NAME

xdotool − command−line X11 automation tool

SYNOPSIS

xdotool *cmd args...*

Notation: Some documentation uses *[window]* to denote an optional window argument. This case means that the argument, if not present, will default to "%1". See "WINDOW STACK" for what "%1" means.

DESCRIPTION

xdotool lets you programmatically (or manually) simulate keyboard input and mouse activity, move and resize windows, etc. It does this using X11's XTEST extension and other Xlib functions.

There is some support for Extended Window Manager Hints (aka EWMH or NetWM). See the "EXTENDED WINDOW MANAGER HINTS" section for more information.

KEYBOARD COMMANDS

key *[options] keystroke* [*keystroke ...*]

Options:

−−window window

Send keystrokes to a specific window id. You can use "WINDOW STACK" references like "%1" and "%@" here. If there is a window stack, then "%1" is the default, otherwise the current window is used.

See also: "SENDEVENT NOTES" and "WINDOW STACK"

−−clearmodifiers

Clear modifiers before sending keystrokes. See CLEARMODIFIERS below.

−−delay milliseconds

Delay between keystrokes. Default is 12ms.

Type a given keystroke. Examples being "alt+r", "Control_L+J", "ctrl+alt+n", "BackSpace".

Generally, any valid X Keysym string will work. Multiple keys are separated by '+'. Aliases exist for "alt", "ctrl", "shift", "super", and "meta" which all map to Foo_L, such as Alt_L and Control_L, etc.

In cases where your keyboard doesn't actually have the key you want to type, xdotool will automatically find an unused keycode and use that to type the key.

With respect to "COMMAND CHAINING", this command consumes the remainder of the arguments or until a new xdotool command is seen, because no xdotool commands are valid keystrokes.

Example: Send the keystroke "F2"
```
 xdotool key F2
```

Example: Send 'a' with an accent over it (not on English keyboards, but still works with xdotool)
```
 xdotool key Aacute
```

Example: Send ctrl+l and then BackSpace as separate keystrokes:
```
 xdotool key ctrl+l BackSpace
```

Example: Send ctrl+c to all windows matching title 'gdb' (See "COMMAND CHAINING")
```
 xdotool search −−name gdb key ctrl+c
```

keydown *[options] keystroke*

Same as above, except only keydown (press) events are sent.

keyup *keystroke*

Same as above, except only keyup (release) events are sent.

type *[options] something to type*

Options:

−−window windowid
Send keystrokes to a specific window id. See "SENDEVENT NOTES" below. The default, if no window is given, depends on the window stack. If the window stack is empty the current window is typed at using XTEST. Otherwise, the default is "%1" (see "WINDOW STACK").

−−delay milliseconds
Delay between keystrokes. Default is 12ms.

−−clearmodifiers
Clear modifiers before sending keystrokes. See CLEARMODIFIERS below.

Types as if you had typed it. Supports newlines and tabs (ASCII newline and tab). Each keystroke is separated by a delay given by the **−−delay** option.

With respect to "COMMAND CHAINING", this command consumes the remainder of the arguments and types them. That is, no commands can chain after 'type'.

Example: to type 'Hello world!' you would do:

```
 xdotool type 'Hello world!'
```

MOUSE COMMANDS

mousemove *[options] x y OR 'restore'*
Move the mouse to the specific X and Y coordinates on the screen.

You can move the mouse to the previous location if you specify 'restore' instead of an X and Y coordinate. Restoring only works if you have moved previously in this same command invocation. Further, it does not work with the −−window option.

For example, to click the top-left corner of the screen and move the mouse to the original position before you moved it, use this:

```
 xdotool mousemove 0 0 click 1 mousemove restore
```

−−window WINDOW
Specify a window to move relative to. Coordinates 0,0 are at the top left of the window you choose.

"WINDOW STACK" references are valid here, such as `%1` and `%@`. Though, using `%@` probably doesn't make sense.

−−screen SCREEN
Move the mouse to the specified screen to move to. This is only useful if you have multiple screens and ARE NOT using Xinerama.

The default is the current screen. If you specify −−window, the −−screen flag is ignored.

−−polar
Use polar coordinates. This makes 'x' an angle (in degrees, 0−360, etc) and 'y' the distance.

Rotation starts at 'up' (0 degrees) and rotates clockwise: 90 = right, 180 = down, 270 = left.

The origin defaults to the center of the current screen. If you specify a −−window, then the origin is the center of that window.

−−clearmodifiers
See CLEARMODIFIERS

−−sync
After sending the mouse move request, wait until the mouse is actually moved. If no movement is necessary, we will not wait. This is useful for scripts that depend on actions being completed before moving on.

Note: We wait until the mouse moves at all, not necessarily that it actually reaches your intended destination. Some applications lock the mouse cursor to certain regions of the screen, so waiting for any movement is better in the general case than waiting for a specific target.

mousemove_relative [options] *x y*

Move the mouse x,y pixels relative to the current position of the mouse cursor.

−−polar

Use polar coordinates. This makes 'x' an angle (in degrees, 0−360, etc) and 'y' the distance.

Rotation starts at 'up' (0 degrees) and rotates clockwise: 90 = right, 180 = down, 270 = left.

−−sync

After sending the mouse move request, wait until the mouse is actually moved. If no movement is necessary, we will not wait. This is useful for scripts that depend on actions being completed before moving on.

Note that we wait until the mouse moves at all, not necessarily that it actually reaches your intended destination. Some applications lock the mouse cursor to certain regions of the screen, so waiting for any movement is better in the general case than waiting for a specific target.

−−clearmodifiers

See CLEARMODIFIERS

click *[options] button*

Send a click, that is, a mousedown followed by mouseup for the given button with a short delay between the two (currently 12ms).

Buttons generally map this way: Left mouse is 1, middle is 2, right is 3, wheel up is 4, wheel down is 5.

−−clearmodifiers

Clear modifiers before clicking. See CLEARMODIFIERS below.

−−repeat REPEAT

Specify how many times to click. Default is 1. For a double-click, use '−−repeat 2'

−−delay MILLISECONDS

Specify how long, in milliseconds, to delay between clicks. This option is not used if the *−−repeat* flag is set to 1 (default).

−−window WINDOW

Specify a window to send a click to. See "SENDEVENT NOTES" below for caveats. Uses the current mouse position when generating the event.

The default, if no window is given, depends on the window stack. If the window stack is empty the current window is typed at using XTEST. Otherwise, the default is "%1" (see "WINDOW STACK").

mousedown *[options] button*

Same as **click**, except only a mouse down is sent.

mouseup *[options] button*

Same as **click**, except only a mouse up is sent.

getmouselocation *[−−shell]*

Outputs the x, y, screen, and window id of the mouse cursor. Screen numbers will be nonzero if you have multiple monitors and are not using Xinerama.

−−shell

This makes getmouselocation output shell data you can eval. Example:

```
% xdotool getmouselocation --shell
X=880
Y=443
SCREEN=0
WINDOW=16777250
```

```
% eval $(xdotool getmouselocation --shell)
% echo $X,$Y
714,324
```

behave_screen_edge *[options] where command ...*

Bind an action to events when the mouse hits the screen edge or corner.

Options are:

−−*delay MILLISECONDS*

Delay in milliseconds before running the command. This allows you to require a given edge or corner to be held for a short period before your command will run. If you leave the edge or corner before the delay expires then the time will reset.

−−*quiesce MILLISECONDS*

Delay in milliseconds before the next command will run. This helps prevent accidentally running your command extra times; especially useful if you have a very short −−delay (like the default of 0).

Event timeline

```
 * Mouse hits an edge or corner.
 * If delay is nonzero, the mouse must stay in this edge or corner until delay
 * If still in the edge/corner, trigger.
 * If quiesce is nonzero, then there is a cool-down period where the next
   trigger cannot occur
```

Valid 'where' values are:

left
top-left
top
top-right
right
bottom-left
bottom
bottom-right

Examples:
 # Activate google-chrome when you move the mouse to the bottom-left corner:
 xdotool behave_screen_edge bottom-left \
  search −−class google-chrome windowactivate

```
 # Go to the next workspace (right). Known to work in GNOME (metacity and comp
 xdotool behave_screen_edge --delay 500 bottom-right key XF86Forward

 # Activate firefox and do a web search in a new tab for text in your clipboar
 xdotool behave_screen_edge --delay 1000 top-left \
     search --classname Navigator \
     windowactivate --sync key --delay 250 ctrl+t ctrl+k ctrl+v Return
```

WINDOW COMMANDS

search *[options] pattern*

Search for windows with titles, names, or classes with a regular expression pattern. The output is line-delimited list of X window identifiers. If you are using "COMMAND CHAINING", the search command will only write window ids to stdout if it is the last (or only) command in the chain; otherwise, it is silent.

The result is saved to the window stack for future chained commands. See "WINDOW STACK" and "COMMAND CHAINING" for details.

The default options are `--name --class --classname` (unless you specify one one or more of −−name −−class or −−classname).

The options available are:

−−class
Match against the window class.

−−classname
Match against the window classname.

−−maxdepth N
Set recursion/child search depth. Default is −1, meaning infinite. 0 means no depth, only root windows will be searched. If you only want toplevel windows, set maxdepth of 1 (or 2, depending on how your window manager does decorations).

−−name
Match against the window name. This is the same string that is displayed in the window titlebar.

−−onlyvisible
Show only visible windows in the results. This means ones with map state IsViewable.

−−pid PID
Match windows that belong to a specific process id. This may not work for some X applications that do not set this metadata on its windows.

−−screen N
Select windows only on a specific screen. Default is to search all screens. Only meaningful if you have multiple displays and are not using Xinerama.

−−desktop N
Only match windows on a certain desktop. 'N' is a number. The default is to search all desktops.

−−limit N
Stop searching after finding N matching windows. Specifying a limit will help speed up your search if you only want a few results.

The default is no search limit (which is equivalent to '−−limit 0')

−−title
DEPRECATED. See −−name.

−−all
Require that all conditions be met. For example:

```
xdotool search --all --pid 1424 --name "Hello World"
```

This will match only windows that have "Hello World" as a name and are owned by pid 1424.

−−any
Match windows that match any condition (logically, 'or'). This is on by default. For example:

```
xdotool search --any --pid 1424 --name "Hello World"
```

This will match any windows owned by pid 1424 or windows with name "Hello World"

−−sync
Block until there are results. This is useful when you are launching an application and want to wait until the application window is visible. For example:

```
google-chrome &
xdotool search --sync --onlyvisible --class "google-chrome"
```

selectwindow
Get the window id (for a client) by clicking on it. Useful for having scripts query you humans for what window to act on. For example, killing a window by clicking on it:

```
xdotool selectwindow windowkill
```

behave *window action command ...*

Bind an action to an event on a window. This lets you run additional xdotool commands whenever a matched event occurs.

The command run as a result of the behavior is run with `%1` being the window that was acted upon. Examples follow after the event list.

The following are valid events:

mouse-enter
Fires when the mouse enters a window. This is similar to 'mouse over' events in javascript, if that helps.

mouse-leave
Fires when the mouse leaves a window. This is the opposite of 'mouse−enter'

mouse-click
Fires when the mouse is clicked. Specifically, when the mouse button is released.

focus
Fires when the window gets input focus.

blur
Fires when the window loses focus.

Examples:

```
# Print the cursor location whenever the mouse enters a currently-visible
# window:
xdotool search --onlyvisible . behave %@ mouse-enter getmouselocation

# Print the window title and pid whenever an xterm gets focus
xdotool search --class xterm behave %@ focus getwindowname getwindowpid

# Emulate focus-follows-mouse
xdotool search . behave %@ mouse-enter windowfocus
```

getwindowpid *[window]*

Output the PID owning a given window. This requires effort from the application owning a window and may not work for all windows. This uses _NET_WM_PID property of the window. See "EXTENDED WINDOW MANAGER HINTS" below for more information.

If no window is given, the default is '%1'. If no windows are on the stack, then this is an error. See "WINDOW STACK" for more details.

Example: Find the PID for all xterms:
xdotool search −−class xterm getwindowpid %@

getwindowname *[window]*

Output the name of a given window, also known as the title. This is the text displayed in the window's titlebar by your window manager.

If no window is given, the default is '%1'. If no windows are on the stack, then this is an error. See "WINDOW STACK" for more details.

getwindowgeometry [options] *[window]*

Output the geometry (location and position) of a window. The values include: x, y, width, height, and screen number.

−−shell
Output values suitable for 'eval' in a shell.

getwindowfocus [−f]

Prints the window id of the currently focused window. Saves the result to the window stack. See “WINDOW STACK” for more details.

If the current window has no WM_CLASS property, we assume it is not a normal top-level window and traverse up the parents until we find a window with a WM_CLASS set and return that window id.

If you really want the window currently having focus and don’t care if it has a WM_CLASS setting, then use ’getwindowfocus −f’

windowsize [options] [window] width height

Set the window size of the given window. If no window is given, `%1` is the default. See “WINDOW STACK” and “COMMAND CHAINING” for more details.

Percentages are valid for width and height. They are relative to the geometry of the screen the window is on. For example, to make a window the full width of the screen, but half height:

```
 xdotool windowsize I<window> 100% 50%
```

Percentages are valid with −−usehints and still mean pixel-width relative to the screen size.

The options available are:

−−usehints

Use window sizing hints (when available) to set width and height. This is useful on terminals for setting the size based on row/column of text rather than pixels.

−−sync

After sending the window size request, wait until the window is actually resized. If no change is necessary, we will not wait. This is useful for scripts that depend on actions being completed before moving on.

Note: Because many window managers may ignore or alter the original resize request, we will wait until the size changes from its original size, not necessary to the requested size.

Example: To set a terminal to be 80x24 characters, you would use:
xdotool windowsize −−usehints *some_windowid* 80 24

windowmove *[options] [window] x y*

Move the window to the given position. If no window is given, `%1` is the default. See “WINDOW STACK” and “COMMAND CHAINING” for more details.

If the given x coordinate is literally ’x’, then the window’s current x position will be unchanged. The same applies for ’y’.

Examples:

```
 xdotool getactivewindow windowmove 100 100    # Moves to 100,100
 xdotool getactivewindow windowmove x 100      # Moves to x,100
 xdotool getactivewindow windowmove 100 y      # Moves to 100,y
 xdotool getactivewindow windowmove 100 y      # Moves to 100,y
```

Percentages are valid for width and height. They are relative to the geometry of the screen the window is on. For example, to make a window the full width of the screen, but half height:

```
 xdotool windowmove I<window> 100% 50%
```

−−sync

After sending the window move request, wait until the window is actually moved. If no movement is necessary, we will not wait. This is useful for scripts that depend on actions being completed before moving on.

−−relative

Make movement relative to the current window position.

windowfocus *[options] [window]*

Focus a window. If no window is given, `%1` is the default. See “WINDOW STACK” and “COMMAND CHAINING” for more details.

Uses XSetInputFocus which may be ignored by some window managers or programs.

−−sync

After sending the window focus request, wait until the window is actually focused. This is useful for scripts that depend on actions being completed before moving on.

windowmap *[options] [window]*

Map a window. In X11 terminology, mapping a window means making it visible on the screen. If no window is given, `%1` is the default. See “WINDOW STACK” and “COMMAND CHAINING” for more details.

−−sync

After requesting the window map, wait until the window is actually mapped (visible). This is useful for scripts that depend on actions being completed before moving on.

windowminimize *[options] [window]*

Minimize a window. In X11 terminology, this is called 'iconify.' If no window is given, `%1` is the default. See “WINDOW STACK” and “COMMAND CHAINING” for more details.

−−sync

After requesting the window minimize, wait until the window is actually minimized. This is useful for scripts that depend on actions being completed before moving on.

windowraise *[window_id=%1]*

Raise the window to the top of the stack. This may not work on all window managers. If no window is given, `%1` is the default. See “WINDOW STACK” and “COMMAND CHAINING” for more details.

windowreparent *[source_window] destination_window*

Reparent a window. This moves the *source_window* to be a child window of *destination_window*. If no source is given, `%1` is the default. “WINDOW STACK” window references (like `%1`) are valid for both *source_window* and *destination_window* See “WINDOW STACK” and “COMMAND CHAINING” for more details.

windowclose *[window]*

Close a window. This action will destroy the window, but will not try to kill the client controlling it. If no window is given, `%1` is the default. See “WINDOW STACK” and “COMMAND CHAINING” for more details.

windowkill *[window]*

Kill a window. This action will destroy the window and kill the client controlling it. If no window is given, `%1` is the default. See WINDOW STACK and “COMMAND CHAINING” for more details.

windowunmap *[options] [window_id=%1]*

Unmap a window, making it no longer appear on your screen. If no window is given, `%1` is the default. See “WINDOW STACK” and “COMMAND CHAINING” for more details.

−−sync

After requesting the window unmap, wait until the window is actually unmapped (hidden). This is useful for scripts that depend on actions being completed before moving on.

set_window *[options] [windowid=%1]*

Set properties about a window. If no window is given, `%1` is the default. See “WINDOW STACK” and “COMMAND CHAINING” for more details.

Options:

−−name newname

Set window WM_NAME (the window title, usually)

−−icon−name newiconname
Set window WM_ICON_NAME (the window title when minimized, usually)

−−role newrole
Set window WM_WINDOW_ROLE

−−classname newclassname
Set window class name (not to be confused with window class)

−−class newclass
Set window class (not to be confused with window class name)

−−urgency value
Set window urgency hint. If the value is 1, the window will be marked urgent, and the window manager will somehow highlight it for the user's attention. If the value is 0, the window will be marked non-urgent.

−−overrideredirect value
Set window's override_redirect value. This value is a hint to the window manager for whether or not it should be managed. If the redirect value is 0, then the window manager will draw borders and treat this window normally. If the value is 1, the window manager will ignore this window.

If you change this value, your window manager may not notice the change until the window is mapped again, so you may want to issue 'windowunmap' and 'windowmap' to make the window manager take note.

DESKTOP AND WINDOW COMMANDS

These commands follow the EWMH standard. See the section "EXTENDED WINDOW MANAGER HINTS" for more information.

windowactivate *[options] [window]*
Activate the window. This command is different from windowfocus: if the window is on another desktop, we will switch to that desktop. It also uses a different method for bringing the window up. I recommend trying this command before using windowfocus, as it will work on more window managers.

If no window is given, `%1` is the default. See "WINDOW STACK" and "COMMAND CHAINING" for more details.

−−sync
After sending the window activation, wait until the window is actually activated. This is useful for scripts that depend on actions being completed before moving on.

getactivewindow
Output the current active window. This command is often more reliable than getwindowfocus. The result is saved to the window stack. See "WINDOW STACK" for more details.

set_num_desktops *number*
Changes the number of desktops or workspaces.

get_num_desktops
Output the current number of desktops.

get_desktop_viewport *[−−shell]*
Report the current viewport's position. If −−shell is given, the output is friendly to shell eval.

Viewports are sometimes used instead of 'virtual desktops' on some window managers. A viewport is simply a view on a very large desktop area.

set_desktop_viewport *x y*
Move the viewport to the given position. Not all requests will be obeyed − some windowmangers only obey requests that align to workspace boundaries, such as the screen size.

For example, if your screen is 1280x800, you can move to the 2nd workspace by doing:

```
xdotool set_desktop_viewport 1280 0
```

set_desktop *[options] desktop_number*
Change the current view to the specified desktop.

−−relative
Use relative movements instead of absolute. This lets you move relative to the current desktop.

get_desktop
Output the current desktop in view.

set_desktop_for_window *[window] desktop_number*
Move a window to a different desktop. If no window is given, `%1` is the default. See "WINDOW STACK" and "COMMAND CHAINING" for more details.

get_desktop_for_window *[window]*
Output the desktop currently containing the given window. Move a window to a different desktop. If no window is given, `%1` is the default. See WINDOW STACK and "COMMAND CHAINING" for more details.

MISCELLANEOUS COMMANDS

exec *[options] command [...]*
Execute a program. This is often useful when combined with behave_screen_edge to do things like locking your screen.

Options:

−−sync
Block until the child process exits. The child process exit status is then passed to the parent process (xdotool) which copies it.

Examples:
```
 # Lock the screen when the mouse sits in the top-right corner
 xdotool behave_screen_edge −−delay 1000 top-right \
   exec gnome-screensaver-command −−lock
 # Substitute 'xscreensaver−command −lock' if you use that program.
```

```
 # The following will fail to move the mouse because we use '--sync' and
 # /bin/false exits nonzero:
 xdotool exec --sync /bin/false mousemove 0 0

 # This succeeds, though, since we do not use --sync on the exec command.
 xdotool exec /bin/false mousemove 0 0
```

sleep *seconds*
Sleep for a specified period. Fractions of seconds (like 1.3, or 0.4) are valid, here.

SCRIPTS

xdotool can read a list of commands via stdin or a file if you want. A script will fail when any command fails.

Truthfully, 'script' mode isn't fully fleshed out and may fall below your expectations. If you have suggestions, please email the list or file a bug (See CONTACT).

Scripts can use positional arguments (Represented by `$1`, `$2`, ...) and environment variables (like `$HOME` or `$WINDOWID`). Quoting arguments should work as expected.

Scripts are processed for parameter and environment variable expansion and then run as if you had invoked xdotool with the entire script on one line (using COMMAND CHAINING).

- Read commands from a file:

```
 xdotool filename
```

- Read commands from stdin:

  ```
  xdotool -
  ```

- Read commands from a redirected file

  ```
  xdotool - < myfile
  ```

You can also write scripts that only execute xdotool. Example:

```
#!/usr/local/bin/xdotool
search --onlyvisible --classname $1

windowsize %@ $2 $3
windowraise %@

windowmove %1 0 0
windowmove %2 $2 0
windowmove %3 0 $3
windowmove %4 $2 $3
```

This script will take all windows matched by the classname query given by arg1 ($1) and sizes/moves them into a 2x2 grid with windows sized by the 2nd and 3rd parameters.

Here's an example usage:

```
% ./myscript xterm 600 400
```

Running it like this will take 4 visible xterms, raise them, and move them into a 2x2 tile grid with each window 600x400 pixels in size.

CLEARMODIFIERS

Any command taking the *−−clearmodifiers* flag will attempt to clear any active input modifiers during the command and restore them afterwards.

For example, if you were to run this command:
 xdotool key a

The result would be 'a' or 'A' depending on whether or not you were holding the shift key on your keyboard. Often it is undesirable to have any modifiers active, so you can tell xdotool to clear any active modifiers.

The order of operations if you hold shift while running 'xdotool key −−clearmodifiers a' is this:

1. Query for all active modifiers (finds shift, in this case)
2. Try to clear shift by sending 'key up' for the shift key
3. Runs normal 'xdotool key a'
4. Restore shift key by sending 'key down' for shift

The *−−clearmodifiers* flag can currently clear of the following:

- any key in your active keymap that has a modifier associated with it. (See *xmodmap* (1)'s 'xmodmap −pm' output)
- mouse buttons (1, 2, 3, 4, and 5)
- caps lock

SENDEVENT NOTES

If you are trying to send key input to a specific window, and it does not appear to be working, then it's likely your application is ignoring the events xdotool is generating. This is fairly common.

Sending keystrokes to a specific window uses a different API than simply typing to the active window. If you specify 'xdotool type −−window 12345 hello' xdotool will generate key events and send them directly to window 12345. However, X11 servers will set a special flag on all events generated in this way (see XEvent.xany.send_event in X11's manual). Many programs observe this flag and reject these events.

It is important to note that for key and mouse events, we only use XSendEvent when a specific window is targeted. Otherwise, we use XTEST.

Some programs can be configured to accept events even if they are generated by xdotool. Seek the documentation of your application for help.

Specific application notes (from the author's testing): * Firefox 3 seems to ignore all input when it does not have focus. * xterm can be configured while running with ctrl+leftclick, 'Allow SendEvents' * gnome-terminal appears to accept generated input by default.

WINDOW STACK

Certain commands (search, getactivewindow, getwindowfocus) will find windows for you. These results generally printed to stdout, but they are also saved to memory for future use during the lifetime of the xdotool process. See "COMMAND CHAINING" for more information.

The only modifications support for the window stack are to replace it. That is, two of two sequential searches, only the last one's results will be the window stack.

COMMAND CHAINING

xdotool supports running multiple commands on a single invocation. Generally, you'll start with a search command (see "WINDOW STACK") and then perform a set of actions on those results.

To query the window stack, you can use special notation "%N" where N is a number or the '@' symbol. If `%N` is given, the Nth window will be selected from the window stack. Generally you will only want the first window or all windows. Note that the order of windows in the window stack corresponds to the window stacking order, i.e. the bottom-most window will be reported first (see *XQueryTree* (3)). Thus the order of the windows in the window stack may not be consistent across invocations.

The notation described above is used as the "window" argument for any given command.

For example, to resize all xterms to 80x24:

```
xdotool search --class xterm -- windowsize --usehints %@ 80 24
```

Resize move the current window:

```
xdotool getactivewindow windowmove 0 0
```

In all cases, the default window argument, if omitted, will default to "%1". It is obviously an error if you omit the window argument and the window stack is empty. If you try to use the window stack and it is empty, it is also an error.

To activate the first firefox window found:

```
xdotool search --class firefox windowactivate
```

These would error:

```
xdotool windowactivate
xdotool windowactivate %1
xdotool windowactivate %@
```

When xdotool exits, the current window stack is lost.

Additionally, commands that modify the "WINDOW STACK" will not print the results if they are not the last command. For example:

```
# Output the active window:
% xdotool getactivewindow
20971533

# Output the pid of the active window, but not the active window id:
% xdotool getactivewindow getwindowpid
4686
```

EXTENDED WINDOW MANAGER HINTS

The following pieces of the EWMH standard are supported:

_NET_SUPPORTED
: Asks the window manager what is supported

_NET_CURRENT_DESKTOP
: Query and set the current desktop. Support for this enables these commands: `set_desktop`, `get_desktop`.

_NET_WM_DESKTOP
: Query and set what desktop a window is living in. Support for this enables these commands: `set_desktop_for_window`, `get_desktop_for_window`.

_NET_ACTIVE_WINDOW
: Allows you to query and set the active window by asking the window manager to bring it forward. Support for this enables these commands: `windowactivate`, `getactivewindow`.

_NET_WM_PID
: This feature is application dependent, not window-manager dependent. Query the PID owning a given window. Support for this enables these commands: `getwindowpid`.

SUPPORTED FEATURES

xdotool (and libxdo) will try to function under all circumstances. However, there may be some cases where functionality is not provided by your X server or by your window manager. In these cases, xdotool will try to detect and tell you if an action requires a feature not currently supported by your system.

For window-manager specific features, see ‘‘EXTENDED WINDOW MANAGER HINTS’’.

XTEST
: If your X server does not support XTEST, then some typing and mouse movement features may not work. Specifically, typing and mouse actions that act on the ‘‘current window’’ (window 0 in libxdo) are unlikely to work.

  In most cases, XTEST is a feature you can enable on your X server if it is not enabled by default.

  You can see the list of supported X extensions by typing ’xdpyinfo’ and looking the text ’number of extensions: ...’

BUGS

Typing unusual symbols under non-us keybindings is known to occasionally send the wrong character.

SEE ALSO

xprop (1), *xwininfo* (1),

Project site: <http://www.semicomplete.com/projects/xdotool>

Source code and Issues: <https://github.com/jordansissel/xdotool>

EWMH specification: <http://standards.freedesktop.org/wm−spec/wm−spec−1.3.html>

CONTACT

Please send questions to xdotool−users@googlegroups.com. File bugs and feature requests at the following URL:

<https://github.com/jordansissel/xdotool/issues>

Alternately, if you prefer email, feel free to file bugs by emailing the list. What works for you :)

AUTHOR

xdotool was written by Jordan Sissel.

This manual page was written originally by Daniel Kahn Gillmor <dkg@fifthhorseman.net> for the Debian project (but may be used by others). It is maintained by Jordan Sissel.

Patches, ideas, and other contributions by many, nice folks. See the CHANGELIST file for who provided what.