ActiveWords Scripts and Syntax

NOTE: Scripting is for advanced users and enterprise administrators. Scripting can be used to automate multi-step processes, such as directly delivering the result of an Outlook Inbox search (enter search text in input dialog, open Outlook, go to Inbox, place focus in Search field, input search text). Product support does not include resolving issues with scripts created by users. Several script examples can be found here. Scripting is not a replacement for advanced scripting tools such as Visual Basic, AutoIT, or AutoHotKey. Words can be used to trigger script files created with other scripting tools for more advanced automation.

Syntax Notes:

- Create scripts using the “Script” Type from the New ActiveWord tool.
- Commands are defined within braces {}, as in {control}.
- To repeat a command key include a space and the number of repeats, as in {backspace 5}.
- Use parentheses to hold down and release keys. For example, {control}({shift}{f1}) simulates pressing and holding the Control key, then pressing the Shift key, then pressing the F1 key, then releasing the Control key. Likewise, {alt}({numpad2}{numpad6}{numpad5}) simulates pressing and holding the Alt key, then pressing 2, 6, and 5 on then numeric keypad, then releasing the Alt key.
- To send a brace, enclose the brace in a pair of braces. So, to send an opening or closing brace, use {{} or {}}.
- Commands are not case sensitive.
- Comments are included in scripts by starting a line with a semi-colon.
- Regular characters are sent as Unicode so in most cases shift+character will NOT give you the upper case of the character. Shift is only useful in combination with other modifier keys.

Script commands to simulate non alpha-numeric keys:

Modifier Keys:

CONTROL or CTRL
SHIFT
ALT
WINDOWS or WIN

Modifier Key Syntax:

- {[modifier down]}[keys modified]{[modifier up]} – e.g. – {control down}{k}{control up}

Use this syntax where the modify key needs to be held down at the same time as the keys being modified. The keys being modified may be in braces {} or not depending on the program where the script is used. So, {control down}{k}{control up} may work and/or {control down}k{control up} may work.

- {[modifier]} – e.g. – {alt}jscf

Use this syntax where the modifier key needs to be pressed and released before entering the keys being modified, “{alt}jscf” simulates pressing and releasing the Alt key, then pressing j,s,c, and f one key at a time. This syntax applies to ribbon menus in Office programs, for example.

Other non alpha-numeric keys:

LEFTWIN or WINDOWS
RIGHTWIN
ENTER
TAB
ESC or ESCAPE
HOME
END
LEFT
RIGHT
UP
DOWN
PGUP
PGDWN
NUMLOCK
SCROLLLOCK
SPACE (or just use a space)
PRTSC

BREAK
BACKSPACE or BKSP or BS
CLEAR
CAPSLOCK
INSERT or INS
DELETE or DEL
HELP
F1 through F16
MULTIPLY (* key from main keyboard)
ADD (+ key from main keyboard)
SUBTRACT (- key from main keyboard)
DIVIDE (/ key from main keyboard)
* (key from numeric keypad)
+ (key from numeric keypad)
– (key from numeric keypad)
/ (key from numeric keypad)
NUMPAD# where # is the number on the numeric keypad.

Back to Top

Script commands to trigger behavior.

INPUT – ${INPUT “[entry field text]”,”[dialog title text]”} – User text entry within a script sequence.

Notes on ${input}:

${input} can be used in any Action Type. For example, a URL Action:
http://www.bing.com/search?q=${input “Search the Internet:”,”Bing Intenet Search”}

You can embed ${input} commands inside other commands. For example, *{word ${input “Enter Word to trigger:”,”Enter Word to Trigger”}}*

Requires use of ASCII quotation marks (**no curly quotes**). No spaces on either side of the comma separating [entry filed text] and [dialog title text]. So, “[entry field text]”,”[dialog title text]” will work but “[entry field text]” , “[dialog title text]” will not work.

You can trigger text substitution Actions within input fields but not other Action Type. You can trigger text substitution Actions that contain input commands within input fields.

SHOWMESSAGE – ${showmessage “title”, “message”} – Use this to pop up a message for the user during a script operation.

BLOCKINPUT {blockinput [on/off]} – This command allows scripts to block user

keyboard and mouse input during script operation. If {blockinput on} is included without {blockinput off} keyboard and mouse input will stop being blocked upon completion of script execution.

CONTEXTMENU – {contextmenu} Windows mouse right click menu.

RUN – Run a program with or without parameters. {run “[program]”,”[parameters]”}. Example: {run “outlook.exe”,” /c ipm.note”}

CONTROL 2 – {control 2} -ActiveWord Command – Toggles the Text Monitor on and off.

EDITLASTACTION – {editlasstaction} – ActiveWord Command – Will edit the last triggered Action. This is useful to quickly edit or delete an Action or any associated Words without having to open the list view and filter to locate the Action.

TRIGGERLASTWORD {triggerlastword} – ActiveWord Command – Will trigger the Action associated with the last Word triggered. This is useful to repeat an Action. It can be used one time.

WAIT – {wait #####} – Default is a half second delay, 500 millisecond, i.e. {wait 1000} will delay script processing for one second.

WINSAVE and WINRESTORE {winsave} and {winrestore} – These commands allow you to store a window context within a script, then return to that context later in the script.

WORD – {word [word]} – Triggers the Action associated with [word], e.g., {word cnn} triggers the Action associated with the Word “cnn.” Note: If you have the same word associated with more than one action, one triggering with a single delimiter and to the other with a double space, the script will trigger the Action associated with the Word that triggers with a single delimiter.

SETKEYDELAY – {SetKeyDelay n} – Set the delay used between key strokes. This timing will remain for the duration of the script unless changed by another SETKEYDELAY command. This command can be useful to slow down scripted keyboard entry.

CLIPCOPY – {clipcopy} – Empties clipboard, sends Control-c, and waits for copied contents to be present in the clipboard. Uses both CLIPEMPTY and CLIPWAIT.

CLIPEMPTY – {clipempty} – empty clipboard. Use before {control}{c} then use {ClipWait}.

CLIPWAIT – {clipwait nnn} wait for clipboard to contain data.

CLIPPASTE – {clippaste} – Pastes clipboard contents then waits for paste to be complete. This is important in scripts that copy to the clipboard, paste the clipboard, then include additional commands.

ACTIONTEXT {actiontext} – Will operate on selected text when the ActiveWords Key is pressed and will act on text entered in the ActionPad. For example, the Open URL Action “http://www.google.com/search?q={actiontext}” will search Google for any selected text and for any text entered in the ActionPad that is not a Word. Note: It is best not to associate Words with Actions including {actiontext}. A prompt will appear allowing selection of a specific Action if more than one Action includes the {actiontext} command.

These commands work when a Browser window is the active context:

BROWSERBACK
BROWSERFORWARD
BROWSERREFRESH
BROWSERSTOP
BROWSERSEARCH ???
BROWSERFAVORITES
BROWSERHOME

These commands control media regardless of active context:

VOLUMEMUTE
VOLUMEDOWN
VOLUMEUP
MEDIANEXTTRACK
MEDIAPREVTRACK
MEDIASTOP
MEDIAPLAYPAUSE

Back to Top

Script commands to manage windows (WIN* Commands)

The Windows operating system tracks each open window, and when a window opens and closes. The following commands help manage windows being operated on by a script. NOTE: Script commands waiting for Universal App windows (for example, EDGE) must rely on the general WAIT command rather than WIN* commands as Universal Apps may not provide a reliable means to assure a window is ready for input.

WINWAIT – {WinWait title, class, timeout, process} – Waits for a specific window to exist, may not be in the foreground. This commands assures a previously opened

window exists before processing subsequent commands.

WINWAITIDLE – {WinWaitIdle title, class, timeout, process} – Waits for a specified window to be ready for user input (keyboard, mouse or touch) regardless of whether the specified windows has focus. Reliability is dependent on the application. Use WINWAITIDLE, for example, this could be used to wait for a user to activate the specified window before executing subsequent commands.

WINACTIVATE – {WinActivate title, class, timeout, process} – Waits for a specific window to exist and brings it to the foreground (makes it the active context). Makes the specified window the focus for keyboard input.

WINCLOSE – {WinClose title, class, timeout, process} – Closes the specified window regardless of whether that window is in focus.

WINWAITCLOSE – {WinWaitClose title, class, timeout, process} – Example: Script closes a window that requires a decision (save or not). WINWAITCLOSE would be used to pause the script until the user acts or the timeout occurs cancelling the script.

SETWINTIMEOUT – {SetWinTimeout 1000} – Sets the Timeout to apply to all subsequent WIN* commands having a Timeout modifier. If a timeout occurs the script will abort.

WIN* Commands, Information

WIN* Command Optional Specifiers

- *Title* = full text or substring of window title (quotes ok). It is best to use a string of title text that is unlikely to exist in any other open window. If it is possible multiple windows with the same title string are open in different programs, use the "process" variable to define the process required for the specified window.
- *Class* = window class (quotes ok).
- *Timeout* = The remainder of the script will be aborted if the timeout completes. The default timeout is 500 milliseconds. Value is in milliseconds, so 10000 is ten seconds. It is useful to allow for a long timeout period when the script might be opening an executable for the first time. The next command in the sequence will occur immediately after the Win* command is true, so the timeout should be sufficient for a specified window to open, for example, when a script opens a program or file.
- *Process* = Name of the Windows Process related to the windows being operated on, for example, winword.exe.

Advanced users can use a tool such as Spyxx to identify the above window parameters.

Any optional specifying including a space needs to be within quotes, for example, a document title such as {winwait “Document – Wordpad”}. This also applies to regex syntax such as: {winwait “.*wordpad”} for any window with wordpad at the end of its title.

The WIN* commands use Unicode Regex expressions. For example, if the window title is “Untitled – Notepad” then “Un.*Notepad” will match it (any string beginning with Un and ending in Notepad). The best approach for the average user is to include as much of the Window Title Text as possible to generalize the script. Click here for more information.

If Title, Class, and Process are not specified the last window found from the preceding commands will be used.

Win* commands remember the last window found … so there is no need to re-specify the Title or Class when a prior command already specifies the window to operate on.

Specifying Title, Class, or Process returns the top most window with a match. Generally, once a window Title, Class, or Process has been specified in a script, the same Title, Class, or Process should not be specified for subsequent WIN* commands that operate on the same window.

The syntax requires a space after the WIN* command and a comma between each variable being used. For example,

{winwait ,,5000}
{winwait ,,,winword.exe}
{winwait “Document – Wordpad”}

{winwait [title],[class],[timeout],[process]}
{winwait letterhead,NetUIHWND,5000,winword.exe}

{winwait letterhead,,5000,winword.exe} – wait up to 5 seconds for a window containing text “letterhead” in the title and with the process “winword.exe” to exist. You might use this command after a command that opens the document “letterhead” to wait for it to be opened. If the specified window is found within the timeout period the next command in the script will execute, otherwise the script will stop processing at that point.

Back to Top