NAME

event − Miscellaneous event facilities: define virtual events and generate events

SYNOPSIS

event *option* ?*arg arg* ...?

DESCRIPTION

The **event** command provides several facilities for dealing with window system events, such as defining virtual events and synthesizing events. The command has several different forms, determined by the first argument. The following forms are currently supported:

event add <<*virtual*>> *sequence* ?*sequence* ...?

Associates the virtual event *virtual* with the physical event sequence(s) given by the *sequence* arguments, so that the virtual event will trigger whenever any one of the *sequence*s occurs. *V irtual* may be any string value and *sequence* may have any of the values allowed for the *sequence* argument to the **bind** command. If *virtual* is already defined, the new physical event sequences add to the existing sequences for the event.

event delete <<*virtual*>> ?*sequence sequence* ...?

Deletes each of the *sequence*s from those associated with the virtual event given by *virtual*. *Virtual* may be any string value and *sequence* may have any of the values allowed for the *sequence* argument to the **bind** command. Any *sequence*s not currently associated with *virtual* are ignored. If no *sequence* argument is provided, all physical event sequences are removed for *virtual*, so that the virtual event will not trigger anymore.

event generate *window event* ?*option value option value* ...?

Generates a window event and arranges for it to be processed just as if it had come from the window system. *W indow* gives the path name of the window for which the event will be generated; it may also be an identifier (such as returned by **winfo id**) as long as it is for a window in the current application. *Event* provides a basic description of the event, such as **<Shift-Button-2>** or <<**Paste**>>. If*W indow* is empty the whole screen is meant, and coordinates are relative to the screen. *Event* may have any of the forms allowed for the *sequence* argument of the **bind** command except that it must consist of a single event pattern, not a sequence. *Option-value* pairs may be used to specify additional attributes of the event, such as the x and y mouse position; see **EVENT FIELDS** below. If the−**when** option is not specified, the e vent is processed immediately: all of the handlers for the event will complete before the **event generate** command returns. If the **−when** option is specified then it determines when the event is processed. Certain events, such as key events, require that the window has focus to receive the event properly.

event info ?<<*virtual*>>?

Returns information about virtual events. If the<<*virtual*>> ar gument is omitted, the return value is a list of all the virtual events that are currently defined. If <<*virtual*>> is specified then the return value is a list whose elements are the physical event sequences currently defined for the given virtual event; if the virtual event is not defined then an empty string is returned.

Note that virtual events that are not bound to physical event sequences are *not* returned by **event info**.

EVENT FIELDS

The following options are supported for the **event generate** command. These correspond to the "%" expansions allowed in binding scripts for the **bind** command.

−above *window*

Window specifies the *above* field for the event, either as a window path name or as an integer window id. Valid for **Configure** events. Corresponds to the**%a** substitution for binding scripts.

−borderwidth *size*
Size must be a screen distance; it specifies the *border_width* field for the event. Valid for **Configure** events. Corresponds to the **%B** substitution for binding scripts.

−button *number*
Number must be an integer; it specifies the *detail* field for a **ButtonPr ess** or **ButtonRelease** event, overriding any button number provided in the base *event* argument. Corresponds to the **%b** substitution for binding scripts.

−count *number*
Number must be an integer; it specifies the *count* field for the e vent. Valid for **Expose** events. Corresponds to the **%c** substitution for binding scripts.

−data *string*
String may be any value; it specifies the *user_data* field for the event. Only valid for virtual events. Corresponds to the **%d** substitution for virtual e vents in binding scripts.

−delta *number*
Number must be an integer; it specifies the *delta* field for the **MouseWheel** e vent. The *delta* refers to the direction and magnitude the mouse wheel was rotated. Note the value is not a screen distance but are units of motion in the mouse wheel. Typically these values are multiples of 120. For example, 120 should scroll the text widget up 4 lines and −240 would scroll the text widget down 8 lines. Of course, other widgets may define different behaviors for mouse wheel motion. This field corresponds to the **%D** substitution for binding scripts.

−detail *detail*
Detail specifies the *detail* field for the event and must be one of the following:

NotifyAncestor	**NotifyNonlinearVirtual**
NotifyDetailNone	**NotifyPointer**
NotifyInferior	**NotifyPointerRoot**
NotifyNonlinear	**NotifyVirtual**

Valid for **Enter**, **Leave**, **FocusIn** and **FocusOut** events. Corresponds to the **%d** substitution for binding scripts.

−focus *boolean*
Boolean must be a boolean value; it specifies the *focus* field for the e vent. Valid for **Enter** and **Leave** events. Corresponds to the **%f** substitution for binding scripts.

−height *size*
Size must be a screen distance; it specifies the *height* field for the event. Valid for **Configure** events. Corresponds to the **%h** substitution for binding scripts.

−keycode *number*
Number must be an integer; it specifies the *eycode* field for the event. Valid for **KeyPress** and **KeyRelease** events. Corresponds to the **%k** substitution for binding scripts.

−keysym *name*
Name must be the name of a valid keysym, such as **g**, **space**, or **Return**; its corresponding keycode value is used as the *keycode* field for event, overriding any detail specified in the base *event* argument. Valid for **KeyPress** and **KeyRelease** events. Corresponds to the **%K** substitution for binding scripts.

−mode *notify*
Notify specifies the *mode* field for the event and must be one of **NotifyNormal**, **NotifyGrab**, **NotifyUngrab**, or **NotifyWhileGrabbed**. Valid for **Enter**, **Leave**, **FocusIn**, and **FocusOut** events. Corresponds to the **%m** substitution for binding scripts.

−override *boolean*
Boolean must be a boolean value; it specifies the*o verride_redirect* field for the event. Valid for **Map**, **Reparent**, and **Configure** events. Corresponds to the**%o** substitution for binding scripts.

−place *where*
Where specifies the *place* field for the event; it must be either**PlaceOnT op** or **PlaceOnBottom**. Valid for **Circulate** events. Corresponds to the**%p** substitution for binding scripts.

−root *window*
Window must be either a window path name or an integer window identifier; it specifies the *root* field for the event. Valid for **KeyPress**, **KeyRelease**, **ButtonPress**, **ButtonRelease**, **Enter**, **Leave**, and **Motion** events. Corresponds to the**%R** substitution for binding scripts.

−rootx *coord*
Coord must be a screen distance; it specifies the *x_root* field for the event. Valid for **KeyPress**, **KeyRelease**, **ButtonPress**, **ButtonRelease**, **Enter**, **Leave**, and **Motion** events. Corresponds to the **%X** substitution for binding scripts.

−rooty *coord*
Coord must be a screen distance; it specifies the *y_root* field for the event. Valid for **KeyPress**, **KeyRelease**, **ButtonPress**, **ButtonRelease**, **Enter**, **Leave**, and **Motion** events. Corresponds to the **%Y** substitution for binding scripts.

−sendevent *boolean*
Boolean must be a boolean value; it specifies the*send_e vent* field for the event. Valid for all events. Corresponds to the**%E** substitution for binding scripts.

−serial *number*
Number must be an integer; it specifies the*serial* field for the e vent. Valid for all events. Corresponds to the **%#** substitution for binding scripts.

−state *state*
State specifies the *state* field for the event. For **KeyPress**, **KeyRelease**, **ButtonPress**, **ButtonRelease**, **Enter**, **Leave**, and **Motion** events it must be an integer value. For **Visibility** events it must be one of **VisibilityUnobscured**, **VisibilityPartiallyObscured**, or **VisibilityFullyObscured**. This option overrides any modifiers such as **Meta** or **Control** specified in the base *event*. Corresponds to the **%s** substitution for binding scripts.

−subwindow *window*
Window specifies the *subwindow* field for the event, either as a path name for a Tk widget or as an integer window identifier. Valid for **KeyPress**, **KeyRelease**, **ButtonPress**, **ButtonRelease**, **Enter**, **Leave**, and **Motion** events. Similar to**%S** substitution for binding scripts.

−time *integer*
Integer must be an integer value; it specifies the*time* field for the e vent. Additonally the special value **current** is allowed, this value will be substituted by the current event time. Valid for **KeyPress**, **KeyRelease**, **ButtonPress**, **ButtonRelease**, **Enter**, **Leave**, **Motion**, and **Property** events. Corresponds to the **%t** substitution for binding scripts.

−warp *boolean*
boolean must be a boolean value; it specifies whether the screen pointer should be warped as well. Valid for **KeyPress**, **KeyRelease**, **ButtonPress**, **ButtonRelease**, and **Motion** events. The pointer will only warp to a window if it is mapped.

−width *size*
Size must be a screen distance; it specifies the *width* field for the event. Valid for **Configure** events. Corresponds to the**%w** substitution for binding scripts.

−when *when*

When determines when the event will be processed; it must have one of the following values:

now Process the event immediately, before the command returns. This also happens if the **−when** option is omitted.

tail Place the event on Tcl's event queue behind any events already queued for this application.

head Place the event at the front of Tcl's event queue, so that it will be handled before any other events already queued.

mark Place the event at the front of Tcl's event queue but behind any other events already queued with **−when mark**. This option is useful when generating a series of events that should be processed in order but at the front of the queue.

−x *coord*

Coord must be a screen distance; it specifies the *x* field for the event. Valid for **KeyPress**, **KeyRelease**, **ButtonPress**, **ButtonRelease**, **Motion**, **Enter**, **Leave**, **Expose**, **Configure**, **Gravity**, and **Reparent** events. Corresponds to the **%x** substitution for binding scripts. If *W indow* is empty the coordinate is relative to the screen, and this option corresponds to the **%X** substitution for binding scripts.

−y *coord*

Coord must be a screen distance; it specifies the *y* field for the event. Valid for **KeyPress**, **KeyRelease**, **ButtonPress**, **ButtonRelease**, **Motion**, **Enter**, **Leave**, **Expose**, **Configure**, **Gravity**, and **Reparent** events. Corresponds to the **%y** substitution for binding scripts. If *W indow* is empty the coordinate is relative to the screen, and this option corresponds to the **%Y** substitution for binding scripts.

Any options that are not specified when generating an event are filled with the value 0, except for *serial*, which is filled with the next X event serial number.

PREDEFINED VIRTUAL EVENTS

Tk defines the following virtual events for the purposes of notification:

<<AltUnderlined>>

This is sent to widget to notify it that the letter it has underlined (as an accelerator indicator) with the **−underline** option has been pressed in combination with the Alt key. The usual response to this is to either focus into the widget (or some related widget) or to invoke the widget.

<<Invoke>>

This can be sent to some widgets (e.g. button, listbox, menu) as an alternative to <space>.

<<ListboxSelect>>

This is sent to a listbox when the set of selected item(s) in the listbox is updated.

<<MenuSelect>>

This is sent to a menu when the currently selected item in the menu changes. It is intended for use with context-sensitive help systems.

<<Modified>>

This is sent to a text widget when the contents of the widget are changed.

<<Selection>>

This is sent to a text widget when the selection in the widget is changed.

<<ThemeChanged>>

This is sent to all widgets when the ttk theme changed. The ttk widgets listen to this event and redisplay themselves when it fires. The legacy widgets ignore this event.

<<TkWorldChanged>>
This event is sent to all widgets when a font is changed, for example, by the use of [font configure]. The user_data field (%d) will have the value "FontChanged". For other system wide changes, this event will be sent to all widgets, and the user_data field will indicate the cause of the change. NOTE: all tk and ttk widgets already handle this event internally.

<<TraverseIn>>
This is sent to a widget when the focus enters the widget because of a user-driven “tab to widget” action.

<<TraverseOut>>
This is sent to a widget when the focus leaves the widget because of a user-driven “tab to widget” action.

<<UndoStack>>
This is sent to a text widget when its undo stack or redo stack becomes empty or unempty.

<<WidgetViewSync>>
This is sent to a text widget when its internal data become obsolete, and again when these internal data are back in sync with the widget view. The detail field (%d substitution) is either true (when the widget is in sync) or false (when it is not).

Tk defines the following virtual events for the purposes of unifying bindings across multiple platforms. Users expect them to behave in the following way:

<<Clear>>
Delete the currently selected widget contents.

<<Copy>>
Copy the currently selected widget contents to the clipboard.

<<Cut>>
Move the currently selected widget contents to the clipboard.

<<LineEnd>>
Move to the end of the line in the current widget while deselecting any selected contents.

<<LineStart>>
Move to the start of the line in the current widget while deselecting any selected contents.

<<NextChar>>
Move to the next item (i.e., visible character) in the current widget while deselecting any selected contents.

<<NextLine>>
Move to the next line in the current widget while deselecting any selected contents.

<<NextPara>>
Move to the next paragraph in the current widget while deselecting any selected contents.

<<NextWord>>
Move to the next group of items (i.e., visible word) in the current widget while deselecting any selected contents.

<<Paste>>
Replace the currently selected widget contents with the contents of the clipboard.

<<PasteSelection>>
Insert the contents of the selection at the mouse location. (This event has meaningful **%x** and **%y** substitutions).

<<PrevChar>>
Move to the previous item (i.e., visible character) in the current widget while deselecting any selected contents.

<<PrevLine>>
Move to the previous line in the current widget while deselecting any selected contents.

<<PrevPara>>
Move to the previous paragraph in the current widget while deselecting any selected contents.

<<PrevWindow>>
Traverse to the previous window.

<<PrevWord>>
Move to the previous group of items (i.e., visible word) in the current widget while deselecting any selected contents.

<<Redo>>
Redo one undone action.

<<SelectAll>>
Set the range of selected contents to the complete widget.

<<SelectLineEnd>>
Move to the end of the line in the current widget while extending the range of selected contents.

<<SelectLineStart>>
Move to the start of the line in the current widget while extending the range of selected contents.

<<SelectNextChar>>
Move to the next item (i.e., visible character) in the current widget while extending the range of selected contents.

<<SelectNextLine>>
Move to the next line in the current widget while extending the range of selected contents.

<<SelectNextPara>>
Move to the next paragraph in the current widget while extending the range of selected contents.

<<SelectNextWord>>
Move to the next group of items (i.e., visible word) in the current widget while extending the range of selected contents.

<<SelectNone>>
Reset the range of selected contents to be empty.

<<SelectPrevChar>>
Move to the previous item (i.e., visible character) in the current widget while extending the range of selected contents.

<<SelectPrevLine>>
Move to the previous line in the current widget while extending the range of selected contents.

<<SelectPrevPara>>
Move to the previous paragraph in the current widget while extending the range of selected contents.

<<SelectPrevWord>>
Move to the previous group of items (i.e., visible word) in the current widget while extending the range of selected contents.

<<ToggleSelection>>
Toggle the selection.

<<Undo>>
Undo the last action.

EXAMPLES

MAPPING KEYS TO VIRTUAL EVENTS

In order for a virtual event binding to trigger, two things must happen. First, the virtual event must be defined with the **event add** command. Second, a binding must be created for the virtual event with the **bind** command. Consider the following virtual event definitions:

```
event add <<Paste>> <Control-y>
event add <<Paste>> <Button-2>
event add <<Save>> <Control-X><Control-S>
event add <<Save>> <Shift-F12>
if {[tk windowingsystem] eq "aqua"} {
   event add <<Save>> <Command-s>
}
```

In the **bind** command, a virtual event can be bound like any other builtin event type as follows:

```
bind Entry <<Paste>> {%W insert [selection get]}
```

The double angle brackets are used to specify that a virtual event is being bound. If the user types Control-y or presses button 2, or if a **<<Paste>>** virtual event is synthesized with **event generate**, then the **<<Paste>>** binding will be invoked.

If a virtual binding has the exact same sequence as a separate physical binding, then the physical binding will take precedence. Consider the following example:

```
event add <<Paste>> <Control-y> <Meta-Control-y>
bind Entry <Control-y> {puts Control-y}
bind Entry <<Paste>> {puts Paste}
```

When the user types Control-y the **<Control-y>** binding will be invoked, because a physical event is considered more specific than a virtual event, all other things being equal. However, when the user types Meta-Control-y the **<<Paste>>** binding will be invoked, because the **Meta** modifier in the physical pattern associated with the virtual binding is more specific than the **<Control-y**> sequence for the physical event.

Bindings on a virtual event may be created before the virtual event exists. Indeed, the virtual event never actually needs to be defined, for instance, on platforms where the specific virtual event would be meaningless or ungeneratable.

When a definition of a virtual event changes at run time, all windows will respond immediately to the new definition. Starting from the preceding example, if the following code is executed:

```
bind Entry <Control-y> {}
event add <<Paste>> <Key-F6>
```

the behavior will change such in two ways. First, the shadowed **<<Paste>>** binding will emerge. Typing Control-y will no longer invoke the **<Control-y>** binding, but instead invoke the virtual event **<<Paste>>**. Second, pressing the F6 key will now also invoke the **<<Paste>>** binding.

MOVING THE MOUSE POINTER

Sometimes it is useful to be able to really move the mouse pointer. For example, if you have some software that is capable of demonstrating directly to the user how to use the program. To do this, you need to “warp” the mouse around by using **event generate**, like this:

```
for {set xy 0} {$xy < 200} {incr xy} {
   event generate . <Motion> -x $xy -y $xy -warp 1
   update
```

```
    after 50
}
```

Note that it is usually considered bad style to move the mouse pointer for the user because it removes control from them. Therefore this technique should be used with caution. Also note that it is not guaranteed to function on all platforms.

SEE ALSO

bind(n)

KEYWORDS

event, binding, define, handle, virtual event