

Click to verify



[illegible]

v1.1.34.03 or later installer gives the option. Alternatively, use the /install switch described below. Each version install to its own subdirectory. Running a v1.1.34.02 or older installer (or a custom install with v1.1.34.03 or newer) will overwrite some of the values set in the registry by the v2 installer, such as the version number, installer entry and parts of the file type registration. It will also register the v1 uninstaller, which is not capable of correctly uninstalling both versions. To re-register v2, re-run any v2 installer or run UXinstall.ahk using AutoHotkey32.exe or AutoHotkey64.exe. Default Version Unlike a v1 installation, a default version is not selected during installation. Defaults are instead handled more dynamically by the launcher, and can be configured per-user. Command Line Usage To directly install to the DESTINATION directory, use /installto or /to (the two switches are interchangeable) as shown below, from within the source directory. Use either a downloaded setup.exe or files extracted from a downloaded zip or other source. AutoHotkey_setup.exe /installto "%DESTINATION%" AutoHotkey32.exe UXinstall.ahk /to "%DESTINATION%" To install an additional version from SOURCE (which should be a directory containing AutoHotkey*.exe files), execute the following from within the current installation directory (adjusting the path of AutoHotkey32.exe as needed): AutoHotkey32.exe UXinstall.ahk /install "%SOURCE%" The full command string for the above is registered as InstallCommand under HKLM\Software\AutoHotkey or HKCU\Software\AutoHotkey, with %1 as the substitute for the source directory. Using this registry value may be more future-proof. To re-register the current installation: AutoHotkey32.exe UXinstall.ahk To uninstall: AutoHotkey32.exe UXinstall.ahk /uninstall Alternatively, read the QuietUninstallString value from one of the following registry keys, and execute it: HKLM\Microsoft\Windows\CurrentVersion\Uninstall\AutoHotkey HKCU\Microsoft\Windows\CurrentVersion\Uninstall\AutoHotkey Use the /silent switch to suppress warning or confirmation dialogs and prevent the Dash from being shown when installation is complete. The following actions may be taken automatically, without warning: Terminate scripts to allow AutoHotkey*.exe to be overwritten. Overwrite files that were not previously registered by the installer, or that were modified since registration. Taskbar Buttons The v2 installer does not provide an option to separate taskbar buttons. This was previously achieved by registering each AutoHotkey executable as a host app (IsHostApp), but this approach has limitations, and becomes less manageable when multiple versions can be installed. Instead, each script should set the AppUserModelID of its process or windows to control grouping. Run with UI Access When installing under Program Files, the installer creates an additional set of AutoHotkey exe files that are used to work around some common UAC-related issues. These files are given the " UIA.exe" suffix. When one of these UIA.exe files is used by an administrator to run a script, the script is able to interact with windows of programs that run as admin, without the script itself running as admin. The installer does the following: Copies each AutoHotkey*.exe to AutoHotkey* UIA.exe. Sets the uiAccess attribute in each UIA.exe file's embedded manifest. Creates a self-signed digital certificate named "AutoHotkey" and signs each UIA.exe file. Registers the UIAccess shell verb, which appears in Explorer's context menu as "Run with UI access". By default this executes the launcher, which tries to select an appropriate UIA.exe file to run the script. The launcher can also be configured to run v1 scripts, v2 scripts or both with UI Access by default, but this option has no effect if a UIA.exe file does not exist for the selected version and build. Scripts which need to run other scripts with UI access can simply Run the appropriate UIA.exe file with the normal command line parameters. Alternatively, if the UIAccess shell verb is registered, it can be used via Run. For example: Run "UIAccess "Script.ahk"" 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. UIA prevents the Gui +Parent option from working on an existing window if the new parent is always-on-top and the child window is not. For more details, see Enable interaction with administrative programs on the archived forum. 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 B behave like B: a::b The above example does not alter B itself. B would continue to send the "b" keystroke unless you remap it to something else as shown in the following example: a::b b::a The examples above use lowercase, which is recommended for most purposes because it also remaps the corresponding uppercase letters (that is, it will send uppercase when CapsLock is "on" or Shift is held down). By contrast, specifying an uppercase letter on the right side forces uppercase. For example, the following line would produce an uppercase B when you type either "a" or "A" (as long as CapsLock is off): a::B Conversely, any modifiers included on the left side but not the right side are automatically released when the key is sent. For example, the following two lines would produce a lowercase "b" when you press either Shift+A or Ctrl+A: A::b ^a::b Mouse Remapping To remap the mouse instead of the keyboard, use the same approach. For example: Other Useful Remappings 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. The #HotIf directive can be used to make selected remappings active only in the windows you specify (or while any given condition is met). For example: #HotIf WinActive("ahk_class Notepad") a::b ; Makes the 'a' key send a 'b' key, but only in Notepad. #HotIf ; This puts subsequent remappings and hotkeys in effect for all windows. Remapping a key or button is "complete" in the following respects: Holding down a modifier such as Ctrl or Shift while typing the origin key will put that modifier into effect for the destination key. For example, b::a would produce Ctrl+A if you press Ctrl+B. CapsLock generally affects remapped keys in the same way as normal keys. The destination key or button is held down for as long as you continue to hold down the origin key. However, some games do not support remapping; in such cases, the keyboard and mouse will behave as though not remapped. Remapped keys will auto-repeat while being held down (except keys remapped to become mouse buttons). 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 to perform the same action as Ctrl+Alt+B. For example: a::b ^!a:: ^!b::ToolTip "You pressed " ThisHotkey Alternatively, #InputLevel can be used to override the default behaviour. For example: #InputLevel 1 a::b #InputLevel 0 ^!b::ToolTip "You pressed " ThisHotkey If SendMode is used during script startup, 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 Ctrl, Shift, Alt, and Win). To work around this, avoid using SendMode "Play" during script startup when you have remappings; then use the function 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. If DestinationKey is meant to be {, it has to be escaped, for example, x:: { . Otherwise it is interpreted as the opening brace for the hotkey's function. 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 functions in the script won't assume "b" should stay down during their Send. } ^a up:: { SetKeyDelay -1 ; See note below for why press-duration is not specified with either of these SetKeyDelays. Send "{Blind}{b Up}" } However, the above hotkeys vary under the following circumstances: When the source key is the left Ctrl and the destination key is Alt, 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 Ctrl, except that {RCtrl up} is used. This is done to ensure that the system translates Alt-key combinations as though Ctrl is not being held down, but it also causes the remapping to override any prior {Ctrl down}. {v2.0.8+}: The unsuppressed Ctrl key-up is still sent for backward-compatibility, but is no longer needed for its original purpose. The side effect can be avoided by replacing the remapping with an explicit pair of hotkeys as demonstrated above. 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}" } When the source is a custom combination, the wildcard modifier (*) is omitted to allow the hotkeys to work. If any of the modifier symbols in !#^+ are applied to the source key and not the destination key, they are inserted after the word "Blind" to allow those modifiers to be released by Send. For example, ^a::b would use {Blind^}. MsgBox("Hello!"), -1000 ; Says hello after 1 second. The fat arrow syntax can also be used outside of expressions as shorthand for a normal function or method definition. For example, the following is equivalent to the SayHello definition above, except that this one returns "OK": SayHello() => MsgBox("Hello!") Variables in functions are local to that function by default, except in the following cases: When the function is assume-global. When a variable is referenced but not used as the target of an assignment or the reference operator (&var). When referring to a local variable of an outer function inside a nested function. A function can optionally accept parameters. Parameters are defined by listing them inside the parentheses. For example: MyFunction(FirstParameter, Second, &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. The ByRef marker (&) indicates that the caller must pass a variable reference. Inside the function, any reference to the parameter will actually access the caller's variable. This is similar to omitting & and explicitly dereferencing the parameter inside the function (e.g. %Third%), but in this case the percent signs are omitted. If the parameter is optional and the caller omits it, the parameter acts as a normal local variable. Optional parameters are specified by following the parameter name with := and a default value, which must be a literal quoted string, a number, true, false or unset. The function can return a value. If it does not, the default return value is an empty string. A function definition does not need to precede calls to that function. 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. An #Include file can contain global code to be executed during script startup, but as with code in the main script file, such code will be executed only if the auto-execute thread is not terminated (such as with an unconditional Return) prior to the #Include directive. A warning is displayed by default if any code cannot be executed due to a prior Return. 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. To facilitate sharing scripts, #Include can search a few standard locations for a library script. For details, see Script Library Folders. Miscellaneous Dynamic Variables A dynamic variable reference takes a text value and interprets it as the name of a variable. Note: A variable cannot be created by a dynamic reference, but existing variables can be assigned. This includes all variables which the script contains non-dynamic references to, even if they have not been assigned values. 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 => target MsgBox %second% ; Double-deref => 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% %Y%. This is used to access pseudo-arrays, described below. These techniques can also be applied to properties and methods of objects. For example: clr := {} for n, component in ["red", "green", "blue"] clr.component% := Random(0, 255) MsgBox clr.red "," clr.green "," clr.blue 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. The "index" used to form the final variable name does not have to be numeric; it could instead be a letter or keyword. For these reasons, it is generally recommended to use an Array or Map instead of a pseudo-array: 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 is only conceptual and not a single value, it can't be passed to or returned from a function, or copied as a whole. A pseudo-array cannot be declared as a whole, so some "elements" may resolve to global (or captured) variables while others do not. If a variable is referenced non-dynamically but only assigned dynamically, a load-time warning may be displayed. Such warnings are a very effective way to detect errors, so disabling them is not recommended. Current versions of the language do not permit creating new variables dynamically. This is partly to encourage best practices, and partly to avoid inconsistency between dynamic and non-dynamic variable references in functions. Labels A label identifies a line of code, and can be used as a Goto target or to specify a loop to break out of or continue. A label consist of a name followed by a colon: this is a label. Aside from whitespace and comments, no other code can be written on the same line as a label. For more details, see Labels. AutoHotkey v2 aims to improve the usability and convenience of the language and command set by sacrificing backward compatibility. As such, scripts written for v1 generally will not work without changes on v2. The syntax is overall more consistent, with much fewer quirks and traps, and many other improvements have been made. For extensive details about the differences between v1 and v2, refer to Changes from v1.1 to v2.0. v2.0.0 was released on 2022-12-20. Requirements: Windows 7 and later are supported. Installation v2.0-beta.4 and later can be installed together with v1.1., allowing scripts written for either version to work with minimal fuss. For details, see Installation and Launcher in the documentation. To install using the exe, just download and run it, review the options and click Install. To install using the zip: Open the zip file using File Explorer (no extraction necessary), or extract the contents of the zip file to a temporary directory. Run Install.cmd. If you receive a prompt like "The publisher could not be verified. Are you sure...?", click Run. Click Install (after changing options, if you wish). After installation, the Dash opens automatically. Downloads Latest Installer (or latest zip) Note: Google Safe Browsing sometimes falsely flags the following directory as containing "harmful programs". For more information, see Safe Browsing. All v2.0 Releases Source code is available at GitHub. Get Involved Discuss AutoHotkey v2 Commit History on GitHub Contribute via GitHub Documentation Online Documentation Contribute via GitHub Download via GitHub Software License: GNU General Public License.

- <http://penzion-u-zamku.cz/files/file/692216ac-853b-40e0-8e7c-f1c5bed7fe0d.pdf>
- <http://ke-sen.com/userfiles/file/1752043013.pdf>
- easy books to read in portuguese
- todi
- macecho