: Yes

TextPad:

From the menu bar, select Configure > Preferences.

Expand the Tools entry.

Press the Add button and select "Program".

Copy and paste (adjust to your path): `C:\Windows\System32\cmd.exe --` then press OK.

Triple-click the newly added item (`cmd.exe`) in the ListBox and rename it to your choice (e.g. Launch Script).

Press Apply.

Select the new item in the tree at the left and enter the following information:

Command (should already be filled in): `cmd.exe` (or the full path to it)

Parameters (adjust to your path, if necessary): `/c ""C:\Program Files\AutoHotkey\AutoHotkey.exe" /ErrorStdOut "$File""`

Initial folder: `$FileDir`

Check the following boxes: 1) Run minimized; 2) Capture output.

Press OK. The newly added item should now exist in the Tools menu.

Related

[FileAppend](#) (because it can also send text to stdout)

Example

#ErrorStdOut

#EscapeChar (and explanation of escape sequences)

Changes the script's escape character (e.g. accent vs. backslash).

Note: This directive is not recommended for use in new scripts because a) it does more harm than good and b) it is no longer available in [AutoHotkey v2](#).

```
#EscapeChar NewChar
```

Parameters

NewChar

- Specify a single character.

Remarks

The escape character is used to indicate that the character immediately following it should be interpreted differently than it normally would.

The default escape character is accent/backtick (`), which is at the upper left corner of most English keyboards. Using this character rather than backslash avoids the need for double backslashes in file paths.

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

`,`	<p>command do not need to be escaped because the program knows to treat them literally (but it is best to escape them anyway, for clarity). The same is true for all parameters of MsgBox because it has smart comma handling.</p>
`%`	% (literal percent)
``	` (literal accent; i.e. two consecutive escape characters result in a single literal character)
`;`	<p>; (literal semicolon).</p> <p>Note: It is not necessary to escape a semicolon which has any character <u>other than</u> space or tab to its immediate left, since it would not be interpreted as a comment anyway.</p>
`::`	:: (literal pair of colons). [v1.0.40+]: It is no longer necessary to escape these, except when used literally in a hotstring's replacement text.
`\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  
`\a`	alert (bell) -- corresponds to Ascii value 7. It can also be manifest in some applications by typing Control  
`\f`	formfeed -- corresponds to Ascii value 12. It can also be manifest in some applications by typing Control  
	When the Send command or Hotstrings are used in their default (non-raw) mode, characters such as <code>{ } ^ ! + #</code> have special meaning.

Send	Therefore, to use them literally in these cases, enclose them in braces. For example: <code>Send {^}{!}{{}}</code> .
""	Within an expression , two consecutive quotes enclosed inside a literal string resolve to a single literal quote. For example: <code>Var := "The color ""red"" was found."</code> .

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.
```

Note: These directives are not recommended for use in new scripts because a) they do more harm than good and b) they are no longer available in [AutoHotkey v2](#).

Example

```
#EscapeChar \ ; Change it to be backslash instead
of the default of accent (`).
```

#MaxMem

Sets 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.

This setting restricts only the automatic expansion that a variable does on its own. It does not affect [VarSetCapacity](#).

Related

[VarSetCapacity](#), [Variables](#), [Sort](#), [WinGetText](#), [ControlGetText](#), [#MaxThreads](#)

Example

```
#MaxMem 256 ; Allow 256 MB per variable.
```

#NoEnv [v1.0.43.08+]

Avoids checking empty variables to see if they are environment variables (recommended for all new scripts).

#NoEnv

Specifying the line `#NOENV` anywhere in a script prevents empty variables from being looked up as potential environment variables. For example:

```
#NoEnv
MsgBox %WinDir%
```

The above would **not** retrieve the "WinDir" environment variable (though that could be solved by doing `WinDir := A_WinDir` near the top of the script).

Specifying `#NOENV` is recommended for all new scripts because:

1. It significantly improves performance whenever empty variables are used in an expression or command. It also improves `DllCall`'s performance when unquoted parameter types are used (e.g. `int` vs. `"int"`).
2. It prevents script bugs caused by environment variables whose names unexpectedly match variables used by the script.
3. [AutoHotkey v2](#) will make this behavior the default.

To help ease the transition to `#NoEnv`, the built-in variables `ComSpec` and `ProgramFiles` have been added. They contain the same strings as the corresponding environment variables.

When `#NoEnv` is in effect, the script should use `EnvGet` to retrieve environment variables, or use built-in variables like `A_WinDir`.

Related

[EnvGet](#), [ComSpec](#), [ProgramFiles](#), [A_WinDir](#)

#NoTrayIcon

Disables the showing of a [tray icon](#).

```
#NoTrayIcon
```

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). For 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 [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

[Tray Icon](#), [Menu](#), [ExitApp](#)

Example

```
#NoTrayIcon
```

#Persistent

Keeps a script permanently running (that is, until the user closes it or [ExitApp](#) is encountered).

```
#Persistent
```

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 [OnMessage\(\)](#) or [Gui](#).

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](#).

[v1.0.16+]: This directive also makes a script single-instance. To override this or change the way single-instance behaves, see [#SingleInstance](#).

Related

[#SingleInstance](#), [SetTimer](#), [Menu](#), [Exit](#), [ExitApp](#)

Example

```
#Persistent
```

#SingleInstance

Determines whether a script is allowed to run again when it is already running.

```
#SingleInstance [force|ignore|off]
```

Parameters

force|ignore|off

This parameter determines what happens when a script is launched while a previous instance of itself is already running:

The word FORCE skips the dialog box and replaces the old instance automatically, which is similar in effect to the [Reload](#) command.

The word IGNORE skips the dialog box and leaves the old instance running. In other words, attempts to launch an already-running script are ignored.

The word OFF allows multiple instances of the script to run concurrently.

If the parameter is omitted, a dialog box is displayed asking whether to keep the old instance or replace it with the new one.

Remarks

A script containing [hotkeys](#), [hotstrings](#), [#Persistent](#), [OnMessage\(\)](#), or [Gui](#) is single-instance (dialog & prompt) by default. Other scripts default to allowing multiple instances. This behavior can be disabled or modified as described above.

This directive is ignored when any of the following [command line switches](#) are used: /force /f /restart /r

Related

Reload, #Persistent

Example

```
#SingleInstance force  
#SingleInstance ignore  
#SingleInstance off
```

#Warn [v1.0.95.00+]

Enables or disables warnings for specific conditions which may indicate an error, such as a typo or missing "global" declaration.

```
#Warn [WarningType, WarningMode]
```

Parameters

WarningType

The type of warning to enable or disable. If omitted, it defaults to *All*.

UseUnsetLocal or **UseUnsetGlobal**: Warn when a variable is read without having previously been assigned a value or initialized with [VarSetCapacity](#). If the variable is intended to be empty, assign an empty string to suppress this warning.

This is split into separate warning types for locals and globals because it is more common to use a global variable without prior initialization, due to their persistent and script-wide nature. For this reason, some script authors may wish to enable this type of warning for locals but disable it for globals.

```
#Warn
;y := "" ; This would suppress the
warning.
x := y ; y hasn't been assigned a value.
```

UseEnv: Warn when an environment variable is automatically used in place of an empty script variable. This sometimes occurs when an environment variable's name unexpectedly matches a variable used by the script. This warning occurs when the variable is accessed, but never occurs if the script enables [#NoEnv](#) (recommended for multiple reasons).

```

#Warn
;#NoEnv                ; Add this if "temp" is
not intended to be an environment variable.
;EnvGet temp, TEMP    ; This would copy the
environment variable's value into the
script variable.
temp := ""            ; Despite this line,
temp still seems to have a value.
MsgBox % temp        ; This accesses the
environment variable named "TEMP".

```

LocalSameAsGlobal: Before the script starts to run, display a warning for each *undeclared* local variable which has the same name as a global variable. This is intended to prevent errors caused by forgetting to declare a global variable inside a function before attempting to access it. If the variable really was intended to be local, a declaration such as `local x` or `static y` can be used to suppress the warning. This warning is never shown for variables inside a [force-local](#) function.

```

#Warn
g := 1
ShowG() {              ; The warning is displayed
even if the function is never called.
    ;global g          ; <-- This is required to
access the global variable.
    MsgBox % g        ; Without the declaration,
"g" is an empty local variable.
}

```

ClassOverwrite [v1.1.27+]: Before the script starts to run, show a warning for each assignment targeting a class variable. For example, `box := new BOX` will show a warning if *Box* is a class, since this would overwrite the class (within the super-global variable, *Box*). Warnings are also shown for output variables of commands, but not `ByRef` parameters. Warnings are not shown for nested classes or dynamic variable references.

All: Apply the given *WarningMode* to all supported warning types.

WarningMode

A value indicating how warnings should be delivered. If omitted, it defaults to *MsgBox*.

MsgBox: Show a message box describing the warning. Note that once the message box is dismissed, the script will continue as usual.

StdOut [v1.1.04+]: Send a description of the warning to *stdout* (the program's standard output stream), along with the filename and line number. This allows fancy editors such as SciTE to capture warnings without disrupting the script - the user can later jump to each offending line via the editor's output pane.

OutputDebug: Send a description of the warning to the debugger for display. If a debugger is not active, this will have no effect. For more details, see [OutputDebug](#).

Off: Disable warnings of the given *WarningType*.

Remarks

By default, all warnings are off.

Warnings can't be enabled or disabled at run-time; the settings are determined when a script loads. Therefore, the location in the script is not significant (and, like other `#` directives, `#Warn` cannot be executed conditionally).

However, the ordering of multiple `#Warn` directives is significant: the last occurrence that sets a given warning determines the mode for that warning. So, for example, the two statements below have the combined effect of enabling all warnings except `UseEnv`:

```
#Warn All
#Warn UseEnv, Off

EnvSet EnvVar, 1
```

```
x := EnvVar           ; Okay since #NoEnv has not  
been used.  
x := NotAnEnvVar     ; Warning.
```

Related

[Local and Global Variables](#)

Example

```
#Warn All, Off           ; Disable all  
warnings. This is the default state.  
#Warn                   ; Enable every  
type of warning; show each warning in a message  
box.  
#Warn UseUnsetLocal, OutputDebug ; Warn when a  
local variable is used before it's set; send  
warning to OutputDebug.
```

Software License

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

```
Copyright (C) 1989, 1991 Free Software
Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA
02110-1301, USA
```

```
Everyone is permitted to copy and distribute
verbatim copies
of this license document, but changing it is not
allowed.
```

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate

to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable

sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition

files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations

under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program

does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

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 Ahdoot: 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.

Thaddeus Beck (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.

Aurelian Maga: For writing the source code for [ImageSearch](#) and the faster [PixelSearch](#).

Joost Mulders: Whose source code provided the foundation for [expressions](#).

Laszlo Hars: For advice about data structures and algorithms, which helped greatly speed up arrays and dynamic variables.

Marcus Sonntag (Ultra): For the research, design, coding, and testing of [DllCall](#).

Gena Shimanovich: For debugging and coding assistance; and for some advanced prototype scripts upon which future features may be based.

Eric Morin (numEric): For advice on mathematics; for steadfast debugging of areas like [OnMessage](#), floating point computations, and mouse movement; and for improvements to overall quality.

Philip Hazel: For [Perl-Compatible Regular Expressions \(PCRE\)](#).

Titan/polyethene: For providing community hosting on autohotkey.net, creating many useful scripts and libraries, creating the JavaScript to colorize code-comments in the forum, and many other things.

Philippe Lhoste (PhiLho): For tireless moderation and support in the forum, RegEx advice and testing, syntax and design ideas, and many other things.

John Biederman: For greatly improving the presentation and ergonomics of the documentation.

Jonathan Rennison (JGR): For developing [RegisterCallback\(\)](#), and for beneficial suggestions.

Steve Gray (Lexikos): For developing dynamic function calling and other new functionality; analyzing and fixing bugs; and valuable support and advice for hundreds of individual visitors at the forum.

AutoHotkey_L:

jackeiku: For developing Unicode support and other new functionality.

fincs: For developing native 64-bit support and try/catch/throw, writing a replacement for the old compiler, analyzing and fixing bugs.

Sean: For developing built-in COM functionality and providing valuable insight into COM.

TheGood: For merging the documentation and adapting the installer script.

ac: For developing #Warn.

Russell Davis: For developing `A_PriorKey`, `ahk_path` (the basis of `ahk_exe` in `WinTitle` parameters), `#InputLevel` and `SendLevel`.

Christian Sander: For developing support for SysLink controls.

*And to everyone else who's contributed or sent in bug reports or suggestions:
Thanks!*

Clipboard and ClipboardAll

Clipboard is a built-in [variable](#) that reflects the current contents of the Windows clipboard if those contents can be expressed as text. By contrast, *ClipboardAll* contains everything on the clipboard, such as pictures and formatting.

Each line of text on *Clipboard* typically ends with carriage return and linefeed (CR+LF), which can be expressed in the script as ``r`n`. Files (such as those copied from an open Explorer window via **Control** **C** **E**) are considered to be text: They are automatically converted to their filenames (with full path) whenever *Clipboard* is referenced in the script. To extract the files one by one, follow this example:

```
Loop, parse, clipboard, `n, `r
{
    MsgBox, 4, , File number %A_Index% is
%A_LoopField%.`n`nContinue?
    IfMsgBox, No, break
}
```

To arrange the filenames in alphabetical order, use the [Sort](#) command. To write the filenames on the clipboard to a file, use [FileAppend](#), `%clipboard%`r`n, C:\My File.txt`. To change how long the script will keep trying to open the clipboard -- such as when it is in use by another application -- use [#ClipboardTimeout](#).

Basic examples:

```
clipboard = my text ; Give the clipboard
entirely new contents.
clipboard = ; Empty the clipboard.
clipboard = %clipboard% ; Convert any copied
files, HTML, or other formatted text to plain
text.
clipboard = %clipboard% Text to append. ;
```

```
Append some text to the clipboard.  
StringReplace, clipboard, clipboard, ABC, DEF,  
All ; Replace all occurrences of ABC with DEF  
(also converts the clipboard to plain text).
```

Using ClipWait to improve script reliability:

```
clipboard = ; Start off empty to allow ClipWait  
to detect when the text has arrived.  
Send ^c  
ClipWait ; Wait for the clipboard to contain  
text.  
MsgBox Control-C copied the following contents  
to the clipboard:`n`n%clipboard%
```

ClipboardAll (saving and restoring everything on the clipboard)

ClipboardAll contains everything on the clipboard (such as pictures and formatting). It is most commonly used to save the clipboard's contents so that the script can temporarily use the clipboard for an operation. When the operation is completed, the script restores the original clipboard contents as shown below:

```
ClipSaved := ClipboardAll ; Save the entire
clipboard to a variable of your choice.
; ... here make temporary use of the clipboard,
such as for pasting Unicode text via Transform
Unicode ...
Clipboard := ClipSaved ; Restore the original
clipboard. Note the use of Clipboard (not
ClipboardAll).
ClipSaved = ; Free the memory in case the
clipboard was very large.
```

ClipboardAll may also be saved to a file (in this mode, FileAppend always overwrites any existing file):

```
FileAppend, %ClipboardAll%, C:\Company Logo.clip
; The file extension does not matter.
```

To later load the file back onto the clipboard (or into a variable), follow this example:

```
FileRead, Clipboard, *c C:\Company Logo.clip ;
Note the use of *c, which must precede the
filename.
```

Notes

ClipboardAll is blank when used in ways other than those described above. In addition, *ClipboardAll* is not supported inside [comma-separated expressions](#); that is, it should be assigned on a line by itself such as `ClipSaved := ClipboardAll`.

Variables to which *ClipboardAll* has been assigned are in binary format and thus will appear as gibberish when displayed with `MsgBox` or similar. Also, altering a binary-clipboard variable (by means such as `StringReplace`) will revert it to a normal variable, resulting in the loss of its clipboard data. [v1.0.46+]: Binary-clipboard variables may be passed to [functions](#) by value (formerly they only worked [ByRef](#)).

If *ClipboardAll* cannot retrieve one or more of the data objects (formats) on the clipboard, they will be omitted but all the remaining objects will be stored.

A variable containing clipboard data can be copied to another variable as in this example: `ClipSaved2 := ClipSaved`.

`ClipWait` may be used to detect when the clipboard contains data (optionally including non-text data).

`StrLen / StringLen` may be used to discover the total size of a variable to which *ClipboardAll* has been assigned. However, to get the size in bytes on Unicode versions of AutoHotkey, the length must be multiplied by 2. `A_IsUnicode` can be used to support ANSI and Unicode versions, as in this example: `size := StrLen(ClipSaved) * (A_IsUnicode ? 2 : 1)`.

Variables to which *ClipboardAll* has been assigned can be compared to each other (but not directly to *ClipboardAll*) by means of the `<>` and `=` operators. In the following example, the length of each variable is checked first. If that is not enough to make the determination, the contents are compared to break the tie:

```
if ClipSaved1 <> %ClipSaved2%    ; This must be
an old-style IF statement, not an expression.
    MsgBox The two saved clipboards are
different.
```

Saving *ClipboardAll* to a variable is not restricted by the memory limit set by

#MaxMem.

A saved clipboard file internally consists of a four-byte format type, followed by a four-byte data-block size, followed by the data-block for that format. If the clipboard contained more than one format (which is almost always the case), these three items are repeated until all the formats are included. The file ends with a four-byte format type of 0.

Known limitation: Retrieving *ClipboardAll* while cells from Microsoft Excel are on the clipboard may cause Excel to display a "no printers" dialog.

Clipboard utilities written in AutoHotkey:

- Deluxe Clipboard: Provides unlimited number of private, named clipboards to Copy, Cut, Paste, Append or CutAppend of selected text.
www.autohotkey.com/forum/topic2665.html

- ClipStep: Control multiple clipboards using only the keyboard's

   www.autohotkey.com/forum/topic4836.html

OnClipboardChange

Scripts can detect changes to the content of the Clipboard by using [OnClipboardChange](#).

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.

The following list contains all commands, functions and GUI/Control events which set ErrorLevel, including the possible ErrorLevel values:

ClipWait	0, 1
Control	0, 1
ControlClick	0, 1
ControlFocus	0, 1
ControlGet	0, 1
ControlGetFocus	0, 1
ControlGetText	0, 1
ControlMove	0, 1
ControlSend / ControlSendRaw	0, 1

ControlSetText	0, 1
DllCall()	0, -1, -2, -3, -4, <i>n</i> , <i>An</i>
Drive	0, 1
DriveGet	0, 1
EnvSet	0, 1
EnvUpdate	0, 1
FileAppend	0, 1
FileCopy	0, <i>n</i>
FileCopyDir	0, 1
FileCreateDir	0, 1
FileCreateShortcut	0, 1
FileDelete	0, <i>n</i>
FileGetAttrib	0, 1
FileGetShortcut	0, 1
FileGetSize	0, 1
FileGetTime	0, 1
FileGetVersion	0, 1
FileInstall	0, 1
FileMove	0, <i>n</i>
FileMoveDir	0, 1
FileRead	0, 1
FileReadLine	0, 1
FileRecycle	0, 1
FileRecycleEmpty	0, 1
FileRemoveDir	0, 1

FileSelectFile	0, 1
FileSelectFolder	0, 1
FileSetAttrib	0, <i>n</i>
FileSetTime	0, 1
GroupActivate	0, 1
Gui: GuiSize event	0, 1, 2
Gui: GuiDropFiles event	0, <i>n</i>
GuiControl	0, 1
GuiControlGet	0, 1
Gui control: Link click event	<i>string</i>
Gui control: Custom event	0, <i>n</i>
Gui control: ListView item change event	subset of "SsFfCc"
Hotkey	0, 1, 2, 3, 4, 5, 6, 98, 99
ImageSearch	0, 1, 2
IniDelete	0, 1
IniWrite	0, 1
Input	0, 1, NewInput, Max, Timeout, Match, EndKey:Name
InputBox	0, 1, 2
KeyWait	0, 1
Menu	0, 1
PixelGetColor	0, 1
PixelSearch	0, 1, 2
PostMessage	0, 1

Process	0, <i>PID</i>
RegDelete	0, 1
RegExMatch()	0, <i>string</i> , <i>-n</i>
RegExReplace()	0, <i>string</i> , <i>-n</i>
RegRead	0, 1
RegWrite	0, 1
Run	0, ERROR
RunWait	<i>n</i> , ERROR
SendMessage	FAIL, <i>n</i>
SetWorkingDir	0, 1
Sort: U option	0, <i>n</i>
SoundGet	0, <i>string</i>
SoundGetWaveVolume	0, 1
SoundPlay	0, 1
SoundSet	0, <i>string</i>
SoundSetWaveVolume	0, 1
StatusBarGetText	0, 1
StatusBarWait	0, 1, 2
StringGetPos	0, 1
StringReplace	0, 1, <i>n</i>
UrlDownloadToFile	0, 1
WinGetText	0, 1
WinMenuItemSelect	0, 1
WinSet Style/ExStyle/Region	0, 1
WinWait	0, 1

WinWaitActive / WinWaitNotActive	0, 1
WinWaitClose	0, 1

Example

```
WinWait, MyWindow, , 1
if ErrorLevel > 0 ; i.e. it's not blank or zero.
    MsgBox, The window does not exist.
else
    MsgBox, The window exists.
```

Creating a Keyboard Macro or Mouse Macro

A macro is a series of scripted actions that is "played" upon demand. The most common activity of a macro is to send [simulated keystrokes](#) and [mouse clicks](#) to one or more windows. Such windows respond to each keystroke and mouse click as though you had performed it manually, which allows repetitive tasks to be automated with high speed and reliability.

Although macros can be written by hand, you might find it easier to write long ones with the aid of a macro recorder such as the [Recorder script](#) written by Titan/polyethene. It watches what you type and where you click, and keeps track of which window is [active](#). It transcribes these actions into a working macro that can later be "played back" at a faster speed.

One of the most convenient ways to play back a macro is to assign it to a [hotkey](#) or [hotstring](#). For example, the following hotkey would create an empty e-mail message and prepare it for a certain type recipient, allowing you to personalize it prior to sending:

```
^!s:: ; Control+Alt+S hotkey.
IfWinNotExist Inbox - Microsoft Outlook
    return ; Outlook isn't open to the right
section, so do nothing.
WinActivate ; Activate the window found by the
above command.
Send ^n ; Create new/blank e-mail via
Control+N.
WinWaitActive Untitled Message
Send {Tab 2}Product Recall for ACME Rocket
Skates ; Set the subject line.
Send {Tab}Dear Sir or Madam,{Enter 2}We have
recently discovered a minor defect ... ; etc.
return ; This line serves to finish the hotkey.
```

Hotkey macros like the above are especially useful for tasks you perform several times per day. By contrast, macros used less often can each be kept in a separate script accessible by means of a shortcut in the Start Menu or on the desktop.

To start creating your own macros and hotkeys right away, please read the [Quick-start Tutorial](#).

[-- Home --](#)

Overriding or Disabling Hotkeys

You can disable all built-in Windows hotkeys except **Win** **L** **Win** **U** making the following change to the registry (this should work on all OSes but a 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.

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 the next example, the **Win** **R** key, which is used to open the RUN window, is completely disabled:

```
#r::return
```

Similarly, to disable both **Win** 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** only for Notepad, leaving the key in effect for all other types of windows:

```
$^p::  
IfWinActive ahk_class Notepad  
    return ; i.e. do nothing, which causes  
Control-P to do nothing in Notepad.  
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). See also: [context-sensitive hotkeys](#).

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.

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](#)

```
/*
0-----
-----0
|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. |
0-----|
-----0
*/

;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 NumpadDiv would send sometimes "/"
to the
;screen.
#InstallKeybdHook

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 reaches 1.
MouseRotationAnglePart /= 45

MouseCurrentAccelerationSpeed = 0
MouseCurrentSpeed = %MouseSpeed%
```

```
MouseWheelCurrentAccelerationSpeed = 0  
MouseWheelCurrentSpeed = %MouseSpeed%
```

```
SetKeyDelay, -1  
SetMouseDelay, -1
```

```
Hotkey, *Numpad0, ButtonLeftClick  
Hotkey, *NumpadIns, ButtonLeftClickIns  
Hotkey, *Numpad5, ButtonMiddleClick  
Hotkey, *NumpadClear, ButtonMiddleClickClear  
Hotkey, *NumpadDot, ButtonRightClick  
Hotkey, *NumpadDel, 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, Numpad8, 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
```

```
If ScrollLockState = D
```

```
{
```

```
    Hotkey, *Numpad0, On
```

```
    Hotkey, *NumpadIns, On
```

```
    Hotkey, *Numpad5, On
```

```
    Hotkey, *NumpadDot, On
```

```
    Hotkey, *NumpadDel, On
```

```
    Hotkey, *NumpadDiv, On
```

```
    Hotkey, *NumpadMult, 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, Numpad8, 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, Numpad8, Off
Hotkey, Numpad2, Off
Hotkey, Numpad7, Off
Hotkey, Numpad1, Off
Hotkey, Numpad9, Off
Hotkey, Numpad3, Off
```

```
Hotkey, Numpad6, Off
Hotkey, Numpad4, Off
```

```
Hotkey, !Numpad8, 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
Goto ButtonClickStart
```

```
ButtonLeftClickIns:  
GetKeyState, already_down_state, LButton  
If already_down_state = D  
    return  
Button2 = NumpadIns  
ButtonClick = Left  
Goto 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:
MouseClicked, %ButtonClick%,,, 1, 0, D
SetTimer, ButtonClickEnd, 10
return
```

```
ButtonClickEnd:
GetKeyState, kclickstate, %Button2%, P
if kclickstate = D
    return
```

```
SetTimer, ButtonClickEnd, Off
MouseClicked, %ButtonClick%,,, 1, 0, U
return
```

;Mouse movement support

```
ButtonSpeedUp:
MouseSpeed++
ToolTip, Mouse speed: %MouseSpeed% pixels
SetTimer, RemoveToolTip, 1000
```

```
return
ButtonSpeedDown:
If MouseSpeed > 1
    MouseSpeed--
If 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:
If MouseAccelerationSpeed > 1
    MouseAccelerationSpeed--
If 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:
If MouseMaxSpeed > 1
    MouseMaxSpeed--
```

```
If MouseMaxSpeed = 1
    ToolTip, Mouse maximum speed: %MouseMaxSpeed%
pixel
else
    ToolTip, Mouse maximum speed: %MouseMaxSpeed%
pixels
SetTimer, RemoveToolTip, 1000
return
```

```
ButtonRotationAngleUp:
MouseRotationAnglePart++
If MouseRotationAnglePart >= 8
    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:

;MouseRotationAngle conversion to speed of 90
degrees to right
{
    MouseCurrentSpeedToSide = %MouseRotationAngle%
    MouseCurrentSpeedToSide /= 90.0
    Temp = %MouseCurrentSpeedToSide%
    Transform, Temp, mod, %Temp%, 4

    if Temp >= 0
    {
        if Temp < 1
        {
            MouseCurrentSpeedToSide = 0
            MouseCurrentSpeedToSide += %Temp%
            Goto
EndMouseCurrentSpeedToSideCalculation
        }
    }
    if Temp >= 1
    {
        if Temp < 2
        {
            MouseCurrentSpeedToSide = 1
            Temp -= 1
            MouseCurrentSpeedToSide -= %Temp%
            Goto
EndMouseCurrentSpeedToSideCalculation
        }
    }
    if Temp >= 2
    {
        if Temp < 3
        {

```

```

        MouseCurrentSpeedToSide = 0
        Temp -= 2
        MouseCurrentSpeedToSide -= %Temp%
        Goto
EndMouseCurrentSpeedToSideCalculation
    }
}
if Temp >= 3
{
    if Temp < 4
    {
        MouseCurrentSpeedToSide = -1
        Temp -= 3
        MouseCurrentSpeedToSide += %Temp%
        Goto
EndMouseCurrentSpeedToSideCalculation
    }
}
}
EndMouseCurrentSpeedToSideCalculation:

MouseCurrentSpeedToDirection *=
%MouseCurrentSpeed%
MouseCurrentSpeedToSide *= %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
    MouseMove, %Temp%, %Temp2%, 0, R
}
else if Button = NumpadPgUp
{
    Temp = %MouseCurrentSpeedToDirection%
    Temp += %MouseCurrentSpeedToSide%
    Temp2 = %MouseCurrentSpeedToDirection%
    Temp2 -= %MouseCurrentSpeedToSide%
    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, kstate, %Button%, P  
if kstate = 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  
If 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  
If MouseWheelSpeedMultiplier <= 0  
    MouseWheelSpeedMultiplier = 1
```

```
If MouseWheelSpeedReal >
%MouseWheelSpeedMultiplier%
{
    MouseWheelSpeed--
    MouseWheelSpeedReal = %MouseWheelSpeed%
    MouseWheelSpeedReal *=
%MouseWheelSpeedMultiplier%
}
```

```
If 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
If 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
If MouseWheelSpeedMultiplier <= 0
    MouseWheelSpeedMultiplier = 1
If MouseWheelAccelerationSpeed > 1
```

```

{
    MouseWheelAccelerationSpeed--
    MouseWheelAccelerationSpeedReal =
%MouseWheelAccelerationSpeed%
    MouseWheelAccelerationSpeedReal *=
%MouseWheelSpeedMultiplier%
}
If 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
If 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
If MouseWheelSpeedMultiplier <= 0
    MouseWheelSpeedMultiplier = 1
If MouseWheelMaxSpeed > 1
{
    MouseWheelMaxSpeed--

```

```

        MouseWheelMaxSpeedReal = %MouseWheelMaxSpeed%
        MouseWheelMaxSpeedReal *=
%MouseWheelSpeedMultiplier%
    }
    If MouseWheelMaxSpeedReal = 1
        ToolTip, Mouse wheel maximum speed:
%MouseWheelMaxSpeedReal% line
    else
        ToolTip, Mouse wheel maximum speed:
%MouseWheelMaxSpeedReal% lines
        SetTimer, RemoveToolTip, 1000
        return

ButtonWheelUp:
ButtonWheelDown:

If Button <> 0
{
    If Button <> %A_ThisHotkey%
    {
        MouseWheelCurrentAccelerationSpeed = 0
        MouseWheelCurrentSpeed = %MouseWheelSpeed%
    }
}
StringReplace, Button, A_ThisHotkey, *

ButtonWheelAccelerationStart:
If 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 = D
    Goto ButtonWheelAccelerationStart

MouseWheelCurrentAccelerationSpeed = 0
MouseWheelCurrentSpeed = %MouseWheelSpeed%
Button = 0
return

RemoveToolTip:
SetTimer, RemoveToolTip, Off
ToolTip
return
```

Remapping a Joystick to Keyboard or Mouse

Table of Contents

- [Important Notes](#)
- [Making a Joystick Button Send Keystrokes or Mouse Clicks](#)
- [Making a Joystick Axis or POV Hat Send Keystrokes or Mouse Clicks](#)
- [Remarks](#)

Important Notes

- Although a joystick button or axis can be remapped to become a key or mouse button, it cannot be remapped to some other joystick button or axis. That would be possible only with the help of a joystick emulator such as [vJoy](#).
- AutoHotkey identifies each button on a joystick with a unique number between 1 and 32. To determine these numbers, use the [joystick test script](#).

Making a Joystick Button Send Keystrokes or Mouse Clicks

Below are three approaches, starting at the simplest and ending with the most complex. The most complex method works in the broadest variety of circumstances (such as games that require a key or mouse button to be held down).

Method #1: This method sends simple keystrokes and mouse clicks. For example:

```
Joy1::Send {Left} ; Have button #1 send a left-  
arrow keystroke.  
Joy2::Click ; Have button #2 send a click of  
left mouse button.  
Joy3::Send a{Esc}{Space}{Enter} ; Have button  
#3 send the letter "a" followed by Escape,  
Space, and Enter.  
Joy4::Send Sincerely,{Enter}John Smith ; Have  
button #4 send a two-line signature.
```

To have a button perform more than one command, put the first command *beneath* the button name and make the last command a [return](#). For example:

```
Joy5::  
Run Notepad  
WinWait Untitled - Notepad  
WinActivate  
Send This is the text that will appear in  
Notepad.{Enter}  
return
```

See the [Key List](#) for the complete list of keys and mouse/joystick buttons.

Method #2: This method is necessary in cases where a key or mouse button must be held down for the entire time that you're holding down a joystick button. The following example makes the joystick's second button become the left-arrow key:

```
Joy2::  
Send {Left down} ; Hold down the left-arrow  
key.  
KeyWait Joy2 ; Wait for the user to release the  
joystick button.  
Send {Left up} ; Release the left-arrow key.  
return
```

Method #3: This method is necessary in cases where you have more than one joystick hotkey of the type described in Method #2, and you sometimes press and release such hotkeys simultaneously. The following example makes the joystick's third button become the left mouse button:

```
Joy3::  
Send {LButton down} ; Hold down the left mouse  
button.  
SetTimer, WaitForButtonUp3, 10  
return  
  
WaitForButtonUp3:  
if GetKeyState("Joy3") ; The button is still,  
down, so keep waiting.  
return  
; Otherwise, the button has been released.  
Send {LButton up} ; Release the left mouse  
button.  
SetTimer, WaitForButtonUp3, Off  
return
```

Auto-repeating a keystroke: Some programs or games might require a key to be sent repeatedly (as though you are holding it down on the keyboard). The following example achieves this by sending spacebar keystrokes repeatedly while you hold down the joystick's second button:

```
Joy2::
Send {Space down} ; Press the spacebar down.
SetTimer, WaitForJoy2, 30 ; Reduce the number
30 to 20 or 10 to send keys faster. Increase it
to send slower.
return

WaitForJoy2:
if not GetKeyState("Joy2") ; The button has
been released.
{
    Send {Space up} ; Release the spacebar.
    SetTimer, WaitForJoy2, Off ; Stop
monitoring the button.
    return
}
; Since above didn't "return", the button is
still being held down.
Send {Space down} ; Send another Spacebar
keystroke.
return
```

Context-sensitive Joystick Buttons: The directives [#IfWinActive/Exist](#) can be used to make selected joystick buttons perform a different action (or none at all) depending on the type of window that is active or exists.

Using a Joystick as a Mouse: The [Joystick-To-Mouse script](#) converts a joystick into a mouse by remapping its buttons and axis control.

Making a Joystick Axis or POV Hat Send Keystrokes or Mouse Clicks

To have a script respond to movement of a joystick's axis or POV hat, use [SetTimer](#) and [GetKeyState](#). The following example makes the joystick's X and Y axes behave like the arrow key cluster on a keyboard (left, right, up, and down):

```
#Persistent ; Keep this script running until
the user explicitly exits it.
SetTimer, WatchAxis, 5
return

WatchAxis:
GetKeyState, JoyX, JoyX ; Get position of X
axis.
GetKeyState, JoyY, JoyY ; Get position of Y
axis.
KeyToHoldDownPrev = %KeyToHoldDown% ; Prev now
holds the key that was down before (if any).

if JoyX > 70
    KeyToHoldDown = Right
else if JoyX < 30
    KeyToHoldDown = Left
else if JoyY > 70
    KeyToHoldDown = Down
else if JoyY < 30
    KeyToHoldDown = Up
else
    KeyToHoldDown =

if KeyToHoldDown = %KeyToHoldDownPrev% ; The
correct key is already down (or no key is
needed).
    return ; Do nothing.
```

```

; Otherwise, release the previous key and press
down the new key:
SetKeyDelay -1 ; Avoid delays between
keystrokes.
if KeyToHoldDownPrev ; There is a previous key
to release.
    Send, {%KeyToHoldDownPrev% up} ; Release
it.
if KeyToHoldDown ; There is a key to press
down.
    Send, {%KeyToHoldDown% down} ; Press it
down.
return

```

The following example makes the joystick's POV hat behave like the arrow key cluster on a keyboard; that is, the POV hat will send arrow keystrokes (left, right, up, and down):

```

#Persistent ; Keep this script running until
the user explicitly exits it.
SetTimer, WatchPOV, 5
return

WatchPOV:
GetKeyState, POV, JoyPOV ; Get position of the
POV control.
KeyToHoldDownPrev = %KeyToHoldDown% ; Prev now
holds the key that was down before (if any).

; Some joysticks might have a smooth/continuous
POV rather than one in fixed increments.
; To support them all, use a range:
if POV < 0 ; No angle to report
    KeyToHoldDown =

```

```

else if POV > 31500 ; 315 to 360
degrees: Forward
    KeyToHoldDown = Up
else if POV between 0 and 4500 ; 0 to 45
degrees: Forward
    KeyToHoldDown = Up
else if POV between 4501 and 13500 ; 45 to 135
degrees: Right
    KeyToHoldDown = Right
else if POV between 13501 and 22500 ; 135 to 225
degrees: Down
    KeyToHoldDown = Down
else ; 225 to 315
degrees: Left
    KeyToHoldDown = Left

if KeyToHoldDown = %KeyToHoldDownPrev% ; The
correct key is already down (or no key is
needed).
    return ; Do nothing.

; Otherwise, release the previous key and press
down the new key:
SetKeyDelay -1 ; Avoid delays between
keystrokes.
if KeyToHoldDownPrev ; There is a previous key
to release.
    Send, {%KeyToHoldDownPrev% up} ; Release
it.
if KeyToHoldDown ; There is a key to press
down.
    Send, {%KeyToHoldDown% down} ; Press it
down.
return

```

Auto-repeating a keystroke: Both examples above can be modified to send the key repeatedly rather than merely holding it down (that is, they can mimic physically holding down a key on the keyboard). To do this, replace the following line:

```
return ; Do nothing.
```

WITH:

```
{
    if KeyToHoldDown
        Send, {%KeyToHoldDown% down} ; Auto-
repeat the keystroke.
    return
}
```

Remarks

A joystick other than first may be used by preceding the button or axis name with the number of the joystick. For example, `2Joy1` would be the second joystick's first button.

To find other useful joystick scripts, visit the [AutoHotkey forum](#). A keyword search such as *Joystick and GetKeyState and Send* is likely to produce topics of interest.

Related Topics

[Joystick-To-Mouse script \(using a joystick as a mouse\)](#)

[List of joystick buttons, axes, and controls](#)

[GetKeyState](#)

[Remapping the keyboard and mouse](#)

Joystick Test Script

This script helps determine the button numbers and other attributes 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](#)

```
; July 6, 2005: Added auto-detection of joystick
number.
; May 8, 2005 : Fixed: JoyAxes is no longer
queried as a means of
; detecting whether the joystick is connected.
Some joysticks are
; gamepads and don't have even a single axis.

; If you want to unconditionally use a specific
joystick number, change
; the following value from 0 to the number of the
joystick (1-16).
; A value of 0 causes the joystick number to be
auto-detected:
JoystickNumber = 0

; END OF CONFIG SECTION. Do not make changes below
this point unless
; you wish to alter the basic functionality of the
script.

; Auto-detect the joystick number if called for:
if JoystickNumber <= 0
{
    Loop 16 ; Query each joystick number to find
out which ones exist.
```

```

    {
        GetKeyState, JoyName, %A_Index%JoyName
        if JoyName <>
        {
            JoystickNumber = %A_Index%
            break
        }
    }
    if JoystickNumber <= 0
    {
        MsgBox The system does not appear to have
any joysticks.
        ExitApp
    }
}

```

```

#SingleInstance

```

```

SetFormat, float, 03 ; Omit decimal point from
axis position percentages.

```

```

GetKeyState, joy_buttons,
%JoystickNumber%JoyButtons

```

```

GetKeyState, joy_name, %JoystickNumber%JoyName

```

```

GetKeyState, joy_info, %JoystickNumber%JoyInfo

```

```

Loop

```

```

{

```

```

    buttons_down =

```

```

    Loop, %joy_buttons%

```

```

    {

```

```

        GetKeyState, joy%A_Index%,
%JoystickNumber%joy%A_Index%

```

```

        if joy%A_Index% = D

```

```

            buttons_down =

```

```

%buttons_down%%A_Space%%A_Index%

```

```

    }

```

```

    GetKeyState, JoyX, %JoystickNumber%JoyX

```

```

    axis_info = X%JoyX%

```

```

    GetKeyState, JoyY, %JoystickNumber%JoyY

```

```

axis_info =
%axis_info%%A_Space%%A_Space%Y%JoyY%
  IfInString, joy_info, Z
  {
    GetKeyState, JoyZ, %JoystickNumber%JoyZ
    axis_info =
%axis_info%%A_Space%%A_Space%Z%JoyZ%
  }
  IfInString, joy_info, R
  {
    GetKeyState, JoyR, %JoystickNumber%JoyR
    axis_info =
%axis_info%%A_Space%%A_Space%R%JoyR%
  }
  IfInString, joy_info, U
  {
    GetKeyState, JoyU, %JoystickNumber%JoyU
    axis_info =
%axis_info%%A_Space%%A_Space%U%JoyU%
  }
  IfInString, joy_info, V
  {
    GetKeyState, JoyV, %JoystickNumber%JoyV
    axis_info =
%axis_info%%A_Space%%A_Space%V%JoyV%
  }
  IfInString, joy_info, P
  {
    GetKeyState, joyp, %JoystickNumber%JoyPOV
    axis_info =
%axis_info%%A_Space%%A_Space%POV%joyp%
  }
  ToolTip, %joy_name%
  (#%JoystickNumber%):`n%axis_info`nButtons Down:
%buttons_down`n`n(right-click the tray icon to
exit)
  Sleep, 100

```

```
}  
return
```

Using a Joystick as a Mouse

This script converts a joystick into a three-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 the following value to make the mouse  
cursor move faster:
```

```
JoyMultiplier = 0.30
```

```
; Decrease the following 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 the following to true to invert the Y-  
axis, which causes the mouse to  
; move vertically in the direction opposite the  
stick:
```

```
InvertYAxis := false
```

```
; Change these values to use joystick button  
numbers other than 1, 2, and 3 for  
; the left, right, and middle mouse buttons.  
Available numbers are 1 through 32.
```

```
; Use the Joystick Test Script to find out your  
joystick's numbers more easily.
```

```
ButtonLeft = 1
```

```
ButtonRight = 2
```

```
ButtonMiddle = 3

; If your joystick has a POV control, you can use
it as a mouse wheel. The
; following value is the number of milliseconds
between turns of the wheel.
; Decrease it to have the wheel turn faster:
WheelDelay = 250

; 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

JoystickPrefix = %JoystickNumber%Joy
Hotkey, %JoystickPrefix%%ButtonLeft%, ButtonLeft
Hotkey, %JoystickPrefix%%ButtonRight%, ButtonRight
Hotkey, %JoystickPrefix%%ButtonMiddle%,
ButtonMiddle

; Calculate the axis displacements that are needed
to start moving the cursor:
JoyThresholdUpper := 50 + JoyThreshold
JoyThresholdLower := 50 - JoyThreshold
if InvertYAxis
    YAxisMultiplier = -1
else
    YAxisMultiplier = 1

SetTimer, WatchJoystick, 10 ; Monitor the
movement of the joystick.
```

```
GetKeyState, JoyInfo, %JoystickNumber%JoyInfo  
IfInString, JoyInfo, P ; Joystick has POV  
control, so use it as a mouse wheel.
```

```
SetTimer, MouseWheel, %WheelDelay%
```

```
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
```

```
ButtonMiddle:
```

```
SetMouseDelay, -1 ; Makes movement smoother.
```

```
MouseClick, middle,,, 1, 0, D ; Hold down the  
right mouse button.
```

```
SetTimer, WaitForMiddleButtonUp, 10
```

```
return
```

```
WaitForLeftButtonUp:
if GetKeyState(JoystickPrefix . ButtonLeft)
    return ; The button is still, down, so keep
waiting.
; 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:
if GetKeyState(JoystickPrefix . ButtonRight)
    return ; The button is still, down, so keep
waiting.
; Otherwise, the button has been released.
SetTimer, WaitForRightButtonUp, Off
MouseClick, right,,, 1, 0, U ; Release the mouse
button.
return
```

```
WaitForMiddleButtonUp:
if GetKeyState(JoystickPrefix . ButtonMiddle)
    return ; The button is still, down, so keep
waiting.
; Otherwise, the button has been released.
SetTimer, WaitForMiddleButtonUp, Off
MouseClick, middle,,, 1, 0, U ; Release the mouse
button.
return
```

```
WatchJoystick:
MouseNeedsToBeMoved := false ; Set default.
SetFormat, float, 03
GetKeyState, JoyX, %JoystickNumber%JoyX
GetKeyState, JoyY, %JoystickNumber%JoyY
if JoyX > %JoyThresholdUpper%
```

```

{
    MouseNeedsToBeMoved := true
    DeltaX := JoyX - JoyThresholdUpper
}
else if JoyX < %JoyThresholdLower%
{
    MouseNeedsToBeMoved := true
    DeltaX := JoyX - JoyThresholdLower
}
else
    DeltaX = 0
if JoyY > %JoyThresholdUpper%
{
    MouseNeedsToBeMoved := true
    DeltaY := JoyY - JoyThresholdUpper
}
else if JoyY < %JoyThresholdLower%
{
    MouseNeedsToBeMoved := true
    DeltaY := JoyY - JoyThresholdLower
}
else
    DeltaY = 0
if MouseNeedsToBeMoved
{
    SetMouseDelay, -1 ; Makes movement smoother.
    MouseMove, DeltaX * JoyMultiplier, DeltaY *
JoyMultiplier * YAxisMultiplier, 0, R
}
return

MouseWheel:
GetKeyState, JoyPOV, %JoystickNumber%JoyPOV
if JoyPOV = -1 ; No angle.
    return
if (JoyPOV > 31500 or JoyPOV < 4500) ; Forward
    Send {WheelUp}

```

```
else if JoyPOV between 13500 and 22500 ; Back  
    Send {WheelDown}  
return
```

WinLIRC Client

This script receives notifications from [WinLIRC](#) whenever you press a button on your remote control. It can be used to automate Winamp, Windows Media Player, etc. It's easy to configure. For example, if WinLIRC recognizes a button named "VolUp" on your remote control, create a label named VolUp and beneath it use the command `SoundSet +5` to increase the soundcard's volume by 5%.

[Download This Script](#) | [Other Sample Scripts](#) | [Home](#)

```
; Here are the steps to use this script:
; 1) Configure WinLIRC to recognize your remote
control and its buttons.
;   WinLIRC is at http://winlirc.sourceforge.net
; 2) Edit the WinLIRC path, address, and port in
the CONFIG section below.
; 3) Launch this script. It will start the WinLIRC
server if needed.
; 4) Press some buttons on your remote control. A
small window will
;   appear showing the name of each button as you
press it.
; 5) Configure your buttons to send keystrokes and
mouse clicks to
;   windows such as Winamp, Media Player, etc.
See the examples below.

; This script requires AutoHotkey 1.0.38.04 or
later.
; HISTORY OF CHANGES
; March 2, 2007:
; - Improved reliability via "Critical" in
ReceiveData().
; October 5, 2005:
; - Eliminated Winsock warning dialog "10054" upon
system shutdown/logoff.
```

```
; - Added option "DelayBetweenButtonRepeats" to
throttle the repeat speed.

; -----
-
; CONFIGURATION SECTION: Set your preferences
here.
; -----
-
; Some remote controls repeat the signal rapidly
while you're holding down
; a button. This makes it difficult to get the
remote to send only a single
; signal. The following setting solves this by
ignoring repeated signals
; until the specified time has passed. 200 is
often a good setting. Set it
; to 0 to disable this feature.
DelayBetweenButtonRepeats = 200

; Specify the path to WinLIRC, such as
C:\WinLIRC\winlirc.exe
WinLIRC_Path =
%A_ProgramFiles%\WinLIRC\winlirc.exe

; Specify WinLIRC's address and port. The most
common are 127.0.0.1 (localhost) and 8765.
WinLIRC_Address = 127.0.0.1
WinLIRC_Port = 8765

; Do not change the following two lines. Skip them
and continue below.
Gosub WinLIRC_Init
return

; -----
; ASSIGN ACTIONS TO THE BUTTONS ON YOUR REMOTE
```

```
; -----  
; Configure your remote control's buttons below.  
Use WinLIRC's names  
; for the buttons, which can be seen in your  
WinLIRC config file  
; (.cf file) -- or you can press any button on  
your remote and the  
; script will briefly display the button's name in  
a small window.  
;  
; Below are some examples. Feel free to revise or  
delete them to suit  
; your preferences.
```

VolUp:

```
SoundSet +5 ; Increase master volume by 5%. On  
Vista, replace this line with: Send {Volume_Up}  
return
```

VolDown:

```
SoundSet -5 ; Reduce master volume by 5%. On  
Vista, replace this line with: Send {Volume_Down}  
return
```

ChUp:

```
WinGetClass, ActiveClass, A  
if ActiveClass in Winamp v1.x, Winamp PE ; Winamp  
is active.  
    Send {right} ; Send a right-arrow keystroke.  
else ; Some other type of window is active.  
    Send {WheelUp} ; Rotate the mouse wheel up by  
one notch.  
return
```

ChDown:

```
WinGetClass, ActiveClass, A  
if ActiveClass in Winamp v1.x, Winamp PE ; Winamp
```

```
is active.  
    Send {left} ; Send a left-arrow keystroke.  
else ; Some other type of window is active.  
    Send {WheelDown} ; Rotate the mouse wheel  
down by one notch.  
return
```

Menu:

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

```
Send Here are some keystrokes sent to Notepad.  
{Enter}  
return
```

```
; The examples above give a feel for how to  
accomplish common tasks.  
; To learn the basics of AutoHotkey, check out the  
Quick-start Tutorial  
; at http://www.autohotkey.com/docs/Tutorial.htm
```

```
; -----  
; END OF CONFIGURATION SECTION  
; -----  
; Do not make changes below this point unless you  
want to change the core  
; functionality of the script.
```

WinLIRC_Init:

```
OnExit, ExitSub ; For connection cleanup
```

purposes.

```
; Launch WinLIRC if it isn't already running:
Process, Exist, winlirc.exe
if not ErrorLevel ; No PID for WinLIRC was found.
{
    IfNotExist, %WinLIRC_Path%
    {
        MsgBox The file "%WinLIRC_Path%" does not
        exist. Please edit this script to specify its
        location.
        ExitApp
    }
    Run %WinLIRC_Path%
    Sleep 200 ; Give WinLIRC a little time to
    initialize (probably never needed, just for peace
    of mind).
}

; Connect to WinLIRC (or any type of server for
that matter):
socket := ConnectToAddress(WinLIRC_Address,
WinLIRC_Port)
if socket = -1 ; Connection failed (it already
displayed the reason).
    ExitApp

; Find this script's main window:
Process, Exist ; This sets ErrorLevel to this
script's PID (it's done this way to support
compiled scripts).
DetectHiddenWindows On
ScriptMainWindowId := WinExist("ahk_class
AutoHotkey ahk_pid " . ErrorLevel)
DetectHiddenWindows Off

; When the OS notifies the script that there is
```

```

incoming data waiting to be received,
; the following causes a function to be launched
to read the data:
NotificationMsg = 0x5555 ; An arbitrary message
number, but should be greater than 0x1000.
OnMessage(NotificationMsg, "ReceiveData")

; Set up the connection to notify this script via
message whenever new data has arrived.
; This avoids the need to poll the connection and
thus cuts down on resource usage.
FD_READ = 1 ; Received when data is available
to be read.
FD_CLOSE = 32 ; Received when connection has
been closed.
if DllCall("Ws2_32\WSAAsyncSelect", "UInt",
socket, "UInt", ScriptMainWindowId, "UInt",
NotificationMsg, "Int", FD_READ|FD_CLOSE)
{
    MsgBox % "WSAAsyncSelect() indicated Winsock
error " . DllCall("Ws2_32\WSAGetLastError")
    ExitApp
}
return

```

```

ConnectToAddress(IPAddress, Port)
; This can connect to most types of TCP servers,
not just WinLIRC.
; Returns -1 (INVALID_SOCKET) upon failure or the
socket ID upon success.
{
    VarSetCapacity(wsaData, 400)
    result := DllCall("Ws2_32\WSAStartup",
"UShort", 0x0002, "UInt", &wsaData) ; Request
Winsock 2.0 (0x0002)

```

```

    ; Since WSASStartup() will likely be the first
Winsock function called by this script,
    ; check ErrorLevel to see if the OS has
Winsock 2.0 available:
    if ErrorLevel
    {
        MsgBox WSASStartup() could not be called
due to error %ErrorLevel%. Winsock 2.0 or higher
is required.
        return -1
    }
    if result ; Non-zero, which means it failed
(most Winsock functions return 0 upon success).
    {
        MsgBox % "WSASStartup() indicated Winsock
error " . DllCall("Ws2_32\WSAGetLastError")
        return -1
    }

    AF_INET = 2
    SOCK_STREAM = 1
    IPPROTO_TCP = 6
    socket := DllCall("Ws2_32\socket", "Int",
AF_INET, "Int", SOCK_STREAM, "Int", IPPROTO_TCP)
    if socket = -1
    {
        MsgBox % "socket() indicated Winsock error
" . DllCall("Ws2_32\WSAGetLastError")
        return -1
    }

    ; Prepare for connection:
    SizeOfSocketAddress = 16
    VarSetCapacity(SocketAddress,
SizeOfSocketAddress)
    InsertInteger(2, SocketAddress, 0, AF_INET)
; sin_family

```

```

        InsertInteger(DllCall("Ws2_32\htons",
"UShort", Port), SocketAddress, 2, 2) ; sin_port
        InsertInteger(DllCall("Ws2_32\inet_addr",
"AStr", IPAddress), SocketAddress, 4, 4) ;
sin_addr.s_addr

        ; Attempt connection:
        if DllCall("Ws2_32\connect", "UInt", socket,
"UInt", &SocketAddress, "Int",
SizeOfSocketAddress)
        {
            MsgBox % "connect() indicated Winsock
error " . DllCall("Ws2_32\WSAGetLastError") . ".
Is WinLIRC running?"
            return -1
        }
        return socket ; Indicate success by returning
a valid socket ID rather than -1.
}

```

```

ReceiveData(wParam, lParam)
; By means of OnMessage(), this function has been
set up to be called automatically whenever new
data
; arrives on the connection. It reads the data
from WinLIRC and takes appropriate action
depending
; on the contents.
{
    Critical ; Prevents another of the same
message from being discarded due to thread-
already-running.
    socket := wParam
    ReceivedDataSize = 4096 ; Large in case a lot
of data gets buffered due to delay in processing

```

previous data.

```
    VarSetCapacity(ReceivedData, ReceivedDataSize,
0) ; 0 for last param terminates string for use
with recv().
```

```
    ReceivedDataLength := DllCall("Ws2_32\recv",
"UInt", socket, "Str", ReceivedData, "Int",
ReceivedDataSize, "Int", 0)
```

```
    if ReceivedDataLength = 0 ; The connection
was gracefully closed, probably due to exiting
WinLIRC.
```

```
        ExitApp ; The OnExit routine will call
WSACleanup() for us.
```

```
    if ReceivedDataLength = -1
    {
```

```
        WinsockError :=
DllCall("Ws2_32\WSAGetLastError")
        if WinsockError = 10035 ; WSAEWOULDBLOCK,
which means "no more data to be read".
```

```
            return 1
```

```
            if WinsockError <> 10054 ; WSAECONNRESET,
which happens when WinLIRC closes via system
shutdown/logoff.
```

```
                ; Since it's an unexpected error,
report it. Also exit to avoid infinite loop.
```

```
                MsgBox % "recv() indicated Winsock
error " . WinsockError
```

```
                ExitApp ; The OnExit routine will call
WSACleanup() for us.
```

```
    }
```

```
    ; Otherwise, process the data received.
```

```
Testing shows that it's possible to get more than
one line
```

```
    ; at a time (even for explicitly-sent IR
signals), which the following method handles
properly.
```

```
    ; Data received from WinLIRC looks like the
```

following example (see the WinLIRC docs for details):

```
    ; 0000000000eab154 00 NameOfButton
NameOfRemote
    Loop, parse, ReceivedData, `n, `r
    {
        if A_LoopField in ,BEGIN,SIGHUP,END ;
Ignore blank lines and WinLIRC's start-up
messages.
            continue
        ButtonName = ; Init to blank in case
there are less than 3 fields found below.
        Loop, parse, A_LoopField, %A_Space% ;
Extract the button name, which is the third field.
        if A_Index = 3
            ButtonName := A_LoopField
            global DelayBetweenButtonRepeats ;
Declare globals to make them available to this
function.
            static PrevButtonName, PrevButtonTime,
RepeatCount ; These variables remember their
values between calls.
            if (ButtonName != PrevButtonName ||
A_TickCount - PrevButtonTime >
DelayBetweenButtonRepeats)
            {
                if IsLabel(ButtonName) ; There is a
subroutine associated with this button.
                    Gosub %ButtonName% ; Launch the
subroutine.
                else ; Since there is no associated
subroutine, briefly display which button was
pressed.
                    {
                        if (ButtonName == PrevButtonName)
                            RepeatCount += 1
                        else
```

```

        RepeatCount = 1
        SplashTextOn, 150, 20, Button from
WinLIRC, %ButtonName% (%RepeatCount%)
        SetTimer, SplashOff, 3000 ; This
allows more signals to be processed while
displaying the window.
    }
    PrevButtonName := ButtonName
    PrevButtonTime := A_TickCount
}
}
return 1 ; Tell the program that no further
processing of this message is needed.
}

```

```

SplashOff:
SplashTextOff
SetTimer, SplashOff, Off
return

```

```

InsertInteger(pInteger, ByRef pDest, pOffset = 0,
pSize = 4)
; The caller must ensure that pDest has sufficient
capacity. To preserve any existing contents in
pDest,
; only pSize number of bytes starting at pOffset
are altered in it.
{
    Loop %pSize% ; Copy each byte in the integer
into the structure as raw binary data.
        DllCall("RtlFillMemory", "UInt", &pDest +
pOffset + A_Index-1, "UInt", 1, "UChar", pInteger
>> 8*(A_Index-1) & 0xFF)

```

```
}
```

```
ExitSub: ; This subroutine is called  
automatically when the script exits for any  
reason.  
; MSDN: "Any sockets open when WSACleanup is  
called are reset and automatically  
; deallocated as if closesocket was called."  
DllCall("ws2_32\WSACleanup")  
ExitApp
```

Script Performance

To maximize performance, avoid using `SetFormat` (except the [fast mode](#)) and include the following lines near the top of each script:

```
#NoEnv
SetBatchLines -1
ListLines Off
```

In addition, the following commands may also affect performance depending on the nature of the script: [SendMode](#), [SetKeyDelay](#), [SetMouseDelay](#), [SetWinDelay](#), [SetControlDelay](#), and [SetDefaultMouseSpeed](#).

Built-in Performance Features

Each script is semi-compiled while it is being loaded and syntax-checked. In addition to reducing the memory consumed by the script, this also greatly improves runtime performance.

Here are the technical details of the optimization process (semi-compiling):

- 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 is a variable.
- Each command name is replaced by an address in a jump table.
- Each line is pre-parsed into a list of parameters, and each parameter is pre-parsed into a list of [variables](#) (if any).
- Each [expression](#) is tokenized and converted from infix to postfix.
- Each reference to a [variable](#) or [function](#) is resolved to a memory address.
- Literal integers in expressions and math/comparison commands are replaced with binary integers.

In addition, during script execution, binary numbers are cached in variables to

avoid conversions to/from strings. See [SetFormat](#) for details.

Var := expression

Evaluates an expression and stores the result in a [variable](#).

```
Var := expression
```

Parameters

Var

The name of the [variable](#) in which to store the result of *expression*.

Expression

See [expressions](#) and the examples below for details.

Remarks

The := operator is optimized so that it performs just as quickly as the = operator for simple cases such as the following:

```
x := y ; Same performance as x = %y%  
x := 5 ; Same performance as x = 5.  
x := "literal string" ; Same performance as x =  
literal string.
```

The words `true` and `false` are built-in constants containing 1 and 0. They can be used to make a script more readable as in these examples:

```
CaseSensitive := false  
ContinueSearch := true
```

It is possible to create a [pseudo-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%i% := Var/100 + 5`. See [Arrays](#) for more

information.

Related

[Expressions](#), [IF \(expression\)](#), [Functions](#), [SetEnv](#), [EnvSet](#), [EnvAdd](#), [EnvSub](#), [EnvMult](#), [EnvDiv](#), [If](#), [Arrays](#)

Examples

```
Var := 3
Var := "literal string"
Var := Price * (1 - Discount/100)

Finished := not Done or A_Index > 100
if not Finished
{
    FileAppend, %NewText%\n, %TargetFile%
    return
}
else
    ExitApp
```

Arrays

In AutoHotkey, there are two different types of things that are related to arrays:

- [Object-based Arrays](#)
- [Pseudo-Arrays](#) (not recommended for use)

Note: The following code examples show different approaches which lead to the same end result.

Object-based Arrays [AHK_L 31+]

Such arrays can be [associative arrays](#) or [simple arrays](#). Associative arrays are created using the Object function or the brace syntax, while simple arrays are created using the array function or bracket syntax. For more information, see the [AutoHotkey Beginner Tutorial](#) or the [Objects](#) page.

The following example shows the usage of a simple array:

```
; Create the array, initially empty:
Array := [] ; or Array := Array()

; Write to the array:
Loop, Read, %A_WinDir%\system.ini ; This loop
retrieves each line from the file, one at a
time.
{
    Array.Push(A_LoopReadLine) ; Append this
line to the array.
}

; Read from the array:
; Loop % Array.MaxIndex() ; More traditional
approach.
for index, element in Array ; Enumeration is the
recommended approach in most cases.
{
    ; Using "Loop", indices must be consecutive
numbers from 1 to the number
    ; of elements in the array (or they must be
calculated within the loop).
    ; MsgBox % "Element number " . A_Index . "
is " . Array[A_Index]

    ; Using "for", both the index (or "key") and
its associated value
```

```

; are provided, and the index can be *any*
value of your choosing.
    MsgBox % "Element number " . index . " is "
    . element
}

```

This shows only a small subset of the [functionality](#) provided by [objects](#). Items can be set, retrieved, inserted, removed and enumerated. Strings and objects can be used as keys in addition to numbers. Objects can be stored as values in other objects and passed as function parameters or return values. Objects can also be [extended](#) with new functionality.

Though Push() and enumerators have their uses, some users might find it easier to use the more traditional approach (the commented out lines are the counterparts using the [pseudo-arrays](#) described below):

```

; Each array must be initialized before use:
Array := []

; Array%j% := A_LoopField
Array[j] := A_LoopField

; Array%j%_%k% := A_LoopReadLine
Array[j, k] := A_LoopReadLine

ArrayCount := 0
Loop, Read, %A_WinDir%\system.ini
{
    ArrayCount += 1
    ; Array%ArrayCount% := A_LoopReadLine
    Array[ArrayCount] := A_LoopReadLine
}

Loop % ArrayCount
{
    ; element := Array%A_Index%
    element := Array[A_Index]
}

```

```
    ; MsgBox % "Element number " . A_Index . "  
is " . Array%A_Index%  
    MsgBox % "Element number " . A_Index . "  
is " . Array[A_Index]  
}
```

ArrayCount is left as a variable for convenience, but can be stored in the array itself with `Array.Count := n` or it can be removed and

`Array.MaxIndex()` used in its place. If a starting index other than 1 is desired, `Array.MinIndex()` can also be used.

Pseudo-Arrays

Note: If possible, always use the object-based array mentioned above. It is superior to a pseudo-array in almost every aspect: it is space-saving, more flexible, clearer, and similar to many other programming languages.

Pseudo-arrays are mostly conceptual: Each array is really just a collection of sequentially numbered [variables](#) or [functions](#), each one being perceived as an *element* of the array. AutoHotkey does not link these elements together in any way.

In addition to array-creating commands like [StringSplit](#) and [WinGet List](#), any command that accepts an OutputVar or that assigns a value to a variable can be used to create an array. The simplest example is the [assignment operator \(:=\)](#), as shown below:

```
Array%j% := A_LoopField
```

Multidimensional arrays are possible by using a separator character of your choice between the indices. For example:

```
Array%j%_%k% := A_LoopReadLine
```

The following example demonstrates how to create and access an array, in this case a series of names retrieved from a text file:

```
; Write to the array:  
ArrayCount = 0  
Loop, Read, %A_WinDir%\system.ini ; This loop  
retrieves each line from the file, one at a  
time.  
{  
    ArrayCount += 1 ; Keep track of how many  
items are in the array.
```

```

    Array%ArrayCount% := A_LoopReadLine ; Store
this line in the next array element.
}

; Read from the array:
Loop %ArrayCount%
{
    ; The following line uses the := operator to
retrieve an array element:
    element := Array%A_Index% ; A_Index is a
built-in variable.
    ; Alternatively, you could use the "% "
prefix to make MsgBox or some other command
expression-capable:
    MsgBox % "Element number " . A_Index . " is
" . Array%A_Index%
}

```

A concept related to arrays is the use of [NumPut\(\)](#) and [NumGet\(\)](#) to store/retrieve a collection of numbers in binary format. This might be helpful in cases where performance and/or memory conservation are important.

Regular Expression Callouts [AHK_L 14+]

Callouts provide a means of temporarily passing control to the script in the middle of regular expression pattern matching. For detailed information about the PCRE-standard callout feature, see [pcre.txt](#).

Callouts are currently supported only by [RegExMatch](#) and [RegExReplace](#).

Syntax

The syntax for a callout in AutoHotkey is `(?CNumber:Function)`, where both *Number* and *Function* are optional. Colon ':' is allowed only if *Function* is specified, and is optional if *Number* is omitted. If *Function* is specified but is not the name of a user-defined function, a compile error occurs and pattern-matching does not begin.

If *Function* is omitted, the function name must be specified in a variable named **pcre_callout**. If both a global variable and local variable exist with this name, the local variable takes precedence. If *pcre_callout* does not contain the name of a user-defined function, callouts which omit *Function* are ignored.

Callout Functions

```
Function(Match, CalloutNumber, FoundPos, Haystack,
NeedleRegEx)
{
    ...
}
```

Callout functions may define up to 5 parameters:

- **Match:** Equivalent to the *OutputVar* of [RegExMatch](#), including the creation of array variables if appropriate.
- **CalloutNumber:** Receives the *Number* of the callout.
- **FoundPos:** Receives the position of the current potential match.

- **Haystack:** Receives the *Haystack* passed to `RegexMatch` or `RegexReplace`.
- **NeedleRegex:** Receives the *NeedleRegex* passed to `RegexMatch` or `RegexReplace`.

These names are suggestive only. Actual names may vary.

Pattern-matching may proceed or fail depending on the return value of the callout function:

- If the function returns **0** or does not return a numeric value, matching proceeds as normal.
- If the function returns **1** or greater, matching fails at the current point, but the testing of other matching possibilities goes ahead.
- If the function returns **-1**, matching is abandoned.
- If the function returns a value less than **-1**, it is treated as a PCRE error code and matching is abandoned. `RegexMatch` returns a blank string, while `RegexReplace` returns the original *Haystack*. In either case, `ErrorLevel` contains the error code.

For example:

```
Haystack = The quick brown fox jumps over the
lazy dog.
RegexMatch(Haystack, "i)(The) (\w+)\b(?
CCallout)")
Callout(m) {
    MsgBox m=%m%`nm1=%m1%`nm2=%m2%
    return 1
}
```

In the above example, *Func* is called once for each substring which matches the part of the pattern preceding the callout. `\b` is used to exclude incomplete words in matches such as *The quic*, *The qui*, *The qu*, etc.

EventInfo

Additional information is available by accessing the `pcre_callout_block` structure

via `A_EventInfo`.

```
version          := NumGet(A_EventInfo, 0,
"Int")
callout_number   := NumGet(A_EventInfo, 4,
"Int")
offset_vector    := NumGet(A_EventInfo, 8)
subject          := NumGet(A_EventInfo, 8 +
A_PtrSize)
subject_length   := NumGet(A_EventInfo, 8 +
A_PtrSize*2, "Int")
start_match      := NumGet(A_EventInfo, 12 +
A_PtrSize*2, "Int")
current_position := NumGet(A_EventInfo, 16 +
A_PtrSize*2, "Int")
capture_top      := NumGet(A_EventInfo, 20 +
A_PtrSize*2, "Int")
capture_last     := NumGet(A_EventInfo, 24 +
A_PtrSize*2, "Int")
pad := A_PtrSize=8 ? 4 : 0 ; Compensate for 64-
bit data alignment.
callout_data     := NumGet(A_EventInfo, 28 +
pad + A_PtrSize*2)
pattern_position := NumGet(A_EventInfo, 28 +
pad + A_PtrSize*3, "Int")
next_item_length := NumGet(A_EventInfo, 32 +
pad + A_PtrSize*3, "Int")
if version >= 2
    mark := StrGet(NumGet(A_EventInfo, 36 +
pad + A_PtrSize*3, "Int"), "UTF-8")
```

For more information, see [pcre.txt](#), `NumGet` and `A_PtrSize`.

Auto-Callout

Including `C` in the options of the pattern enables the auto-callout mode. In this

mode, callouts equivalent to `(?C255)` are inserted before each item in the pattern. For example, the following template may be used to debug regular expressions:

```
; Set the default callout function.
pcre_callout = DebugRegex

; Call RegExMatch with auto-callout option C.
RegExMatch("xxxabc123xyz", "C)abc.*xyz")

DebugRegex(Match, CalloutNumber, FoundPos,
Haystack, NeedleRegex)
{
    ; See pcre.txt for descriptions of these
fields.
    start_match      := NumGet(A_EventInfo, 12
+ A_PtrSize*2, "Int")
    current_position := NumGet(A_EventInfo, 16
+ A_PtrSize*2, "Int")
    pad := A_PtrSize=8 ? 4 : 0
    pattern_position := NumGet(A_EventInfo, 28
+ pad + A_PtrSize*3, "Int")
    next_item_length := NumGet(A_EventInfo, 32
+ pad + A_PtrSize*3, "Int")

    ; Point out >>current match<<.
    _HAYSTACK:=SubStr(Haystack, 1, start_match)
        . ">>" SubStr(Haystack, start_match + 1,
current_position - start_match)
        . "<<" SubStr(Haystack, current_position
+ 1)

    ; Point out >>next item to be evaluated<<.
    _NEEDLE:= SubStr(NeedleRegex, 1,
pattern_position)
        . ">>" SubStr(NeedleRegex,
pattern_position + 1, next_item_length)
```

```
        . "<<" SubStr(NeedleRegex,  
pattern_position + 1 + next_item_length)  
  
    ListVars  
    ; Press Pause to continue.  
    Pause  
}
```

Remarks

Callouts are executed on the current quasi-thread, but the previous value of `A_EventInfo` will be restored after the callout function returns. `ErrorLevel` is not set until immediately before `RegexMatch` or `RegexReplace` returns.

PCRE is optimized to abort early in some cases if it can determine that a match is not possible. For all callouts to be called in such cases, it may be necessary to disable these optimizations by specifying `(*NO_START_OPT)` at the start of the pattern. This requires [v1.1.05] or later.

Language Codes

The following list contains the language name that corresponds to each language code that can be contained in the `A_Language` variable. The language code itself is the last four digits on the left side of the equal sign below. For example, if `A_Language` contains 0436, the system's default language is Afrikaans.

Note: Codes that contain letters might use either uppercase or lowercase.

You can compare `A_Language` directly to one or more of the 4-digit codes below; for example: `if (A_Language = "0436")`. Alternatively, you can paste the entire list into a script and then access the name of the current language as demonstrated at the bottom of the list.

```
languageCode_0436 = Afrikaans
languageCode_041c = Albanian
languageCode_0401 = Arabic_Saudi_Arabia
languageCode_0801 = Arabic_Iraq
languageCode_0c01 = Arabic_Egypt
languageCode_1001 = Arabic_Libya
languageCode_1401 = Arabic_Algeria
languageCode_1801 = Arabic_Morocco
languageCode_1c01 = Arabic_Tunisia
languageCode_2001 = Arabic_Oman
languageCode_2401 = Arabic_Yemen
languageCode_2801 = Arabic_Syria
languageCode_2c01 = Arabic_Jordan
languageCode_3001 = Arabic_Lebanon
languageCode_3401 = Arabic_Kuwait
languageCode_3801 = Arabic_UAE
languageCode_3c01 = Arabic_Bahrain
languageCode_4001 = Arabic_Qatar
languageCode_042b = Armenian
languageCode_042c = Azeri_Latin
languageCode_082c = Azeri_Cyrillic
```

languageCode_042d = Basque
languageCode_0423 = Belarusian
languageCode_0402 = Bulgarian
languageCode_0403 = Catalan
languageCode_0404 = Chinese_Taiwan
languageCode_0804 = Chinese_PRC
languageCode_0c04 = Chinese_Hong_Kong
languageCode_1004 = Chinese_Singapore
languageCode_1404 = Chinese_Macau
languageCode_041a = Croatian
languageCode_0405 = Czech
languageCode_0406 = Danish
languageCode_0413 = Dutch_Standard
languageCode_0813 = Dutch_Belgian
languageCode_0409 = English_United_States
languageCode_0809 = English_United_Kingdom
languageCode_0c09 = English_Australian
languageCode_1009 = English_Canadian
languageCode_1409 = English_New_Zealand
languageCode_1809 = English_Irish
languageCode_1c09 = English_South_Africa
languageCode_2009 = English_Jamaica
languageCode_2409 = English_Caribbean
languageCode_2809 = English_Belize
languageCode_2c09 = English_Trinidad
languageCode_3009 = English_Zimbabwe
languageCode_3409 = English_Philippines
languageCode_0425 = Estonian
languageCode_0438 = Faeroese
languageCode_0429 = Farsi
languageCode_040b = Finnish
languageCode_040c = French_Standard
languageCode_080c = French_Belgian
languageCode_0c0c = French_Canadian
languageCode_100c = French_Swiss
languageCode_140c = French_Luxembourg
languageCode_180c = French_Monaco

languageCode_0437 = Georgian
languageCode_0407 = German_Standard
languageCode_0807 = German_Swiss
languageCode_0c07 = German_Austrian
languageCode_1007 = German_Luxembourg
languageCode_1407 = German_Liechtenstein
languageCode_0408 = Greek
languageCode_040d = Hebrew
languageCode_0439 = Hindi
languageCode_040e = Hungarian
languageCode_040f = Icelandic
languageCode_0421 = Indonesian
languageCode_0410 = Italian_Standard
languageCode_0810 = Italian_Swiss
languageCode_0411 = Japanese
languageCode_043f = Kazakh
languageCode_0457 = Konkani
languageCode_0412 = Korean
languageCode_0426 = Latvian
languageCode_0427 = Lithuanian
languageCode_042f = Macedonian
languageCode_043e = Malay_Malaysia
languageCode_083e = Malay_Brunei_Darussalam
languageCode_044e = Marathi
languageCode_0414 = Norwegian_Bokmal
languageCode_0814 = Norwegian_Nynorsk
languageCode_0415 = Polish
languageCode_0416 = Portuguese_Brazilian
languageCode_0816 = Portuguese_Standard
languageCode_0418 = Romanian
languageCode_0419 = Russian
languageCode_044f = Sanskrit
languageCode_081a = Serbian_Latin
languageCode_0c1a = Serbian_Cyrillic
languageCode_041b = Slovak
languageCode_0424 = Slovenian
languageCode_040a = Spanish_Traditional_Sort

```
languageCode_080a = Spanish_Mexican
languageCode_0c0a = Spanish_Modern_Sort
languageCode_100a = Spanish_Guatemala
languageCode_140a = Spanish_Costa_Rica
languageCode_180a = Spanish_Panama
languageCode_1c0a = Spanish_Dominican_Republic
languageCode_200a = Spanish_Venezuela
languageCode_240a = Spanish_Colombia
languageCode_280a = Spanish_Peru
languageCode_2c0a = Spanish_Argentina
languageCode_300a = Spanish_Ecuador
languageCode_340a = Spanish_Chile
languageCode_380a = Spanish_Uruguay
languageCode_3c0a = Spanish_Paraguay
languageCode_400a = Spanish_Bolivia
languageCode_440a = Spanish_El_Salvador
languageCode_480a = Spanish_Honduras
languageCode_4c0a = Spanish_Nicaragua
languageCode_500a = Spanish_Puerto_Rico
languageCode_0441 = Swahili
languageCode_041d = Swedish
languageCode_081d = Swedish_Finland
languageCode_0449 = Tamil
languageCode_0444 = Tatar
languageCode_041e = Thai
languageCode_041f = Turkish
languageCode_0422 = Ukrainian
languageCode_0420 = Urdu
languageCode_0443 = Uzbek_Latin
languageCode_0843 = Uzbek_Cyrillic
languageCode_042a = Vietnamese
```

```
the_language := languageCode_%A_Language% ; Get
the name of the system's default language.
MsgBox %the_language% ; Display the language
name.
```

Context Sensitive Help in Any Editor -- by Rajat

This script makes `Ctrl` `2` (another hotkey of your choice) show the help file page for the selected AutoHotkey command or keyword. If nothing is selected, the command name will be extracted from the beginning of the current line.

[Download This Script](#) | [Other Sample Scripts](#) | [Home](#)

```
; The hotkey below uses the clipboard to provide  
compatibility with the maximum  
; number of editors (since ControlGet doesn't work  
with most advanced editors).  
; It restores the original clipboard contents  
afterward, but as plain text,  
; which seems better than nothing.
```

```
$^2::
```

```
; The following values are in effect only for the  
duration of this hotkey thread.  
; Therefore, there is no need to change them back  
to their original values  
; because that is done automatically when the  
thread ends:
```

```
SetWinDelay 10
```

```
SetKeyDelay 0
```

```
AutoTrim, On
```

```
if A_OSType = WIN32_WINDOWS ; Windows 9x  
    Sleep, 500 ; Give time for the user to  
release the key.
```

```
C_ClipboardPrev = %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, 0.1
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 = %C_ClipboardPrev%
        return
    }
}
C_Cmd = %clipboard% ; This will trim leading and
trailing tabs & spaces.
clipboard = %C_ClipboardPrev% ; Restore the
original clipboard for the user.
Loop, parse, C_Cmd, %A_Space`, ; The first space
or comma is the end of the command.
{
    C_Cmd = %A_LoopField%
    break ; i.e. we only need one iteration.
}
IfWinNotExist, AutoHotkey Help
{
    ; Determine AutoHotkey's location:
    RegRead, ahk_dir, HKEY_LOCAL_MACHINE,
SOFTWARE\AutoHotkey, InstallDir
    if ErrorLevel ; Not found, so look for it in
some other common locations.
    {

```

```
if A_AhkPath
    SplitPath, A_AhkPath,, ahk_dir
else IfExist ..\..\AutoHotkey.chm
    ahk_dir = ..\..
else IfExist
%A_ProgramFiles%\AutoHotkey\AutoHotkey.chm
    ahk_dir = %A_ProgramFiles%\AutoHotkey
else
{
    MsgBox Could not find the AutoHotkey
folder.
    return
}
}
Run %ahk_dir%\AutoHotkey.chm
WinWait AutoHotkey Help
}
```

; The above has set the "last found" window which we use below:

```
WinActivate
WinWaitActive
StringReplace, C_Cmd, C_Cmd, #, {#}
Send, !n{home}+{end}%C_Cmd%{enter}
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 but should work for other major releases as well. Please post changes and improvements in the forum or contact the author.

This example makes the  key equivalent to pressing Winamp's pause/unpause button:

```
^!p::  
IfWinNotExist ahk_class Winamp v1.x  
    return  
; Otherwise, the above has set the "last found"  
window for use below.  
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
c	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:
SendMessage, 1024, 0, 120,, ahk_class Winamp
v1.x
if ErrorLevel <> FAIL
{
    ErrorLevel += 1 ; Winamp's count starts at
0, so adjust by 1.
    MsgBox, Track #%ErrorLevel% is active or
playing.
}
```

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.

[Download This Script](#) | [Other Sample Scripts](#) | [Home](#)

```
#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, Button1, &Add
ControlSetText, Button2, &Delete
return
```

Numpad 000 Key

This example script makes the special  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.

[Download This Script](#) | [Other Sample Scripts](#) | [Home](#)

```
#MaxThreadsPerHotkey 5 ; Allow multiple threads
for this hotkey.
$Numpad0::
#MaxThreadsPerHotkey 1
; 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
```

Archived Changes

This document contains a history of changes made within the original branch of AutoHotkey development, by Chris Mallett.

For newer changes, see [Changes & New Features](#).

1.0.48.05 - September 25, 2009

Fixed crash of [SendMessage](#) and [PostMessage](#) when wParam or lParam is omitted (broken by 1.0.48.04). [thanks Lexikos]

1.0.48.04 - September 25, 2009

Fixed `StringSplit` to work properly inside `functions` of `compiled scripts` (broken by 1.0.35.01). [thanks engunneer & Lexikos]

Fixed `SendPlay` not to wait for the release of the Windows key prior to sending an "L" keystroke (broken by 1.0.48.01). [thanks Lexikos]

Fixed `A_EndChar` to be valid when the `B0 option` is present, and to be empty when there is no `ending character` (broken by 1.0.44.09). [thanks AI2000]

Fixed `FormatTime` to yield a valid time of day even when the specified month is out-of-range (broken by 1.0.48.00). [thanks silveredge78]

Fixed `FileCreateDir` to support a leading backslash even when it is the only backslash; e.g. `\dir`. [thanks jaco0646]

Fixed `GuiControl/GuiControlGet/Gui/SendMessage` to work reliably even when they trigger a `callback` or `OnMessage` function. [thanks Lexikos]

Fixed `RegExMatch()` not to produce too few replacements when an empty-string match is followed by a non-empty-string match.

Changed `while()` to be recognized as a `loop` rather than a `function`. [thanks Crash&Burn]

Improved `UrlDownloadToFile` to support FTP and Gopher. [thanks Lexikos]

Improved the `stdout/asterisk mode` of `FileAppend` to write immediately rather than lazily to standard output. [thanks Lexikos]

Added full support for `if % expression`. [thanks kenomby]

1.0.48.03 - May 3, 2009

Fixed `ListLines On` not to erase the most recent log entry in the line history. [thanks Lexikos]

Fixed `ListView` to respond properly to mouse dragging when `timers` are running. [thanks Solar]

Fixed `key-up hotkeys` so that if one is created while its key is being held down, the release of the key doesn't trigger the wrong hotkey. [thanks Peter & engunneer]

1.0.48.02 - April 19, 2009

Changed and fixed [Gosub](#) and [GroupActivate](#) so that when a function calls an [external/public subroutine](#), that subroutine will treat all dynamic variables as globals, and will have outside-of-function GUI behavior. [thanks kenomby & Lexikos]

Improved performance of [True/False/A_EventInfo](#) in [expressions](#) by treating them as integers vs. strings.

1.0.48.01 - April 15, 2009

Changed: For Windows Vista and later, hotkeys that include the `Win`  (e.g. #a) will wait for LWin and RWin to be released before sending any text containing an "L" keystroke. This prevents such a hotkey from locking the PC. This behavior applies to all sending modes except `SendPlay` (which doesn't need it) and `blind mode`.

Fixed `A_LoopFileExt` to be blank for any filename having no extension but a period in its pathname. [thanks Yek-Toho-Tua].

Fixed the `assignment` of integers that are 19 or 20 characters long to work as they did prior to v1.0.48. [thanks Laszlo & Lexikos]

Fixed `function definitions` to work properly inside a `block`. [thanks rmarko]

Improved performance of `A_Index` in `expressions` by treating it as an integer rather than a string.

Improved performance of `if var [not] in/contains`. [developed by Lexikos]

Improved `ListLines` with an option to turn line-logging Off or On. [thanks kenomby & ruespe]

1.0.48 - February 25, 2009

Compatibility: The change most likely to affect backward compatibility is that floating point numbers stored in variables now have higher precision. Scripts that rely on tiny differences in precision would either need to be reviewed and updated, or have their compatibility improved by using `SetFormat Float` (e.g. `SetFormat, Float, 0.6`) *anywhere* in the script. `SetFormat Float` disables the higher precision, but gives up some of the new, faster floating point performance.

Performance: The main theme of this release is faster performance. Almost all scripts should run faster -- especially those that make heavy use of [expressions](#) and integer math/comparisons (which may run up to three times as fast). To achieve the full benefit, a script either should avoid using `SetFormat` or should use [SetFormat's fast mode](#).

Performance improvements

[Expressions](#) and [function calls](#) are compiled more heavily, making them much faster (especially complex integer expressions, including those with [commas](#)).

Binary numbers are cached for variables to avoid conversions to/from strings. This makes numerical operations involving variables much faster.

Literal integers in expressions and math/comparison commands are replaced with binary integers, which makes them faster; e.g. `X+5` and `if x > 5`.

[LOOPS](#), [IFs](#), and [ELSEs](#) that have blocks (braces) are faster due to skipping the opening '{'. A side-effect is that the '{' is omitted from [ListLines](#).

[Thread-creation](#) performance is improved, which should help rapid-fire threads in [OnMessage\(\)](#), [RegisterCallback\(\)](#), and [GUI events](#).

Changes that might affect existing scripts (other than higher-precision floating point described at the top)

When `SetFormat, Integer, Hex` is in effect, assigning a literal decimal

integer to a variable also converts it to hex. Usually this is only a display issue.

For `OnMessage()` performance, the message number and HWND arrive as standard numbers rather than appearing unconditionally as hex. Usually this is only a display issue.

To achieve various improvements in performance, scripts now use slightly more memory (proportionate to the number of variables and expressions).

Changed and fixed `if var is time` and other uses of YYYYMMDDHHMISS date-time stamps to recognize that months outside the range 1-12 are invalid. [thanks Nick]

Changed and improved `dynamic function calling` to allow passing more parameters than defined by a function, in which case the parameters are evaluated but discarded. [developed by Lexikos]

Other improvements

Added function `IsFunc()`, which indicates whether a function may be called dynamically. [developed by Lexikos]

Added the `while-loop`, which repeats its commands until its `expression` evaluates to false. [developed by Lexikos]

Added an `assume-static mode` for functions. [developed by Lexikos]

Added built-in variables `A_IsPaused` and `A_IsCritical`. [developed by Lexikos]

Improved `NumPut()` to support UInt64 like `DllCall()`. [thanks Sean]

Improved mouse wheel support by adding `WheelLeft` and `WheelRight` as hotkeys and supporting them in `Send`, `Click`, and related commands. However, `WheelLeft/Right` has no effect on operating systems older than Windows Vista. [developed by Lexikos]

Upgraded `compiled script` compressor from UPX 3.00 to 3.03.

Fixes

Fixed inability to use [MsgBox](#)'s timeout parameter when the "Text" parameter had an expression containing commas.

Fixed [Menu, Delete, Item-that's-a-submenu](#) not to disrupt the associated submenu. [thanks animeaime & Lexikos]

Fixed the [GUI Hotkey control](#) to return usable hotkey names even for dead keys (e.g. "^" instead of Zircumflex). [thanks DerRaphael]

Fixed [RegDelete](#) so that it won't delete an entire root key when SubKey is blank. [thanks Icarus]

Fixed [registry loops](#) to support subkey names longer than 259 (rare). In prior versions, such subkeys would either be skipped or cause a crash. [thanks Krzysztof Sliwinski & Eggi]

Fixed FileSelectFolder by providing [an option](#) to make it compatible with BartPE/WinPE. [thanks markreflex]

Fixed [window/control IDs \(HWNDs\)](#), which in rare cases wrongly started with 0xFFFFFFFF instead of just 0x. [thanks Micahs]

Fixed inability of [Send commands](#) to use the Down/Up modifiers with the "}" character. [thanks neovars]

1.0.47.06 - March 9, 2008

Fixed crash when a [function](#) was called concurrently with an optional [ByRef parameter](#) omitted by one [thread](#) but not omitted by the other. [thanks DeathByNukes]

Fixed [Menu, Tray, Mainwindow](#) to enable the menu items in the main window's View menu. [thanks Lexikos]

Added [dynamic function calling](#). [developed by Lexikos]

1.0.47.05 - November 21, 2007

Fixed the [Sort command](#): 1) fixed the "function" option not to misbehave when it's the last option in the list; 2) fixed the "unique" option so that when the delimiter is CRLF, the last item can be detected as a duplicate even when it doesn't end in CRLF; 3) fixed the "unique" option not to append a trailing delimiter when the last item is a duplicate. [thanks Roland]

Fixed [RegExMatch\(\)](#) and [RegExReplace\(\)](#) to yield correct results even when Haystack and OutputVar are both the same variable. [thanks Superfraggle]

Fixed inability to pass a parameter that is "a variable to which [ClipboardAll](#) has been assigned". [thanks Joy2DWorld & Lexikos]

Updated RegEx/PCRE from 7.0 to 7.4. For a summary of the major changes, see www.pcre.org/news.txt. For full details of every change and fix, see www.pcre.org/changelog.txt.

Added GUI control "[Tab2](#)" that fixes rare redrawing problems in the original "Tab" control (e.g. activating a GUI window by clicking on a control's scrollbar). The original Tab control is retained for backward compatibility because "Tab2" puts its tab control after its contained controls in the tab-key navigation order. [thanks Xander]

1.0.47.04 - August 28, 2007

Fixed **key-up hotkeys** like `a up::` not to block the pressing of the "a" key unless the hotkey's **#IfWin criteria** are met. [thanks Roland]

Fixed `Round(Var, NegativeNumber)`, which in rare cases was off by 1. [thanks Icarus]

Fixed crash of scripts that end in a syntax error consisting of an orphaned IF-statement (broken by 1.0.47.00). [thanks msgbox of the German forum]

Eliminated the "GetClipboardData" error dialog. Instead, an empty string is retrieved when the data cannot be accessed within the **#ClipboardTimeout** period. [thanks ManaUser & Sean]

Changed GUI **checkboxes** and **radio buttons** to default to "no word-wrap" when no width, height, or CR/LF characters are specified. This solves display issues under certain unusual DPI settings. [thanks Boskoop]

1.0.47.03 - August 1, 2007

Fixed [expressions](#) to allow literal negative hexadecimal numbers that end in "E";
e.g. `fn(-0xe)`. [thanks Laszlo]

Fixed [block syntax](#) to allow a [function-call](#) immediately to the right of a `'}`'.
[thanks Roland]

1.0.47.02 - July 19, 2007

Fixed the [Number option](#) of Edit controls to properly display a balloon tip when the user types something other than a digit. [thanks tfcahm]

Fixed WM_TIMER not to be blocked unless it's posted to the script's main window. [thanks tfcahm]

Fixed [wildcard hotkeys](#) not to acquire [tilde behavior](#) when the same hotkey exists in the script with a tilde. [thanks Lexikos]

Fixed [declaration initializers](#) not to retain whitespace at the end of literal numbers. Also, they now allow spaces between a closing quote and the next comma. [thanks Hardeep]

1.0.47.01 - July 8, 2007

Fixed [RunAs](#) not to crash or misbehave when a domain is specified. [thanks Markus Frohnmaier]

Changed [relational operators](#) to yield integers even when the inputs are floating point; e.g. `1.0 < 2.0` yields 1 vs. 1.0. [thanks Lexikos]

1.0.47 - June 19, 2007

Added support for [function libraries](#), which allow a script to call a function in an external file without having to use [#Include](#).

Added [RegisterCallback\(\)](#), which creates a machine-code address that when called, redirects the call to a function in the script. [developed by Jonathan Rennison (JGR)]

Added [NumGet\(\)](#) and [NumPut\(\)](#), which retrieve/store binary numbers with much greater speed than [Extract/InsertInteger](#).

Improved [Sort](#) with an option to do custom sorting according to the criteria in a callback function. [thanks Laszlo]

Improved [OnMessage\(\)](#) with an option to allow more than one simultaneous [thread](#). [thanks JGR]

Improved [Critical](#) with an option to change the [message-check interval](#), which may improve reliability for some usages. [thanks Majkinetor and JGR]

Changed [Critical](#) to put [SetBatchLines -1](#) into effect.

Changed the error messages produced by [#ErrorStdOut](#) to contain a space before the colon. [thanks Toralf]

Fixed [OnMessage\(\)](#) [functions](#) that return one of their own local variables to return the number in that variable, not 0.

Fixed potential crashing of built-in variables that access the registry (e.g. [A_AppData](#), [A_Desktop](#), [A_MyDocuments](#), [A_ProgramFiles](#)). [thanks Tekl]

1.0.46.17 - May 31, 2007

Fixed [A_UserName](#) (broken by 1.0.46.16).

1.0.46.16 - May 30, 2007

Fixed `Gui, Tab, TabName` when used after a previous `Gui Tab`.
[thanks Toralf]

Improved `SetTimer` to treat `negative periods` as "run only once". [thanks Majkinetor]

Added `GuiControlGet Hwnd`, which is a more modular/dynamic way to retrieve a control's HWND. [thanks Majkinetor]

Added built-in variables `A_ThisLabel` and `A_ThisFunc`, which contain the names of the currently-executing label/function. [thanks Titan & Majkinetor]

1.0.46.15 - May 9, 2007

Fixed `GuiControl`, `GuiControlGet`, and `Gui ListView/TreeView` to support `static variables` and `ByRefs` that point to globals/statics. [thanks Peter]

Fixed `FileInstall` causing the `Random` command to become non-random in `compiled scripts`. [thanks Velocity]

Reduced the size of `compiled scripts` by about 16 KB due to UPX 3.0. [thanks to atnbueno for discovering the optimal command-line switches]

1.0.46.14 - May 2, 2007

Added the "require administrator" flag to the installer to avoid a warning dialog on Windows Vista. [thanks Roussi Nikolov]

1.0.46.13 - May 1, 2007

Fixed [hotkeys](#) like `*x` to fire even when `x` is also a hotkey that is prevented from firing due to `#IfWin`. [thanks Joy2DWorld & Engunneer]

Improved [optional parameters](#) to accept quoted/literal strings as default values.

Improved [ByRef parameters](#) with the ability to be [optional](#) (i.e. they may accept default values). [thanks Corrupt]

1.0.46.12 - April 24, 2007

Fixed inability to recognize a literal scientific notation number that begins with 0, such as 0.15e+1. [thanks Laszlo]

1.0.46.11 - April 23, 2007

Fixed inability to have a [function-call](#) as the first item in certain [comma-separated expressions](#). [thanks Majkinetor]

Fixed WinTitles like `ahk_id %ControlHwnd%` in [ControlGet's](#) [FindString/Choice/List](#), and [Control's](#) [Add/Delete/Choose](#). [thanks Freighter & PhiLho]

Improved floating point support to recognize scientific notation; e.g. 1.2e-5 (the decimal point is mandatory). Also improved "[SetFormat Float](#)" with an option to output in scientific notation. [thanks Laszlo]

1.0.46.10 - March 22, 2007

Fixed [StringSplit](#) inside [assume-local functions](#) so that it creates a local [array](#) even when `OutputArray0` exists as a global but not a local. [thanks KZ]

Improved [ListView's item-changed notification \("I"\)](#) to indicate via `ErrorLevel` whether the item has been selected/deselected, focused/unfocused, and/or checked/unchecked. [thanks foom]

Added an additional layer of protection to [compiled scripts](#). It is recommended that scripts containing sensitive data or source code be recompiled with the [/NoDecompile switch](#).

1.0.46.09 - March 4, 2007

Fixed `:=` deep inside expressions when used to assign the result of a recursive [function](#) to a [local variable](#) (broken by 1.0.46.06). [thanks Laszlo]

Fixed inability to pass certain [ternary expressions](#) to [ByRef parameters](#). [thanks Titan]

Fixed `GuiControlGet`, `OutputVar`, `Pos` so that it doesn't make the `OutputVar` blank. [thanks PhiLho]

Changed and fixed continuation sections so that the ["Comment" option](#) doesn't force the [LTrim option](#) into effect. [thanks Titan]

Changed the [Terminal Server Awareness flag](#) back to "disabled" on `AutoHotkey.exe` and compiled scripts. This improves flexibility and backward compatibility (see [discussion](#) at forum).

1.0.46.08 - February 7, 2007

Fixed unreliability of [ComSpec](#) and [environment variables](#) on Windows 9x (broken by v1.0.46.07). [thanks Loriss]

Changed: When AutoHotkey.exe is launched without a script specified, it will now run (or prompt you to create) the file AutoHotkey.ahk in the [My Documents](#) folder. The only exception is when AutoHotkey.ini exists in the working directory, in which case it uses the old behavior of executing that file.

Improved [DllCall](#) to support an integer in place of the function name, which is interpreted as the address of the function to call. [thanks Sean]

1.0.46.07 - January 23, 2007

Fixed crash of illegally-named [dynamic variables](#) on the left of an equal-sign assignment (broken by v1.0.45). [thanks PhiLho]

Fixed [FileMoveDir](#)'s "Option 2" to work properly even when the directory is being both renamed and moved. [thanks bugmenot]

Fixed inability to pass a variable [ByRef](#) if that same expression changed it from empty to non-empty (when [#NoEnv](#) is absent). [thanks Joy2DWorld]

Changed DllCall's [A_LastError](#) to reflect only changes made by the script, not by AutoHotkey itself. [thanks Azerty]

1.0.46.06 - January 16, 2007

Applied minor fixes and improvements to [regular expressions](#) by upgrading from PCRE 6.7 to 7.0. One of the most notable improvements is the ``a` option, which recognizes any type of newline (namely ``r`, ``n`, or ``r`n`). Similarly, the `\R` escape sequence means "any single newline of any type". See also: [Full PCRE changelog](#)

Changed and fixed all [Control commands](#) and [StatusBarWait](#) to obey [SetTitleMatchMode RegEx](#) as documented.

Changed [RegExReplace\(\)](#) to return the original/unaltered string rather than "" when an error occurs.

Changed: Enabled the [Terminal Server Awareness flag](#) on AutoHotkey.exe and [compiled scripts](#).

Improved performance when [assigning](#) large strings returned from [user-defined functions](#). [thanks Laszlo]

1.0.46.05 - January 4, 2007

Fixed the [Input command](#) to allow named end keys like {F9} to work even when the shift key is being held down (broken by v1.0.45). [thanks Halweg]

Fixed inability of "[Gui Show](#)" to focus the GUI window when the tray menu is used both to reload the script and to show the GUI window. [thanks Rnon]

Fixed inability to pass some types of [assignments \(:=\)](#) to a [ByRef](#) parameter. [thanks Laszlo]

1.0.46.04 - January 2, 2007

Fixed inability to pass the result of an `assignment (:=)` to a `ByRef` parameter.
[thanks Titan]

1.0.46.03 - December 18, 2006

Fixed [ListView](#)'s floating point sorting to produce the correct ordering. [thanks oldbrother/Goyyah/Laszlo]

1.0.46.02 - December 17, 2006

Fixed [environment variables](#) to work properly as input variables in various commands such as [StringLen](#) and [StringReplace](#) (broken by 1.0.44.14). [thanks Camarade_Tux]

1.0.46.01 - December 15, 2006

NOTE: Although this release has been extensively tested, several low-level enhancements were made. If you have any mission-critical scripts, it is recommended that you retest them and/or wait a few weeks for any bugs to get fixed.

Fixed comma-separated **declaration initializers** such as `local x = 1, y = 2` to work even when immediately below an if/else/loop statement.

Fixed **comma-separated expressions** so when the leftmost item is an assignment, it will occur before the others rather than after. [thanks Laszlo]

Changed and fixed **function-calls** so that any changes they make to **dynamic variable names**, **environment variables**, and **built-in variables** (such as **Clipboard**) are always visible to subsequent parts of the expression that called them.

Changed: When a **multi-statement comma** is followed immediately by a variable and an equal sign, that equal sign is automatically treated as a **:= assignment**. For example, all of the following are assignments: `x:=1, y=2, a=b=c`.

Changed **comma-separated expressions** to produce the following effects: 1) the leftmost `/=` operator becomes **true divide** rather than **EnvDiv**; 2) blank values are not treated as zero in math expressions (thus they yield blank results).

Improved the performance of **expressions** by 5 to 20% (depending on type).

Improved the **new assignment operators** such as `.=` to support the **Clipboard variable** (even in **comma-separated expressions**).

Improved the `.=` **operator** so that it doesn't require a space to its left.

Improved GUI controls to accept **static variables** as their **associated variables** (formerly only globals were allowed).

Added option **HwndOutputVar** to `Gui Add`, which stores a control's HWND

in OutputVar. [thanks Corrupt & Toralf]

1.0.46 - November 29, 2006

NOTE: Although this release has been extensively tested and is not expected to break any existing scripts, several low-level enhancements were made. If you have any mission-critical scripts, it is recommended that you retest them and/or wait a few weeks for any bugs to get fixed.

Added function `SubStr()`, which retrieves the specified number of characters at the specified position in a string.

Added assignment operators `//=`, `.=`, `|=`, `&=`, `^=`, `>>=`, and `<<=`, which can be used anywhere in expressions. For example, `Var .= "abc"` appends the string "abc" to `Var`'s current contents.

Added full support in expressions for the operators `:=`, `++`, `--`, `+=`, `-=`, `*=`, and `/=` (formerly, they could be used only as the leftmost operator on a line). All **assignment operators** (especially `++` and `--`) behave in a C-like way when their result is used by some other operator.

Added the **ternary operator** (`?:`), which is a shorthand replacement for the **if-else statement**. For example, `var := x>y ? 2 : 3` assigns the value 2 if `x` is greater than `y`; otherwise it assigns 3.

Added support for **comma-separated expressions**, which allow a single line to contain multiple assignments, function calls, and other expressions. [thanks PhiLho & Titan]

Improved **variable declarations** to support initialization on the same line. Note: A **static variable**'s initialization occurs only once, before the script begins executing.

Improved **line continuation** to support all expression operators. For example, a line that starts with `"?"` or `"+"` is automatically appended to the line above it.

Improved performance of operators `."` and `.="` to be as fast as the percent-sign method of appending a string.

Improved expressions to allow more types of [consecutive unary operators](#) such as `!!Var`. [thanks Laszlo]

Changed [Critical](#) to check messages less often (20 vs. 10ms), which improves the reliability of frequently-called [OnMessage functions](#). [thanks Majkinetor]

Changed: A variable named simply "?" is no longer valid in expressions due to the new [ternary operator](#).

Fixed [hotkeys](#) to support `:::` (colon as a hotkey) and `: & x` (colon as a [hotkey prefix](#)).

Fixed the installer to remove psapi.dll from the AutoHotkey folder (except on Windows NT4). This avoids a conflict with Internet Explorer 7. [thanks to all who reported it]

1.0.45.04 - November 15, 2006

Fixed crash on Windows 9x when a script doesn't actually run (e.g. due to syntax error) (broken by v1.0.45). [thanks rogerg]

Changed `Control Style|ExStyle` to report ErrorLevel 0 vs. 1 when the requested style change wasn't necessary because it was already in effect.

Improved `#Include` to support `%A_AppData%` and `%A_AppDataCommon%`.
[thanks Tekl]

1.0.45.03 - November 12, 2006

Fixed [file-pattern loops](#) not to crash when recursing into paths longer than 259 characters. In addition, such paths and files are now ignored (skipped over) by file-pattern loops, [FileSetAttrib](#), and [FileSetTime](#). [thanks PhilR]

Fixed [functions](#) that call themselves and assign the result to one of their own [locals](#) (broken by v1.0.45). [thanks bjennings]

Fixed crash of scripts containing [regular expressions](#) that have compile errors. [thanks PhiLho]

Fixed [GuiControl](#) not to convert [checkboxes](#) into 3-state unless requested. [thanks JBensimon]

Changed [UrlDownloadToFile](#) to announce a user-agent of "AutoHotkey" to the server rather than a blank string. [thanks jaco0646]

Improved [continuation sections](#) to support semicolon comments inside the section via the option-word *Comments*.

1.0.45.02 - November 8, 2006

Fixed [StringUpper](#) and [StringLower](#) to work when OutputVar is the clipboard (broken by v1.0.45). [thanks songsoverruins]

Fixed the hotkeys ~Alt, ~Control, and ~Shift to fire upon press-down rather than release (broken by v1.0.44).

Background: Without the tilde, Alt/Control/Shift fire upon release to avoid taking over both the left and right key. But a specific left/right hotkey like LAlt or RShift fires upon press-down.

1.0.45.01 - November 7, 2006

Fixed [FileReadLine](#) and [FileSelectFile](#) not to crash or misbehave when other [threads](#) interrupt them (broken by v1.0.45). [thanks toralf]

Fixed [RegExMatch\(\)](#) so that when there's no match, named subpatterns are properly set to "" in the output array. [thanks PhiLho]

Fixed [RegExMatch\(\)](#)'s "J" option to properly write duplicate named subpatterns to the output array. [thanks PhiLho]

Changed [SetWorkingDir](#) and [#Include DirName](#) to succeed even for a root directory such as C: that lacks a backslash.

Improved [DllCall\(\)](#) to display a warning dialog if the called function writes to a variable of zero capacity.

1.0.45 - November 4, 2006

NOTE: Although this release has been extensively tested and is not expected to break any existing scripts, several low-level performance enhancements were made. If you have any mission-critical scripts, it is recommended that you retest them and/or wait a few weeks for any bugs to get fixed.

Added support for [regular expressions](#) via [RegExMatch\(\)](#), [RegExReplace\(\)](#), and [SetTitleMatchMode RegEx](#). [thanks Philip Hazel & PhiLho]

Improved performance and memory utilization of [StringReplace](#).

Improved performance of the [:= operator](#) for expressions and functions involving long strings.

Improved [ControlClick](#) with a new option "NA" that avoids activating the target window (this mode also improves reliability in some cases). In addition, it's been documented that [SetControlDelay -1](#) can improve the reliability of [ControlClick](#) in some cases. [thanks nnesori]

Changed [GUI buttons](#) to default to "no word-wrap" when no width, height, or CR/LF characters were specified. This may solve button display issues under some desktop themes.

Fixed "[Transform HTML](#)" for the following characters: &`n><

Fixed misinterpretation of lines starting with "if not is" such as "if not [IsDone](#)".

Fixed inability of "[Gui Show](#)" to move a window vertically downward to where its bottommost row of pixels is now.

Fixed inability to use [GroupActivate](#) as the only line beneath an IF or ELSE.

Fixed inability of the [Input command](#) to differentiate between end-keys enclosed in braces and their (un)shifted counterparts; e.g. '{' vs. '['. [thanks Laszlo]

Older Changes

Visit www.autohotkey.com/changelog/ for even older changes.

ObjBindMethod() [v1.1.20+]

Creates a [BoundFunc object](#) which calls a method of a given object.

```
BoundFunc := ObjBindMethod(Obj, Method, Params)
```

Parameters

Obj

Any object.

Method

A method name.

Params

Any number of parameters.

Remarks

For details and examples, see [BoundFunc object](#).



AutoHotkey

Automation. Hotkeys. Scripting.

Welcome...

Getting Started

Before AutoHotkey can do anything, you need to write a script telling it what to do.

To learn how, read [the tutorial](#) or [dive straight in](#).

Basically: [create a script file](#) → [add some hotkeys](#) → run the script file.

You can also get help or find many useful scripts on [our friendly discussion forum](#).

Why am I seeing this?

You ran AutoHotkey without [telling it](#) which script file to run, and it couldn't find a script in either of [the usual places](#).

AutoIt v2 Compatibility

[AutoHotkey v1.1.09+] **do not support** AutoIt v2 (.aut) scripts. Older versions of AutoHotkey supported AutoIt v2 scripts by changing the default settings and behaviour of some commands. This page contains information which has been removed from other pages in the documentation.

The following only applies to .aut files, and only on [v1.1.08.01] or earlier.

<code>#AllowSameLineComments</code>	By default, comments are not allowed on the same line as a command.
<code>A_ScriptDir</code>	A final backslash is included.
<code>DetectHiddenText</code>	Defaults to Off.
<code>#EscapeChar</code>	Defaults to backslash (\).
<code>FileCopy</code>	<code>ErrorLevel</code> is set to 1 if any of the files could not be copied.
<code>IniDelete</code>	<code>ErrorLevel</code> is not changed.
<code>IniRead</code>	The <i>Default</i> parameter is not supported; the string ERROR will always be stored in <i>OutputVar</i> if there was a problem reading the value.
<code>IniWrite</code>	<code>ErrorLevel</code> is not changed.
<code>InputBox</code>	If the user presses the CANCEL button, <i>OutputVar</i> is set to be blank. <code>ErrorLevel</code> is not changed unless the dialog times out.
<code>SetBatchLines</code>	Defaults to 1, which causes the script to sleep after every line.
<code>SetKeyDelay</code>	The default <i>Delay</i> for the traditional <code>SendEvent</code> mode is 20.
<code>Send</code>	The character <code>#</code> is ignored.

SplashTextOn

Height includes the window's title bar.

Escape Char Conversion

Running a script file with the extension `.aut .ahk` on [AutoHotkey v1.1.08.01] or earlier does not execute the script; it instead converts the script from the AutoIt v2 default escape character (backslash) to the AutoHotkey default (backtick).

Obsolete Commands

The following commands were supported by automatically translating them to the corresponding AutoHotkey commands (and this was previously undocumented): LeftClick, LeftClickDrag, RightClick, RightClickDrag, HideAutoItWin, Repeat, EndRepeat. This automatic translation has been removed in [v1.1.09].

Mouse Gestures -- by deguix

This script watches how you move the mouse whenever the right mouse button is being held down. If it sees you "draw" a recognized shape or symbol, it will launch a program or perform another custom action of your choice (just like hotkeys). See the included README file for how to define gestures.

[Download This Script \(17 KB Zip File\)](#) | [Other Sample Scripts](#) | [Home](#)

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**  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  
; pressing down the mouse button rather than  
holding it down the whole time.  
; This script requires v1.0.25+.
```

```
~MButton & LButton::  
CapsLock & LButton::  
CoordMode, Mouse ; Switch to screen/absolute  
coordinates.  
MouseGetPos, EWD_MouseStartX, EWD_MouseStartY,  
EWD_MouseWin  
WinGetPos, EWD_OriginalPosX, EWD_OriginalPosY,,  
ahk_id %EWD_MouseWin%  
WinGet, EWD_WinState, MinMax, ahk_id  
%EWD_MouseWin%  
if EWD_WinState = 0 ; Only if the window isn't  
maximized  
    SetTimer, EWD_WatchMouse, 10 ; Track the mouse  
as the user drags it.  
return
```

```
EWD_WatchMouse:  
GetKeyState, EWD_LButtonState, LButton, P  
if EWD_LButtonState = U ; Button has been
```

```

released, so drag is complete.
{
    SetTimer, EWD_WatchMouse, Off
    return
}
GetKeyState, EWD_EscapeState, Escape, P
if EWD_EscapeState = D ; Escape has been pressed,
so drag is cancelled.
{
    SetTimer, EWD_WatchMouse, Off
    WinMove, ahk_id %EWD_MouseWin%, ,
%EWD_OriginalPosX%, %EWD_OriginalPosY%
    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
WinGetPos, EWD_WinX, EWD_WinY, , , ahk_id
%EWD_MouseWin%
SetWinDelay, -1 ; Makes the below move
faster/smoother.
WinMove, ahk_id %EWD_MouseWin%, , EWD_WinX +
EWD_MouseX - EWD_MouseStartX, EWD_WinY +
EWD_MouseY - EWD_MouseStartY
EWD_MouseStartX := EWD_MouseX ; Update for the
next timer-call to this subroutine.
EWD_MouseStartY := EWD_MouseY
return

```

Easy Window Dragging -- KDE style (requires XP/2k/NT) -- by Jonny

This script makes it much easier to move or resize a window: 1) Hold down the **Alt** [] and LEFT-click anywhere inside a window to drag it to a new location; 2) Hold down **Alt** [] RIGHT-click-drag anywhere inside a window to easily resize it; 3) Press **Alt** [] but before releasing it the second time, left-click to minimize the window under the mouse cursor, right-click to maximize it, or middle-click to close it.

[Download This Script](#) | [Other Sample Scripts](#) | [Home](#)

```
; This script was inspired by and built on many
like it
; in the forum. Thanks go out to ck, thinkstorm,
Chris,
; and aurelian for a job well done.

; Change history:
; November 07, 2006: Optimized resizing code in
!RButton, courtesy of bluedawn.
; February 05, 2006: Fixed double-alt (the ~Alt
hotkey) to work with latest versions of AHK.

; The Double-Alt modifier is activated by pressing
; Alt twice, much like a double-click. Hold the
second
; press down until you click.
;
; The shortcuts:
; Alt + Left Button : Drag to move a window.
; Alt + Right Button : Drag to resize a window.
; Double-Alt + Left Button : Minimize a window.
; Double-Alt + Right Button : Maximize/Restore a
```

```
window.  
; Double-Alt + Middle Button : Close a window.  
;  
; You can optionally release Alt after the first  
; click rather than holding it down the whole  
time.
```

```
If (A_AhkVersion < "1.0.39.00")  
{  
    MsgBox,20,,This script may not work properly  
with your version of AutoHotkey. Continue?  
    IfMsgBox,No  
    ExitApp  
}
```

```
; This is the setting that runs smoothest on my  
; system. Depending on your video card and cpu  
; power, you may want to raise or lower this  
value.
```

```
SetWinDelay,2
```

```
CoordMode,Mouse  
return
```

```
!LButton::
```

```
If DoubleAlt
```

```
{
```

```
    MouseGetPos,,,KDE_id
```

```
    ; This message is mostly equivalent to
```

```
WinMinimize,
```

```
    ; but it avoids a bug with PSPad.
```

```
    PostMessage,0x112,0xf020,,,ahk_id %KDE_id%
```

```
    DoubleAlt := false
```

```
    return
```

```
}
```

```
; Get the initial mouse position and window id,
```

```

and
; abort if the window is maximized.
MouseGetPos,KDE_X1,KDE_Y1,KDE_id
WinGet,KDE_Win,MinMax,ahk_id %KDE_id%
If KDE_Win
    return
; Get the initial window position.
WinGetPos,KDE_WinX1,KDE_WinY1,,ahk_id %KDE_id%
Loop
{
    GetKeyState,KDE_Button,LButton,P ; Break if
button has been released.
    If KDE_Button = U
        break
    MouseGetPos,KDE_X2,KDE_Y2 ; Get the current
mouse position.
    KDE_X2 -= KDE_X1 ; Obtain an offset from the
initial mouse position.
    KDE_Y2 -= KDE_Y1
    KDE_WinX2 := (KDE_WinX1 + KDE_X2) ; Apply this
offset to the window position.
    KDE_WinY2 := (KDE_WinY1 + KDE_Y2)
    WinMove,ahk_id
%KDE_id%,,%KDE_WinX2%,%KDE_WinY2% ; Move the
window to the new position.
}
return

!RButton::
If DoubleAlt
{
    MouseGetPos,,,KDE_id
; Toggle between maximized and restored state.
WinGet,KDE_Win,MinMax,ahk_id %KDE_id%
If KDE_Win
    WinRestore,ahk_id %KDE_id%
Else

```

```

        WinMaximize,ahk_id %KDE_id%
    DoubleAlt := false
    return
}
; Get the initial mouse position and window id,
and
; abort if the window is maximized.
MouseGetPos,KDE_X1,KDE_Y1,KDE_id
WinGet,KDE_Win,MinMax,ahk_id %KDE_id%
If KDE_Win
    return
; Get the initial window position and size.
WinGetPos,KDE_WinX1,KDE_WinY1,KDE_WinW,KDE_WinH,ahk_id %KDE_id%
; Define the window region the mouse is currently
in.
; The four regions are Up and Left, Up and Right,
Down and Left, Down and Right.
If (KDE_X1 < KDE_WinX1 + KDE_WinW / 2)
    KDE_WinLeft := 1
Else
    KDE_WinLeft := -1
If (KDE_Y1 < KDE_WinY1 + KDE_WinH / 2)
    KDE_WinUp := 1
Else
    KDE_WinUp := -1
Loop
{
    GetKeyState,KDE_Button,RButton,P ; Break if
button has been released.
    If KDE_Button = U
        break
    MouseGetPos,KDE_X2,KDE_Y2 ; Get the current
mouse position.
    ; Get the current window position and size.

WinGetPos,KDE_WinX1,KDE_WinY1,KDE_WinW,KDE_WinH,ah

```

```

k_id %KDE_id%
    KDE_X2 -= KDE_X1 ; Obtain an offset from the
initial mouse position.
    KDE_Y2 -= KDE_Y1
    ; Then, act according to the defined region.
    WinMove,ahk_id %KDE_id%,, KDE_WinX1 +
(KDE_WinLeft+1)/2*KDE_X2 ; X of resized window
, KDE_WinY1 +
(KDE_WinUp+1)/2*KDE_Y2 ; Y of resized window
, KDE_WinW -
KDE_WinLeft *KDE_X2 ; W of resized window
, KDE_WinH -
KDE_WinUp *KDE_Y2 ; H of resized window
    KDE_X1 := (KDE_X2 + KDE_X1) ; Reset the
initial position for the next iteration.
    KDE_Y1 := (KDE_Y2 + KDE_Y1)
}
return

; "Alt + MButton" may be simpler, but I
; like an extra measure of security for
; an operation like this.
!MButton::
If DoubleAlt
{
    MouseGetPos,,,KDE_id
    WinClose,ahk_id %KDE_id%
    DoubleAlt := false
    return
}
return

; This detects "double-clicks" of the alt key.
~Alt::
DoubleAlt := A_PriorHotkey = "~Alt" AND
A_TimeSincePriorHotkey < 400
Sleep 0

```

```
KeyWait Alt ; This prevents the keyboard's auto-  
repeat feature from interfering.  
return
```

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 MButton, the
tilde (~) prefix
; is used so that MButton'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.
f_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      ; %A_Desktop%
Favorites    ; %A_Desktop%\..\Favorites
My Documents ; %A_MyDocuments%
```

```
Program Files; %A_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
if f_HotkeyFirstChar = ~ ; Show menu only for
certain window types.
    f_AlwaysShowMenu = n
else
    f_AlwaysShowMenu = y

; 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.
f_AtStartingPos = n
f_MenuItemCount = 0
Loop, Read, %f_FavoritesFile%
{
    if f_FileExt <> Exe
    {
        ; 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

```

```

        {
            IfInString, 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:
    f_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
f_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 ahk_id %f_window_id%
        ; Retrieve any filename that might already
be in the field so
; 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%
        ; Tek1 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 mclick deactivates it.
    SetKeyDelay, 0 ; This will be in effect only
for the duration of this thread.
    IfInString, 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
f_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_Edit1Pos = ; 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
```

IntelliSense -- by Rajat (requires XP/2k/NT)

This script watches while you edit an AutoHotkey script. When it sees you type a command followed by a comma or space, it displays that command's parameter list to guide you. In addition, you can press **Ctrl** **F1** (another hotkey of your choice) to display that command's page in the help file. To dismiss the parameter list, press **Escape** **Enter**

[Download This Script](#) | [Other Sample Scripts](#) | [Home](#)

```
; Requires v1.0.41+

; CONFIGURATION SECTION: Customize the script with
the following variables.

; The hotkey below is pressed to display the
current command's page in the
; help file:
I_HelpHotkey = ^F1

; The string below must exist somewhere in the
active window's title for
; IntelliSense to be in effect while you're
typing. Make it blank to have
; IntelliSense operate in all windows. Make it
Pad to have it operate in
; editors such as Metapad, Notepad, and Textpad.
Make it .ahk to have it
; operate only when a .ahk file is open in
Notepad, Metapad, etc.
I_Editor = pad

; If you wish to have a different icon for this
```

```

script to distinguish it from
; other scripts in the tray, provide the filename
below (leave blank to have
; no icon). For example:
E:\stuff\Pics\icons\GeoIcons\Information.ico
I_Icon =

; END OF CONFIGURATION SECTION (do not make
changes below this point unless
; you want to change the basic functionality of
the script).

SetKeyDelay, 0
#SingleInstance

if I_HelpHotkey <>
    Hotkey, %I_HelpHotkey%, I_HelpHotkey

; Change tray icon (if one was specified in the
configuration section above):
if I_Icon <>
    IfExist, %I_Icon%
        Menu, Tray, Icon, %I_Icon%

; Determine AutoHotkey's location:
RegRead, ahk_dir, HKEY_LOCAL_MACHINE,
SOFTWARE\AutoHotkey, InstallDir
if ErrorLevel ; Not found, so look for it in some
other common locations.
{
    if A_AhkPath
        SplitPath, A_AhkPath,, ahk_dir
    else IfExist ..\..\AutoHotkey.chm
        ahk_dir = ..\..
    else IfExist
%A_ProgramFiles%\AutoHotkey\AutoHotkey.chm
        ahk_dir = %A_ProgramFiles%\AutoHotkey

```

```

else
{
    MsgBox Could not find the AutoHotkey
folder.
    ExitApp
}
}

ahk_help_file = %ahk_dir%\AutoHotkey.chm

; Read command syntaxes:
Loop, Read,
%ahk_dir%\Extras\Editors\Syntax\Commands.txt
{
    I_FullCmd = %A_LoopReadLine%

    ; Directives have a first space instead of a
first comma.
    ; So use whichever comes first as the end of
the command name:
    StringGetPos, I_cPos, I_FullCmd, `,
    StringGetPos, I_sPos, I_FullCmd, %A_Space%
    if (I_cPos = -1 or (I_cPos > I_sPos and I_sPos
<> -1))
        I_EndPos := I_sPos
    else
        I_EndPos := I_cPos

    if I_EndPos <> -1
        StringLeft, I_CurrCmd, I_FullCmd,
%I_EndPos%
    else ; This is a directive/command with no
parameters.
        I_CurrCmd = %A_LoopReadLine%

    StringReplace, I_CurrCmd, I_CurrCmd, [, , All
StringReplace, I_CurrCmd, I_CurrCmd,

```

```

%A_Space%, , All
    StringReplace, I_FullCmd, I_FullCmd, ``n, `n,
All
    StringReplace, I_FullCmd, I_FullCmd, ``t, `t,
All

    ; Make arrays of command names and full cmd
syntaxes:
    I_Cmd%A_Index% = %I_CurrCmd%
    I_FullCmd%A_Index% = %I_FullCmd%
}

; Use the Input command to watch for commands that
the user types:
Loop
{
    ; Editor window check:
    WinGetTitle, ActiveTitle, A
    IfNotInString, ActiveTitle, %I_Editor%
    {
        ToolTip
        Sleep, 500
        Continue
    }

    ; Get all keys till endkey:
    Input, I_Word, V, {enter}{escape}{space}`,
    I_EndKey = %ErrorLevel%

    ; ToolTip is hidden in these cases:
    if I_EndKey in EndKey:Enter,EndKey:Escape
    {
        ToolTip
        Continue
    }

    ; Editor window check again!

```

```

WinGetActiveTitle, ActiveTitle
IfNotInString, ActiveTitle, %I_Editor%
{
    ToolTip
    Continue
}

; Compensate for any indentation that is
present:
StringReplace, I_Word, I_Word, %A_Space%, , All
StringReplace, I_Word, I_Word, %A_Tab%, , All
if I_Word =
    Continue

; Check for commented line:
StringLeft, I_Check, I_Word, 1
if (I_Check = ";" or I_Word = "If") ; "If"
seems a little too annoying to show tooltip for.
    Continue

; Match word with command:
I_Index =
Loop
{
    ; It helps performance to resolve dynamic
variables only once.
    ; In addition, the value put into
I_ThisCmd is also used by the
; I_HelpHotkey subroutine:
I_ThisCmd := I_Cmd%A_Index%
if I_ThisCmd =
    break
if (I_Word = I_ThisCmd)
{
    I_Index := A_Index
    I_HelpOn = %I_ThisCmd%
    break
}
}

```

```

    }
}

; If no match then resume watching user input:
if I_Index =
    Continue

; Show matched command to guide the user:
I_ThisFullCmd := I_FullCmd%I_Index%
ToolTip, %I_ThisFullCmd%, A_CaretX, A_CaretY +
20
}

```

```

I_HelpHotkey:
WinGetTitle, ActiveTitle, A
IfNotInString, ActiveTitle, %I_Editor%, Return

```

```

ToolTip ; Turn off syntax helper since there is
no need for it now.

```

```

SetTitleMatchMode, 1 ; In case it's 3. This
setting is in effect only for this thread.
IfWinNotExist, AutoHotkey Help

```

```

{
    IfNotExist, %ahk_help_file%
    {
        MsgBox, Could not find the help file:
%ahk_help_file%.
        return
    }
    Run, %ahk_help_file%
    WinWait, AutoHotkey Help
}

```

```

if I_ThisCmd = ; Instead, use what was most

```

recently typed.

```
I_ThisCmd := I_Word
```

; The above has set the "last found" window which we use below:

```
WinActivate
```

```
WinWaitActive
```

```
StringReplace, I_ThisCmd, I_ThisCmd, #, {#} ;
```

Replace leading #, if any.

```
Send, !n{home}+{end}%I_HelpOn%{enter}
```

```
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:
k_FontSize = 10
k_FontName = Verdana ; This can be blank to use
the system's default font.
k_FontStyle = Bold ; Example of an alternative:
Italic Underline

; Names for the tray menu items:
k_MenuItemHide = Hide on-screen &keyboard
k_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:
k_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:
```

```
k_KeyWidth = %k_FontSize%
k_KeyWidth *= 3
k_KeyHeight = %k_FontSize%
k_KeyHeight *= 3
k_KeyMargin = %k_FontSize%
k_KeyMargin /= 6
k_SpacebarWidth = %k_FontSize%
k_SpacebarWidth *= 25
k_KeyWidthHalf = %k_KeyWidth%
k_KeyWidthHalf /= 2
```

```
k_KeySize = w%k_KeyWidth% h%k_KeyHeight%
k_Position = x+%k_KeyMargin% %k_KeySize%
```

```
;---- Create a GUI window for the on-screen
keyboard:
```

```
Gui, Font, s%k_FontSize% %k_FontStyle%,
%k_FontName%
Gui, -Caption +E0x200 +ToolWindow
TransparentColor = F1ECED
Gui, Color, %TransparentColor% ; 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%, F
Gui, Add, Button, %k_Position%, G
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:
k_WindowY = %k_WorkAreaBottom%
k_WindowY -= %k_WindowHeight%

WinMove, A,, %k_WindowX%, %k_WindowY%
WinSet, AlwaysOnTop, On, ahk_id %k_ID%
WinSet, TransColor, %TransColor% 220, ahk_id
%k_ID%

```

```

;---- Set all keys as hotkeys. See
www.asciitable.com
k_n = 1
k_ASCII = 45

Loop
{
    Transform, k_char, Chr, %k_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
    k_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:: ; Must use Ctrl not Control to match  
button names.
```

```
~*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::
```

```
k_KeyPress:
```

```
StringReplace, k_ThisHotkey, A_ThisHotkey, ~
```

```
StringReplace, k_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, U
```

```
Return
```

```
k_ShowHide:
if 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:
k_MenuExit:
ExitApp
```

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 unhidden 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 unhidden automatically.

[Download This Script](#) | [Other Sample Scripts](#) | [Home](#)

```
; CHANGES:
; July 22, 2005 (changes provided by egilmour):
; - Added new hotkey to unhide the last hidden
window (Win+U)
;
; November 3, 2004 (changes provided by trogdor):
; - 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

; This is the hotkey used to unhide the last
hidden window:
mwt_UnHotkey = #u ; Win+U

; 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
you choose to have the
; script use the keyboard hook via
#InstallKeybdHook 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
Hotkey, %mwt_UnHotkey%, mwt_UnMinimize
```

```
; 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, E&xit and Unhide All,
mwt_RestoreAllThenExit
}
Menu, Tray, Add, &Unhide 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 has 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,Progman
{
    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, those 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 +=
%mw_ActiveIDShortLength%
        mwt_ActiveTitleLength += 1 ; +1 the 1
space between title & ID.
        if mwt_ActiveTitleLength > %mw_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 = %mw_ActiveTitleLength%
            TrimCount -= %mw_MaxLength%
            StringTrimRight, mwt_ActiveTitle,
mwt_ActiveTitle, %TrimCount%
        }
        ; Build unique title:
        mwt_ActiveTitle = %mw_ActiveTitle%
%mw_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, %mw_MaxWindows%
{
    if mwt_WindowID%A_Index% = %mw_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, %A_ThisMenuItem%
; Find window based on its unique title stored as
the menu item name:
Loop, %mwt_MaxWindows%
{
    if mwt_WindowTitle%A_Index% = %A_ThisMenuItem%
; Match found.

```

```

    {
        StringTrimRight, IDToRestore,
mwt_WindowID%A_Index%, 0
        WinShow, ahk_id %IDToRestore%
        WinActivate ahk_id %IDToRestore% ;
Sometimes needed.
        mwt_WindowID%A_Index% = ; Make it blank
to free up a slot.
        mwt_WindowTitle%A_Index% =
        mwt_WindowCount -= 1
        break
    }
}
return

;; This will pop the last minimized window off the
stack and unhide it.
mwt_UnMinimize:
;; Make sure there's something to unhide.
if mwt_WindowCount > 0
{
    ;; Get the id of the last window minimized and
unhide it
        StringTrimRight, IDToRestore,
mwt_WindowID%mwt_WindowCount%, 0
        WinShow, ahk_id %IDToRestore%
        WinActivate ahk_id %IDToRestore%

    ;; Get the menu name of the last window
minimized and remove it
        StringTrimRight, MenuToRemove,
mwt_WindowTitle%mwt_WindowCount%, 0
        Menu, Tray, delete, %MenuToRemove%

    ;; clean up our 'arrays' and decrement the
window count

```

```

    mwt_WindowID%mwt_WindowCount% =
    mwt_WindowTitle%mwt_WindowCount% =
    mwt_WindowCount -= 1
}
return

mwt_RestoreAllThenExit:
Gosub, mwt_RestoreAll
ExitApp ; Do a true exit.

mwt_RestoreAll:
Loop, %mwt_MaxWindows%
{
    if mwt_WindowID%A_Index% <>
    {
        StringTrimRight, IDToRestore,
mwt_WindowID%A_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, MenuToRemove,
mwt_WindowTitle%A_Index%, 0
        Menu, Tray, delete, %MenuToRemove%
        mwt_WindowID%A_Index% = ; Make it blank
to free up a slot.
        mwt_WindowTitle%A_Index% =
        mwt_WindowCount -= 1
    }
    if mwt_WindowCount = 0
        break
}
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 : Mon Jan 31 10:08:37 2005  
 ;  
 ; 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
; -scex 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/2004-10/188589/
; 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 200 kbytes. I can be reached at:
phi1618 (*a.t*)
; gmail :DOT: com.
;
; To use Seek.ahk, first, you'll need to
```

```
install
;   AutoHotkey v1.0.25 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:
; - %A_StartMenu%
; - %A_StartMenuCommon%
;
; 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
v1.0.25
; or higher version (http://www.autohotkey.com).
; Thanks to Chris for his great work on
AutoHotkey. :)
;
; - The following environment variables must be
valid:
; a. TMP
;
```

```

;*****
;
;*****
;
;
; KNOWN PROBLEMS:
;
; - Nil
;
;*****
;*****
;
;
; 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)
;
; - Fuzzy search when user enters multiple query
strings,
; separated by space.
; (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
;   those 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)
;
; - Search by wildcards/regex.
;   (Suggested by Steve)
;
;
; *****
```

```
*****
;
; 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 (TrackKeyPhrase) to
track search-string.
; - Added internal switch (ToolTipFilename) to
show filename
; using tooltip.
;
; * 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.
;
; * v2.0.3
; - Added /on option to DIR-command to sort by
name.
; - Fuzzy search when user enters multiple query
strings,
; separated by space, for e.g. "med pla". It's a
match
; when all the strings ("med" & "pla") are
found. This
; will match "Media Player", "Macromedia Flash
Player",
; "Play Medieval King", "medpla", "plamed".
; - Corrected tab-movement sequence by adding all
buttons
; right from the start, but disable them until
they can
; be used.
; - Added status bar to replace tooltip-feedback.
; - Removed obsolete internal switch
(ToolTipFilename).
; - Replaced the use of "dir" command with
AutoHotkey's
```

```
; own "Loop" command for scanning directory
contents.
; "dir" cannot handle extended character set and
thus
; non-English (e.g German) directory and
filename are
; captured wrongly. (Thanks Wolfgang Bujatti and
; Sietse Fliege for testing this modification)
; - Added internal switch (ScanMode) to define
whether
; files and/or directories are to be included in
scan.
; - Replaced hardcoded directory paths of Start
Menu with
; built-in variables A_StartMenu,
A_StartMenuCommon.
; With this, Seek now works for different
locales with
; different naming convention of the Start Menu.
; (Thanks Wolfgang Bujatti and Sietse Fliege for
help
; in testing another method before these new
variables
; are available.)
; - Added the pre-selection of the last-run
program
; record so that a quick double-<ENTER> will run
it.
;
;
;*****
;
;*****

;*****
;
;<--- 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:\utl\xplorer2_lite\xplorer2.exe

; User's customised list of additional directories
to be
; included in the scanning. The full path must not
be
; enclosed by quotes or double-quotes. 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.
dirListing = %A_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 = %A_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

; Specify what should be included in scan.
; 0: Directories are excluded (only files).
; 1: All files and directories are included.
; 2: Only directories are included (no files).
ScanMode = 1

;.....
.....

; INIT
;#NoTrayIcon
StringCaseSense, Off
version = Seek v2.0.3

; DISPLAY HELP INSTRUCTIONS
If 1 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.`n`nI 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.`n`nOptions:`n  -cache`tUse the
cached directory-listing if available (this is the
default mode when no option is specified)`n  -
scan`tForce a directory scan to retrieve the
latest directory listing`n  -scex`tScan & exit
(this is useful for scheduling the potentially
time-consuming directory-scanning as a background
job)`n  -help`tShow this help
```

```
    Goto QuitNoSave
```

```
}
```

```
; CHECK THAT THE MANDATORY ENVIRONMENT VARIABLES
EXIST AND ARE VALID
```

```
; *TMP*
```

```
IfNotExist, %A_Temp% ; PATH DOES NOT EXIST
```

```
{
```

```
    MsgBox This mandatory environment variable is
either not defined or invalid:`n`n    TMP =
%A_Temp%`n`nPlease fix it before running Seek.
```

```
    Goto QuitNoSave
```

```
}
```

```
; IF NOT SCAN-AND-EXIT
```

```
IfNotEqual 1, -scex
```

```
{
```

```
    ; RETRIEVE THE LAST USED KEY-PHRASE FROM CACHE
FILE
```

```
    ; TO BE USED AS THE DEFAULT QUERY STRING
```

```
If TrackKeyPhrase = ON
```

```
{
```

```
    FileReadLine, PrevKeyPhrase, %keyPhrase%,
```

```
1
```

```
    FileReadLine, PrevOpenTarget, %keyPhrase%,
```

```
2
```

```
}
```

```

NewKeyPhrase = %PrevKeyPhrase%
NewOpenTarget = %PrevOpenTarget%

; ADD THE TEXT BOX FOR USER TO ENTER THE QUERY
STRING
Gui, 1:Add, Edit, vFilename W600,
%NewKeyPhrase%

; ADD MY FAV TAGLINE
Gui, 1:Add, Text, X625 Y10, What do you seek,
my friend?

; ADD THE STATUS BAR FOR PROVIDING FEEDBACK TO
USER
Gui, 1:Add, Text, vStatusBar X10 Y31 R1 W764

; ADD THE SELECTION LISTBOX FOR DISPLAYING
SEARCH RESULTS
Gui, 1:Add, ListBox, vOpenTarget
gTargetSelection X10 Y53 R28 W764 HScroll
Disabled, %List%

; ADD THESE BUTTONS, BUT DISABLE THEM FOR NOW
Gui, 1:Add, Button, gButtonOPEN vButtonOPEN
Default X10 Y446 Disabled, Open
Gui, 1:Add, Button, gButtonOPENDIR
vButtonOPENDIR X59 Y446 Disabled, Open Directory
Gui, 1:Add, Button, gButtonSCANSTARTMENU
vButtonSCANSTARTMENU X340 Y446 Disabled, Scan
Start-Menu

; ADD THE EXIT BUTTON
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,-scex
    rescan = Y
; CHECK WHETHER THE DIRECTORY LISTING CACHE FILE
ALREADY EXISTS. IF NOT, DO A RE-SCAN.
Else IfNotExist, %dirListing%
    rescan = Y

If rescan = Y ; DO A RE-SCAN
{
    ; SHOW STATUS UNLESS USER SPECIFIES SCAN-AND-
EXIT OPTION
    IfNotEqual 1, -scex
        GuiControl,, StatusBar, Scanning directory
listing...

        ; SCAN START-MENU AND STORE DIRECTORY/PROGRAM
LISTINGS IN CACHE FILE
        Gosub ScanStartMenu

        ; QUIT IF USER SPECIFIES SCAN-AND-EXIT OPTION
        IfEqual 1, -scex, Goto, QuitNoSave
}

GuiControl,, StatusBar, Retrieving last query
result...

; RETRIEVE THE MATCHING LIST FOR THE LAST USED
KEY-PHRASE
Gosub SilentFindMatches

; REMOVE THE STATUS TEXT
GuiControl,, StatusBar,

; DIRECTORY LISTING IS NOW LOADED. ENABLE THE

```

```
OTHER BUTTONS.  
; THESE BUTTONS ARE DISABLED EARLIER BECAUSE THEY  
SHOULD NOT  
; BE FUNCTIONAL UNTIL THIS PART OF THE SCRIPT.  
GuiControl, 1:Enable, ButtonOPEN  
GuiControl, 1:Enable, ButtonOPENDIR  
GuiControl, 1:Enable, ButtonSCANSTARTMENU
```

```
; TURN ON INCREMENTAL SEARCH  
SetTimer, tIncrementalSearch, 500
```

```
; REFRESH THE GUI  
Gosub EnterQuery
```

```
Return
```

```
.*****  
/  
*****
```

```
.  
/  
*
```

```
;  
/                               END OF MAIN PROGRAM  
*
```

```
.  
/  
*
```

```
.*****  
/  
*****
```

```
;=== BEGIN ButtonSCANSTARTMENU EVENT  
=====
```

```
ButtonSCANSTARTMENU:
```

```
Gui, 1:Submit, NoHide  
GuiControl,, StatusBar, Scanning directory  
listing...
```

```

; DISABLE LISTBOX WHILE SCANNING IS IN PROGRESS
GuiControl, 1:Disable, OpenTarget
GuiControl, 1:Disable, ButtonEXIT
GuiControl, 1:Disable, ButtonOPEN
GuiControl, 1:Disable, ButtonOPENDIR
GuiControl, 1:Disable, ButtonSCANSTARTMENU

; DO THE SCANNING
Gosub ScanStartMenu

; INFORM USER THAT SCANNING HAS COMPLETED
If Filename =
{
    ; IF QUERY STRING IS EMPTY...
    GuiControl, 1:Enable, ButtonEXIT
    GuiControl, 1:Enable, ButtonOPEN
    GuiControl, 1:Enable, ButtonSCANSTARTMENU
    GuiControl,, StatusBar, Scan completed.
    Gosub EnterQuery
}
Else
{
    ; IF QUERY STRING EXISTS...
    ; FILTER FOR SEARCH STRING WITH THE NEW
LISTING
    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.
; THE PATH MUST NOT BE ENCLOSED BY QUOTES OR
DOUBLE-QUOTES.
;
; FOR ENGLISH VERSION OF WINDOWS
scanPath = %A_StartMenu%|%A_StartMenuCommon%

; INCLUDE ADDITIONAL USER-DEFINED PATHS FOR
SCANNING
IfExist, %SeekMyDir%
{
    Loop, read, %SeekMyDir%
    {
        IfNotExist, %A_LoopReadLine%
            MsgBox, 8192, %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%
    }
}

; DELETE EXISTING FILE BEFORE CREATING A NEW
VERSION
FileDelete, %dirListing%

; SCAN DIRECTORY LISTING (DELIMITER = |) BY
RECURSING
; EACH DIRECTORY TO RETRIEVE THE CONTENTS. HIDDEN
FILES
; ARE EXCLUDED.

```

```

Loop, parse, scanPath, |
{
    Loop, %A_LoopField%\*, %ScanMode%, 1
    {
        FileGetAttrib, fileAttrib,
%A_LoopFileFullPath%
        IfNotInString, fileAttrib, H ; EXCLUDE
HIDDEN FILE
        FileAppend, %A_LoopFileFullPath%\n,
%dirListing%
    }
}

Return

;... END ScanStartMenu SUBROUTINE
.....

;=== BEGIN FindMatches SUBROUTINE
=====
; SEARCH AND DISPLAY ALL MATCHING RECORDS IN THE
LISTBOX
FindMatches:

Gui, 1:Submit, NoHide
CurFilename = %Filename%
GuiControl,, StatusBar,

; CHECK FOR EMPTY QUERY STRING
If CurFilename =
{
    MsgBox, 8192, %version%, Please enter the key
word/phrase to search for.
    Goto EnterQuery
}

```

```

; tIncrementalSearch IS BEING INTERRUPTED. LET IT
FINISHES.
If NewKeyPhrase <> %CurFilename%
{
    ; INFORM USER THAT PATIENCE IS A VIRTUE
    GuiControl,, StatusBar, Seeking...
    ResumeFindMatches = TRUE
    Return
}

If List = |
{
    ; NOT EVEN A SINGLE MATCHING RECORD IS FOUND.
    ; LET USER MODIFY THE QUERY STRING AND TRY
    AGAIN.
    MsgBox, 8192, %version%, The query string
"%CurFilename%" does not match any record. Try
again.
    GuiControl, 1:Disable, ButtonOPENDIR
    GuiControl, 1:Enable, ButtonSCANSTARTMENU
    Goto EnterQuery
}
Else
{
    ; SELECT THE FIRST RECORD IF NO OTHER RECORD
    HAS BEEN SELECTED
    Gui, 1:Submit, NoHide
    GuiControl, 1:Enable, OpenTarget
    GuiControl, 1:Enable, ButtonOPEN
    GuiControl, 1:Enable, ButtonOPENDIR
    GuiControl, 1:Enable, ButtonSCANSTARTMENU
    GuiControl, Focus, OpenTarget
    If OpenTarget =
        GuiControl, 1:Choose, OpenTarget, |1
}

; REFRESH GUI

```

```

Gui, 1:Show, Center, %version%

Return

;... END FindMatches SUBROUTINE
.....

;=== BEGIN SilentFindMatches SUBROUTINE
=====

SilentFindMatches:

Gui, 1:Submit, NoHide
sfmFilename = %Filename%

; FILTER MATCHING RECORDS BASED ON USER QUERY
STRING
List = |
If sfmFilename <>
{
    Loop, read, %dirListing%
    {
        Gui, 1:Submit, NoHide
        tFilename = %Filename%
        If sfmFilename <> %tFilename%
        {
            ; 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
        MatchFound = Y
        Loop, parse, sfmFilename, %A_Space%
        {
            IfNotInString, name, %A_LoopField%
            {
                MatchFound = N
                Break
            }
        }
        IfEqual, MatchFound, Y
        {
            ; ADD RECORD TO LIST
            List = %List%%A_LoopReadLine|

            ; PRE-SELECT IF THIS MATCHES THE
LAST-RUN PROGRAM
            If (A_LoopReadLine =
PrevOpenTarget && sfmFilename = PrevKeyPhrase)
                List = %List%|
        }
    }
}

; REFRESH LIST WITH SEARCH RESULTS
GuiControl, 1:, OpenTarget, %List%

If List = |
{
    ; NO MATCHING RECORD IS FOUND
    ; DISABLE LISTBOX
    GuiControl, 1:Disable, OpenTarget
    GuiControl, 1:Disable, ButtonOPENDIR
}
Else

```

```

{
    ; MATCHING RECORDS ARE FOUND
    ; ENABLE LISTBOX
    GuiControl, 1:Enable, OpenTarget
    GuiControl, 1:Enable, ButtonOPENDIR
}

; 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
GuiControl, 1:Enable, ButtonOPEN
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
{

```

```

    Gosub ButtonOPEN
}
Else
{
    ; STUB - FOR FUTURE USE
    If A_GuiControlEvent = Normal
    {
        ; DO NOTHING FOR NOW
    }
}

Return

;... END TargetSelection EVENT
.....

;=== BEGIN ButtonOPEN EVENT
=====

; USER CLICKED ON 'OPEN' BUTTON OR PRESSED <ENTER>
ButtonOPEN:
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.
GuiControlGet, focusControl, 1:Focus
If focusControl = Edit1
{
    GuiControl, Focus, OpenTarget
    GuiControl, 1:Disable, OpenTarget
    GuiControl, 1:Disable, ButtonOPENDIR
    GuiControl, 1:Disable, ButtonSCANSTARTMENU
    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)
IfNotExist, %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, %OpenTarget%
}
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, 1:Submit, NoHide
```

```
; CHECK THAT USER HAS SELECTED A RECORD ALREADY
```

```
If OpenTarget =
```

```
{
```

```
    MsgBox, 8192, %version%, Please make a  
    selection first.
```

```
    Goto EnterQuery
```

```
}
```

```
; RUN SUBROUTINE TO OPEN A DIRECTORY
```

```
Gosub sOpenDir
```

```
Goto Quit
```

```
;... END ButtonOPENDIR EVENT
```

```
.....
```

```
;=== BEGIN sOpenDir SUBROUTINE
```

```
=====
```

sOpenDir:

```
; 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, %OpenTarget%
If status <> Ready ; NOT A DIRECTORY
{
    SplitPath, OpenTarget, name, dir, ext,
name_no_ext, drive
    OpenTarget = %dir%
}

; CHECK WHETHER DIRECTORY EXISTS
IfNotExist, %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
}

; 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 sOpenDir SUBROUTINE
.....

;=== BEGIN tIncrementalSearch EVENT
=====
; AUTOMATICALLY CONDUCT REAL-TIME INCREMENTAL
SEARCH
; TO FIND MATCHING RECORDS WITHOUT WAITING FOR
USER
; TO PRESS <ENTER>
tIncrementalSearch:

Loop
; REPEAT SEARCHING UNTIL USER HAS STOPPED CHANGING
THE QUERY STRING
{
    Gui, 1:Submit, NoHide
    CurFilename = %Filename%
    If NewKeyPhrase <> %CurFilename%
    {
        OpenTarget =
        Gosub 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.
If ResumeFindMatches = TRUE
{
    ResumeFindMatches = FALSE
    Gosub FindMatches
}

; CONTINUE MONITORING FOR CHANGES
SetTimer, tIncrementalSearch, 500

Return

;... END tIncrementalSearch EVENT
.....

;=== BEGIN Quit SUBROUTINE
=====

Quit:
ButtonEXIT:
GuiClose:
GuiEscape:

Gui, 1:Submit, NoHide

; SAVE THE KEY WORD/PHRASE FOR NEXT RUN IF IT HAS
CHANGED
If TrackKeyPhrase = ON
{
    If (PrevKeyPhrase <> Filename ||
PrevOpenTarget <> OpenTarget)
    {
        FileDelete, %keyPhrase%
    }
}

```

```
        FileAppend, %Filename%\`n, %keyPhrase%
        FileAppend, %OpenTarget%\`n, %keyPhrase%
    }
}

QuitNoSave:
ExitApp ; JOB DONE. G'DAY!

;... END Quit SUBROUTINE
.....

;*****
;
;<--- END OF PROGRAM --->
;*****
;

; /* vim: set noexpandtab shiftwidth=4: */
```

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).

[Download This Script](#) | [Other Sample Scripts](#) | [Home](#)

```
; 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:
UMDelay = 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.

MenuItems = 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

MSWord:

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, U, 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, MenuItems, D/
TempMenu = %MenuItems%%DynMenu%

;clears earlier entries
Loop
{
    IfEqual, MenuItem%A_Index%, , Break
    MenuItem%A_Index% =
}

;creates new entries
Loop, Parse, TempMenu, /
{
    MenuItem%A_Index% = %A_LoopField%
}

;creates the menu
Menu = %MenuTitle%
Loop
{
    IfEqual, MenuItem%A_Index%, , Break
    numItems ++
    StringTrimLeft, MenuText, MenuItem%A_Index%, 0
    Menu = %Menu%`n%MenuText%
}

MouseGetPos, mX, mY

```

```
Hotkey, ~LButton, MenuClick
Hotkey, ~LButton, On
ToolTip, %Menu%, %mX%, %mY%
WinActivate, %MenuTitle%
Return
```

```
MenuClick:
Hotkey, ~LButton, Off
IfWinNotActive, %MenuTitle%
{
    ToolTip
    Return
}
```

```
MouseGetPos, mX, mY
ToolTip
mY -= 3           ;space after which first line
starts
mY /= 13        ;space taken by each line
IfLess, mY, 1, Return
IfGreater, mY, %numItems%, Return
StringTrimLeft, TargetSection, MenuItem%mY%, 0
StringReplace, TargetSection, TargetSection,
%A_Space%, , A
Gosub, %TargetSection%
Return
```

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_CBM = 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_Thick% ZX0 ZY0
W%vol_Width% CB%vol_CBM% CW%vol_CW%
vol_BarOptionsWave = 2:B ZH%vol_Thick% ZX0 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%
Y%vol_PosY%
    vol_PosY_wave = %vol_PosY%
    vol_PosY_wave += %vol_Thick%
    vol_BarOptionsWave = %vol_BarOptionsWave%
Y%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:
IfWinNotExist, vol_Wave
    Progress, %vol_BarOptionsWave%, , , vol_Wave
IfWinNotExist, 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:
        WinGetPos, , vol_Wave_Posy, , , vol_Wave
        vol_Wave_Posy -= %vol_Thick%
        Progress, %vol_BarOptionsMaster%
Y%vol_Wave_Posy%, , , vol_Master
    }
    else
        Progress, %vol_BarOptionsMaster%, , ,
vol_Master
}
; Get both volumes in case the user or an external
program changed them:
SoundGet, vol_Master, Master
SoundGet, 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
```

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

#z:: ; 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, A
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.

```

AutoHotkey_L New Features

This document describes features added in the AutoHotkey_L branch of AutoHotkey development, now simply known as "AutoHotkey 1.1".

Caution: This document has been neglected in recent years and may be missing some of the more recent changes.

Control Flow

<i>Break LoopLabel</i>	Break out of a loop or any number of nested loops.
<i>Continue LoopLabel</i>	Continue a loop, even from within any number of nested loops.
<i>For x,y in z</i>	Loop through the contents of an object.
<i>Loop Until</i>	Loop until a condition is true. Applicable to any type of Loop.
<i>Try...Catch...Finally</i>	Provides structured exception handling.
<i>Throw</i>	Throws an exception.

Commands

FileEncoding	Sets the default encoding for FileRead , FileReadLine , Loop Read , FileAppend , and FileOpen . <i>See also:</i> Text Encodings
Gui	See GUI Enhancements below.
IniRead/Write/Delete	Read, write or delete entire sections, or retrieve a list of all section names.
Menu, Icon	Sets or removes a menu item's icon.
Run	Improvements were made to the way parameters are parsed.
SendInput {U+nnnn}	Sends a Unicode character. Unicode characters may be used directly in Unicode builds.
SendLevel	Controls which artificial keyboard and mouse events are ignored by hotkeys and hotstrings.
SetFormat, IntegerFast, h H	Set lower-case or upper-case hexadecimal format.
SetRegView, RegView	Allows registry commands in a 32-bit script to access the 64-bit registry view and vice versa.
Transform, HTML	Perform code page or HTML transformations.
WinGet, ..., ProcessPath	Retrieves the full path and name of the process that owns a given window.

Directives

<code>#If <i>expression</i></code>	Similar to <code>#IfWinActive</code> , but for arbitrary expressions.
<code>#IfTimeout</code>	Sets the maximum time that may be spent evaluating a single <code>#If</code> expression.
<code>#MenuMaskKey</code>	Changes which key is used to mask <code>win</code>  <code>Alt</code>  keyup events.
<code>#Include <Lib></code>	Includes a script file from a function library folder.
<code>#InputLevel</code>	Controls which artificial keyboard and mouse events are ignored by hotkeys and hotstrings.
<code>#Warn</code>	Enables or disables warnings for selected conditions that may be indicative of developer errors.

Functions

ComObj... -- ComObjActive ComObjEnwrap/Unwrap ComObjParameter ComObjType	Retrieves a registered COM object. Wraps/unwraps a COM object. Wraps a value and type to pass as a parameter. Retrieves a COM object's type information.
ComObjArray	Creates a SAFEARRAY for use with COM.
ComObjConnect	Connects a COM object's event sources to functions with a given prefix.
ComObjCreate	Creates a COM object.
ComObjError	Enables or disables notification of COM errors.
ComObjFlags	Retrieves or changes flags which control a COM wrapper object's behaviour.
ComObjGet	Returns a reference to an object provided by a COM component.
ComObjQuery	Queries a COM object for an interface or service.
ComObjType	Retrieves type information from a COM object.
ComObjValue	Retrieves the value or pointer stored in a COM wrapper object.
Exception	Creates an exception object for Throw (also provides limited access to the call stack).
FileOpen	Provides object-oriented file I/O.
Func	Retrieves a reference to a function.
GetKeyName/VK/SC	Retrieves the name or text, virtual key code or scan code of a key.
InStr	Searches for a given <i>occurrence</i> of a string, from the left or the right.

IsByRef	Determines whether a ByRef parameter was supplied with a variable.
IsObject	Determines whether a value is an object.
StrPut / StrGet	Copies a string to or from a memory address, optional converting it between code pages.
Trim	Trims certain characters from the beginning and/or end of a string.
RegEx (?CNum:Func)	Calls a function during evaluation of a regex pattern.
Function Libraries	New "local library" and <code>#Include <LibName></code> .
Variadic Functions	Functions may accept a variable number of parameters via an array.
Static Initializers	Static variables can now be initialized using any expression.

Objects

General	Behaviour and usage of objects in general.
Object	Associative arrays which can be extended with other functionality.
Enumerator	Allows items in a collection to be enumerated.
File	Provides an interface to access a file. FileOpen returns an object of this type.
Func	Represents a user-defined or built-in function which can be called by the script.
ComObject	See ComObj functions above.

Variables

A_Is64bitOS	Contains 1 (true) if the OS is 64-bit or 0 (false) if it is 32-bit.
A_IsUnicode	In Unicode builds, this variable contains 1 (<i>true</i>). In ANSI builds it is not defined, so is effectively <i>false</i> .
A_FileEncoding	Contains the default encoding for various commands; see FileEncoding .
A_OSVersion	Supports Windows 7 and Windows 8; see A_OSVersion .
A_PriorKey	The name of the last key which was pressed prior to the most recent key-press or key-release ... (More)
A_PtrSize	Contains the size of a pointer, in bytes. This is either 4 (32-bit) or 8 (64-bit).
A_RegView	The current registry view as set by SetRegView .
A_ScriptHwnd	The unique ID (HWND/handle) of the script's hidden main window.

Datatypes

Ptr	Equivalent to <i>Int</i> in 32-bit builds and <i>Int64</i> in 64-bit builds. Supported by DllCall , NumPut and NumGet .
AStr , WStr	Supported only by DllCall ; see Script Compatibility .

Unicode

Compatibility	How to deal with Unicode in DllCall, etc.
Script Files	Using Unicode in script files.
SendInput	Using Unicode with SendInput.

Other

ahk_exe	Windows can be identified by the name or path of the process (EXE file) which owns them.
Debugging	Interactive debugging features (line by line execution etc.).
Error Handling	Try/catch/throw and increased usefulness for A_LastError.
GUI Enhancements	Various enhancements to the Gui command and related.
Icon Support	Resource identifiers and improved support for various icon sizes.
Other Changes	Changes affecting script compatibility.
Version History	History of AutoHotkey_L revisions.

Error Handling

Many commands support using [try/catch](#) instead of `ErrorLevel` for error handling. For example:

```
try
{
    FileCopy, file1.txt, C:\folder
    FileDelete, C:\folder\old.txt
}
catch
    MsgBox An error occurred!
```

Additionally, the following commands now set [A_LastError](#) to assist with debugging: `FileAppend`, `FileRead`, `FileReadLine`, `FileDelete`, `FileCopy`, `FileMove`, `FileGetAttrib/Time/Size/Version`, `FileSetAttrib/Time`, `FileCreateDir`, `RegRead`, `RegWrite`, `RegDelete`.

Function Libraries

In addition to the user library in `%A_MyDocuments%\AutoHotkey\Lib` and standard library in the AutoHotkey directory, functions may be auto-included from the "local library" which resides in `%A_ScriptDir%\Lib`. For more information, see [Libraries of Functions](#).

`#Include <LibName>` explicitly includes a library file which may be located in any one of the function libraries.

GUI Enhancements

A number of enhancements have been made to the `Gui` command and related:

- A `name` or `HWND` can be used instead of a number between 1 and 99 when referring to a GUI.
- `Gui, New` creates a new anonymous GUI.
- Any number of named or anonymous GUIs can be created.
- New GUI options: `+HwndOutputVar`, `+ParentGUI`
- A GUI's owner can be any arbitrary window: `+Owner%HWND%`.
- `Gui, Font` can control anti-aliasing of text.
- `ActiveX controls` such as the Internet Explorer WebBrowser control are supported.
- `GuiControlGet`, `OutputVar`, `Name` gets the name of the variable associated with a GUI control.
- Keyboard accelerators such as `Ctrl`  supported automatically when used in `Gui menus`.
- `Font quality` can be controlled by the `Font` sub-command.

Static Variables

Static variables can now be initialized using any expression. For example:

```
Sleep 500
MsgBox % Time() "ms since the script started."
Time() {
    static Tick := A_TickCount
    return A_TickCount - Tick
}
```

Text Encodings

[FileRead](#), [FileReadLine](#), [Loop Read](#) and [FileAppend](#) support the majority of Windows-supported text encodings, not just the system default ANSI code page. [FileEncoding](#) can be used to set the default encoding, which can be overridden for [FileRead](#) and [FileAppend](#) as follows:

```
FileRead, OutputVar, *Pnnn Filename  
FileAppend [, Text, Filename, Encoding]
```

While *nnn* must be a numeric [code page identifier](#), *Encoding* follows the same format as [FileEncoding](#).

See also: [Script Compatibility](#)

Variadic Functions and Function-Calls

Variadic functions can receive a variable number of parameters via an array, while **variadic function-calls** can be used to pass a variable number of parameters to a function.

Improvements to Icon Support

Unusual Sizes

Icon resources of any size supported by the operating system may be extracted from executable files. When multiple sized icon resources exist within an icon group, the most appropriate size is used. Prior to revision 17, an arbitrary icon resource was selected by the system, scaled to the system large icon size, then scaled back to the requested size.

Resource Identifiers

Negative icon numbers may be used to identify a group icon resource within an executable file. For example, the following sets the tray icon to the default icon used by ahk files:

```
Menu, Tray, Icon, %A_AhkPath%, -160
```

List of Windows Messages

Below is a list of values for the *Msg* parameter of [PostMessage](#) and [SendMessage](#). To discover more about how to use a particular message (e.g. `WM_VSCROLL`), look it up at <http://msdn.microsoft.com> or with a search engine of your choice. Also, check out the [Message Tutorial](#).

```
WM_NULL = 0x00
WM_CREATE = 0x01
WM_DESTROY = 0x02
WM_MOVE = 0x03
WM_SIZE = 0x05
WM_ACTIVATE = 0x06
WM_SETFOCUS = 0x07
WM_KILLFOCUS = 0x08
WM_ENABLE = 0x0A
WM_SETREDRAW = 0x0B
WM_SETTEXT = 0x0C
WM_GETTEXT = 0x0D
WM_GETTEXTLENGTH = 0x0E
WM_PAINT = 0x0F
WM_CLOSE = 0x10
WM_QUERYENDSESSION = 0x11
WM_QUIT = 0x12
WM_QUERYOPEN = 0x13
WM_ERASEBKGD = 0x14
WM_SYSCOLORCHANGE = 0x15
WM_ENDSESSION = 0x16
WM_SYSTEMERROR = 0x17
WM_SHOWWINDOW = 0x18
WM_CTLCOLOR = 0x19
WM_WININICHANGE = 0x1A
WM_SETTINGCHANGE = 0x1A
WM_DEVMODECHANGE = 0x1B
WM_ACTIVATEAPP = 0x1C
WM_FONTCHANGE = 0x1D
```

```
WM_TIMECHANGE = 0x1E
WM_CANCELMODE = 0x1F
WM_SETCURSOR = 0x20
WM_MOUSEACTIVATE = 0x21
WM_CHILDACTIVATE = 0x22
WM_QUEUESYNC = 0x23
WM_GETMINMAXINFO = 0x24
WM_PAINTICON = 0x26
WM_ICONERASEBKGND = 0x27
WM_NEXTDLGCTL = 0x28
WM_SPOOLERSTATUS = 0x2A
WM_DRAWITEM = 0x2B
WM_MEASUREITEM = 0x2C
WM_DELETEITEM = 0x2D
WM_VKEYTOITEM = 0x2E
WM_CHARTOITEM = 0x2F

WM_SETFONT = 0x30
WM_GETFONT = 0x31
WM_SETHOTKEY = 0x32
WM_GETHOTKEY = 0x33
WM_QUERYDRAGICON = 0x37
WM_COMPAREITEM = 0x39
WM_COMPACTING = 0x41
WM_WINDOWPOSCHANGING = 0x46
WM_WINDOWPOSCHANGED = 0x47
WM_POWER = 0x48
WM_COPYDATA = 0x4A
WM_CANCELJOURNAL = 0x4B
WM_NOTIFY = 0x4E
WM_INPUTLANGCHANGEREQUEST = 0x50
WM_INPUTLANGCHANGE = 0x51
WM_TCARD = 0x52
WM_HELP = 0x53
WM_USERCHANGED = 0x54
WM_NOTIFYFORMAT = 0x55
WM_CONTEXTMENU = 0x7B
```

WM_STYLECHANGING = 0x7C
WM_STYLECHANGED = 0x7D
WM_DISPLAYCHANGE = 0x7E
WM_GETICON = 0x7F
WM_SETICON = 0x80

WM_NCCREATE = 0x81
WM_NCDESTROY = 0x82
WM_NCCALCSIZE = 0x83
WM_NCHITTEST = 0x84
WM_NCPAINT = 0x85
WM_NCACTIVATE = 0x86
WM_GETDLGCODE = 0x87
WM_NCMOUSEMOVE = 0xA0
WM_NCLBUTTONDOWN = 0xA1
WM_NCLBUTTONUP = 0xA2
WM_NCLBUTTONDBLCLK = 0xA3
WM_NCRBUTTONDOWN = 0xA4
WM_NCRBUTTONUP = 0xA5
WM_NCRBUTTONDBLCLK = 0xA6
WM_NCMBUTTONDOWN = 0xA7
WM_NCMBUTTONUP = 0xA8
WM_NCMBUTTONDBLCLK = 0xA9

WM_KEYFIRST = 0x100
WM_KEYDOWN = 0x100
WM_KEYUP = 0x101
WM_CHAR = 0x102
WM_DEADCHAR = 0x103
WM_SYSKEYDOWN = 0x104
WM_SYSKEYUP = 0x105
WM_SYSCHAR = 0x106
WM_SYSDEADCHAR = 0x107
WM_KEYLAST = 0x108

WM_IME_STARTCOMPOSITION = 0x10D
WM_IME_ENDCOMPOSITION = 0x10E

WM_IME_COMPOSITION = 0x10F
WM_IME_KEYLAST = 0x10F

WM_INITDIALOG = 0x110
WM_COMMAND = 0x111
WM_SYSCOMMAND = 0x112
WM_TIMER = 0x113
WM_HSCROLL = 0x114
WM_VSCROLL = 0x115
WM_INITMENU = 0x116
WM_INITMENUPOPUP = 0x117
WM_MENUSELECT = 0x11F
WM_MENUCHAR = 0x120
WM_ENTERIDLE = 0x121

WM_CTLCOLORMSGBOX = 0x132
WM_CTLCOLOREDIT = 0x133
WM_CTLCOLORLISTBOX = 0x134
WM_CTLCOLORBTN = 0x135
WM_CTLCOLORDLG = 0x136
WM_CTLCOLORSCROLLBAR = 0x137
WM_CTLCOLORSTATIC = 0x138

WM_MOUSEFIRST = 0x200
WM_MOUSEMOVE = 0x200
WM_LBUTTONDOWN = 0x201
WM_LBUTTONUP = 0x202
WM_LBUTTONDBLCLK = 0x203
WM_RBUTTONDOWN = 0x204
WM_RBUTTONUP = 0x205
WM_RBUTTONDBLCLK = 0x206
WM_MBUTTONDOWN = 0x207
WM_MBUTTONUP = 0x208
WM_MBUTTONDBLCLK = 0x209
WM_MOUSEWHEEL = 0x20A
WM_MOUSEHWHEEL = 0x20E

WM_PARENTNOTIFY = 0x210
WM_ENTERMENULOOP = 0x211
WM_EXITMENULOOP = 0x212
WM_NEXTMENU = 0x213
WM_SIZING = 0x214
WM_CAPTURECHANGED = 0x215
WM_MOVING = 0x216
WM_POWERBROADCAST = 0x218
WM_DEVICECHANGE = 0x219

WM_MDICREATE = 0x220
WM_MDIDESTROY = 0x221
WM_MDIACTIVATE = 0x222
WM_MDIRESTORE = 0x223
WM_MDINEXT = 0x224
WM_MDIMAXIMIZE = 0x225
WM_MDITILE = 0x226
WM_MDICASCADE = 0x227
WM_MDIICONARRANGE = 0x228
WM_MDIGETACTIVE = 0x229
WM_MDISETMENU = 0x230
WM_ENTERSIZEMOVE = 0x231
WM_EXITSIZEMOVE = 0x232
WM_DROPFILES = 0x233
WM_MDIREFRESHMENU = 0x234

WM_IME_SETCONTEXT = 0x281
WM_IME_NOTIFY = 0x282
WM_IME_CONTROL = 0x283
WM_IME_COMPOSITIONFULL = 0x284
WM_IME_SELECT = 0x285
WM_IME_CHAR = 0x286
WM_IME_KEYDOWN = 0x290
WM_IME_KEYUP = 0x291

WM_MOUSEHOVER = 0x2A1
WM_NCMOUSELEAVE = 0x2A2

WM_MOUSELEAVE = 0x2A3

WM_CUT = 0x300

WM_COPY = 0x301

WM_PASTE = 0x302

WM_CLEAR = 0x303

WM_UNDO = 0x304

WM_RENDERFORMAT = 0x305

WM_RENDERALLFORMATS = 0x306

WM_DESTROYCLIPBOARD = 0x307

WM_DRAWCLIPBOARD = 0x308

WM_PAINTCLIPBOARD = 0x309

WM_VSCROLLCLIPBOARD = 0x30A

WM_SIZECLIPBOARD = 0x30B

WM_ASKCBFORMATNAME = 0x30C

WM_CHANGECHAIN = 0x30D

WM_HSCROLLCLIPBOARD = 0x30E

WM_QUERYNEWPALETTE = 0x30F

WM_PALETTEISCHANGING = 0x310

WM_PALETTECHANGED = 0x311

WM_HOTKEY = 0x312

WM_PRINT = 0x317

WM_PRINTCLIENT = 0x318

WM_HANDHELDFIRST = 0x358

WM_HANDHELDLAST = 0x35F

WM_PENWINFIRST = 0x380

WM_PENWINLAST = 0x38F

WM_COALESCE_FIRST = 0x390

WM_COALESCE_LAST = 0x39F

WM_DDE_FIRST = 0x3E0

WM_DDE_INITIATE = 0x3E0

WM_DDE_TERMINATE = 0x3E1

WM_DDE_ADVISE = 0x3E2

WM_DDE_UNADVISE = 0x3E3

```
WM_DDE_ACK = 0x3E4  
WM_DDE_DATA = 0x3E5  
WM_DDE_REQUEST = 0x3E6  
WM_DDE_POKE = 0x3E7  
WM_DDE_EXECUTE = 0x3E8  
WM_DDE_LAST = 0x3E8
```

```
WM_USER = 0x400  
WM_APP = 0x8000
```

CLSID List (Windows Class Identifiers)

Certain special folders within the operating system are identified by unique strings. Some of these strings can be used with [FileSelectFile](#), [FileSelectFolder](#), and [Run](#). For example:

```
FileSelectFile, OutputVar,, ::{645ff040-5081-101b-9f08-00aa002f954e} ; Select a file in the Recycle Bin.  
FileSelectFolder, OutputVar, ::{20d04fe0-3aea-1069-a2d8-08002b30309d} ; Select a folder within My Computer.
```

CLSID	Location	Run?
::{d20ea4e1-3957-11d2-a40b-0c5020524153}	Administrative Tools	
::{85bbd920-42a0-1069-a2e4-08002b30309d}	Briefcase	
::{21ec2020-3aea-1069-a2dd-08002b30309d}	Control Panel	
::{d20ea4e1-3957-11d2-a40b-0c5020524152}	Fonts	
::{ff393560-c2a7-11cf-bff4-444553540000}	History	
::{00020d75-0000-0000-c000-000000000046}	Inbox	
::{00028b00-0000-0000-c000-000000000046}	Microsoft Network	
::{20d04fe0-3aea-1069-a2d8-08002b30309d}	My Computer	Yes
::{450d8fba-ad25-11d0-98a8-	My Documents	Yes

0800361b1103}		
::{208d2c60-3aea-1069-a2d7-08002b30309d}	My Network Places	Yes
::{1f4de370-d627-11d1-ba4f-00a0c91eedba}	Network Computers	Yes
::{7007acc7-3202-11d1-aad2-00805fc1270e}	Network Connections	Yes
::{2227a280-3aea-1069-a2de-08002b30309d}	Printers and Faxes	Yes
::{7be9d83c-a729-4d97-b5a7-1b7313c39e0a}	Programs Folder	
::{645ff040-5081-101b-9f08-00aa002f954e}	Recycle Bin	Yes
::{e211b736-43fd-11d1-9efb-0000f8757fcd}	Scanners and Cameras	
::{d6277990-4c6a-11cf-8d87-00aa0060f5bf}	Scheduled Tasks	Yes
::{48e7caab-b918-4e58-a94d-505519c795dc}	Start Menu Folder	
::{7bd29e00-76c1-11cf-9dd0-00a0c9034933}	Temporary Internet Files	
::{bdeadf00-c265-11d0-bced-00a0c90ab50f}	Web Folders	

The "Yes" entries in the last column are not authoritative: the [Run](#) command might support different CLSIDs depending on system configuration. To open a CLSID folder via Run, simply specify the CLSID as the first parameter. For example:

```
Run ::{20d04fe0-3aea-1069-a2d8-08002b30309d} ;
Opens the "My Computer" folder.
Run ::{645ff040-5081-101b-9f08-00aa002f954e} ;
```

Opens the Recycle Bin.

Run ::{450d8fba-ad25-11d0-98a8-0800361b1103}\My Folder ; Opens a folder that exists inside "My Documents".

Run %A_MyDocuments%\My Folder ; Same as the above on most systems.

Send Messages to a Window or Its Controls

by Rajat

This page discusses the [PostMessage](#) and [SendMessage](#) commands and will answer some questions like:

"How do I press a button on a minimized window?"

"How do I select a menu item when [WinMenuItem](#) doesn't work with it?!"

"This is a skinnable window.... how do I send a command that works every time?"

"and what about **hidden** windows?!"

Requirements: [AutoHotkey v1.0.09+] and Winspector Spy ([can be found here](#))

As a first example, note that [WinMenuItem](#) fails to work with the menu bar on Outlook Express's "New Message" window. In other words, this code will not work:

```
WinMenuItem, New Message,, &Insert,  
&Picture...
```

But [PostMessage](#) can get the job done:

```
PostMessage, 0x111, 40239, 0, , New Message
```

Works like a charm! But what the heck is that? 0x111 is the hex code of [wm_command message](#) and 40239 is the code that this particular window understands as menu-item 'Insert Picture' selection. Now let me tell you how to find a value such as 40239:

1. Open Winspector Spy and a "New Message" window.
2. Drag the crosshair from Winspector Spy's window to "New Message" window's titlebar (the portion not covered by Winspector Spy's overlay).
3. Right click the selected window in the list on left and select 'Messages'.
4. Right click the blank window and select 'Edit message filter'.
5. Press the 'filter all' button and then dbl click 'wm_command' on the list on left. This way you will only monitor this message.
6. Now go to the "New Message" window and select from its menu bar: Insert > Picture.
7. Come back to Winspector Spy and press the traffic light button to pause monitoring.
8. Expand the wm_command messages that've accumulated (forget others if any).
9. What you want to look for (usually) is a code 0 message. sometimes there are wm_command messages saying 'win activated' or 'win destroyed' and other stuff.. not needed. You'll find that there's a message saying 'Control ID: 40239' ...that's it!
10. Now put that in the above command and you've got it! It's the wParam value.

For the next example I'm taking Paint because possibly everyone will have that. Now let's say it's an app where you have to select a tool from a toolbar using AutoHotkey; say the dropper tool is to be selected.

What will you do? Most probably a mouse click at the toolbar button, right? But toolbars can be moved and hidden! This one can be moved/hidden too. So if the target user has done any of this then your script will fail at that point. But the following command will still work:

```
PostMessage, 0x111, 639,,, Untitled - Paint
```

Another advantage to [PostMessage](#) is that the Window can be in the background; by contrast, sending mouse clicks would require it to be active.

Here are some more examples. Note: I'm using WinXP Pro (SP1) ... if you use a different OS then your params may change (only applicable to apps like Wordpad and Notepad that come with windows; for others the params shouldn't vary):

```
;makes writing color teal in Wordpad
PostMessage, 0x111, 32788, 0, , Document -
WordPad
```

```
;opens about box in Notepad
PostMessage, 0x111, 65, 0, , Untitled - Notepad
```

```
;toggles word-wrap in Notepad
PostMessage, 0x111, 32, 0, , Untitled - Notepad
```

```
;play/pause in Windows Media Player
PostMessage, 0x111, 32808, 0, , Windows Media
Player
```

```
;suspend the hotkeys of a running AHK script
DetectHiddenWindows, On
SetTitleMatchMode, 2
PostMessage, 0x111, 65305,,, MyScript.ahk -
AutoHotkey ; Use 65306 to Pause vs. Suspend.
```

This above was for PostMessage. [SendMessage](#) works the same way but additionally waits for a return value, which can be used for things such as getting the currently playing track in Winamp (see [Automating Winamp](#) for an example).

Here are some more notes:

- The note above regarding OS being XP and msg values changing with different OSes is purely cautionary. if you've found a msg that works on your system (with a certain version of a software) then you can be sure it'll work on another system too with the same version of the software. In addition, most apps keep these msg values the same even on different versions of themselves (e.g. Windows Media Player and Winamp).
- If you've set the filter to show only wm_command in Winspector Spy and

still you're getting a flood of messages then right click that message and select hide (msg name)... you don't want to have a look at a msg that appears without you interacting with the target software.

- The right pointing arrow in Winspector Spy shows the msg values and the blurred left pointing arrows show the returned value. A 0 (zero) value can by default safely be taken as 'no error' (use it with SendMessage, the return value will be in `%ErrorLevel%`).
- For posting to partial title match add this to script:
`SetTitleMatchMode, 2`
- For posting to hidden windows add this to script:
`DetectHiddenWindows, On`

Note: There are apps with which this technique doesn't work. I've had mixed luck with VB and Delphi apps. This technique is best used with C, C++ apps. With VB apps the 'LParam' of the same command keeps changing from one run to another. With Delphi apps... the GUI of some apps doesn't even use `wm_command`. It probably uses mouse pos & clicks.

Go and explore.... and share your experiences in the AutoHotkey Forum. Feedback is welcome!

This tutorial is not meant for total newbies (no offense meant) since these commands are considered advanced features. So if after reading the above you've not made heads or tails of it, please forget it.

-Rajat