NAME

busy − Make Tk widgets busy, temporarily blocking user interactions

SYNOPSIS

tk busy *window* ?*options*?
tk busy hold *window* ?*options*?
tk busy configure *window* ?*option value*?...
tk busy forget *window* ?*window* ?...
tk busy current ?*pattern*?
tk busy status *window*

DESCRIPTION

The **tk busy** command provides a simple means to block mouse pointer events from Tk widgets, while overriding the widget's cursor with a configurable busy cursor. Note this command does not prevent keyboard events from being sent to the widgets made busy.

INTRODUCTION

There are many times in applications where you want to temporarily restrict what actions the user can take. For example, an application could have a "Run" button that when pressed causes some processing to occur. However, while the application is busy processing, you probably don't want the user to be able to click the "Run" button again. You may also want restrict the user from other tasks such as clicking a "Print" button.

The **tk busy** command lets you make Tk widgets busy. This means that user interactions such as button clicks, moving the mouse, typing at the keyboard, etc. are ignored by the widget. You can set a special cursor (like a watch) that overrides the widget's normal cursor, providing feedback that the application (widget) is temporarily busy.

When a widget is made busy, the widget and all of its descendants will ignore pointer events. It's easy to make an entire panel of widgets busy. You can simply make the toplevel widget (such as ".") busy. This is easier and far much more efficient than recursively traversing the widget hierarchy, disabling each widget and re-configuring its cursor.

Often, the **tk busy** command can be used instead of Tk's **grab** command. Unlike **grab** which restricts all user interactions to one widget, with the **tk busy** command you can have more than one widget active (for example, a "Cancel" dialog and a "Help" button).

EXAMPLE

You can make several widgets busy by simply making its ancestor widget busy using the **hold** operation.

```
frame .top
button .top.button; canvas .top.canvas
pack .top.button .top.canvas
pack .top
# . . .
tk busy hold .top
update
```

All the widgets within **.top** (including **.top**) are now busy. Using **update** insures that **tk busy** command will take effect before any other user events can occur.

When the application is no longer busy processing, you can allow user interactions again and free any resources it allocated by the **forget** operation.

```
tk busy forget .top
```

The busy window has a configurable cursor. You can change the busy cursor using the **configure** operation.

```
tk busy configure .top -cursor "watch"
```

Destroying the widget will also clean up any resources allocated by the **tk busy** command.

OPERATIONS

The following operations are available for the **tk busy** command:

tk busy *window* ?*option value*?...
Shortcut for **tk busy hold** command.

tk busy cget *window option*
Queries the **tk busy** command configuration options for *window*. *Window* must be the path name of a widget previously made busy by the **hold** operation. The command returns the present value of the specified *option*. *Option* may have any of the values accepted by the **hold** operation.

tk busy configure *window* ?*option value*?...
Queries or modifies the **tk busy** command configuration options for *window*. *Window* must be the path name of a widget previously made busy by the **hold** operation. If no options are specified, a list describing all of the available options for *window* (see **Tk_ConfigureInfo** for information on the format of this list) is returned. If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns the empty string. *Option* may have any of the values accepted by the **hold** operation.

Please note that the option database is referenced through *window*. For example, if the widget **.frame** is to be made busy, the busy cursor can be specified for it by either **option** command:

```
option add *frame.busyCursor gumby
option add *Frame.BusyCursor gumby
```

tk busy current ?*pattern*?
Returns the pathnames of all widgets that are currently busy. If a *pattern* is given, only the path names of busy widgets matching *pattern* are returned.

tk busy forget *window* ?*window*?...
Releases resources allocated by the **tk busy** command for *window*, including the transparent window. User events will again be received by *window*. Resources are also released when *window* is destroyed. *Window* must be the name of a widget specified in the **hold** operation, otherwise an error is reported.

tk busy hold *window* ?*option value*?...
Makes the specified *window* (and its descendants in the Tk window hierarchy) appear busy. *Window* must be a valid path name of a Tk widget. A transparent window is put in front of the specified window. This transparent window is mapped the next time idle tasks are processed, and the specified window and its descendants will be blocked from user interactions. Normally **update** should be called immediately afterward to insure that the hold operation is in effect before the application starts its processing. The following configuration options are valid:

−cursor *cursorName*
Specifies the cursor to be displayed when the widget is made busy. *CursorName* can be in any form accepted by **Tk_GetCursor**. The default cursor is **wait** on Windows and **watch** on other platforms.

tk busy status *window*
Returns the status of a widget *window*. If *window* presently can not receive user interactions, **1** is returned, otherwise **0**.

EVENT HANDLING

BINDINGS

The event blocking feature is implemented by creating and mapping a transparent window that completely covers the widget. When the busy window is mapped, it invisibly shields the widget and its hierarchy from all events that may be sent. Like Tk widgets, busy windows have widget names in the Tk window hierarchy.

This means that you can use the **bind** command to handle events in the busy window:

```
tk busy hold .frame.canvas
bind .frame.canvas_Busy <Enter> { ... }
```

Normally the busy window is a sibling of the widget. The name of the busy window is "*widget***_Busy**" where *widget* is the name of the widget to be made busy. In the previous example, the pathname of the busy window is "**.frame.canvas_Busy**". The exception is when the widget is a toplevel widget (such as ".") where the busy window can't be made a sibling. The busy window is then a child of the widget named "*widget***._Busy**" where*widg et* is the name of the toplevel widget. In the following example, the pathname of the busy window is "**._Busy**".

```
tk busy hold .
bind ._Busy <Enter> { ... }
```

ENTER/LEAVE EVENTS

Mapping and unmapping busy windows generates Enter/Leave events for all widgets they cover. Please note this if you are tracking Enter/Leave events in widgets.

KEYBOARD EVENTS

When a widget is made busy, the widget is prevented from gaining the keyboard focus by a user clicking on it by the busy window. But if the widget already had focus, it still may receive keyboard events. The widget can also still receive focus through keyboard traversal. To prevent this, you must move focus to another window and make sure the focus can not go back to the widgets made busy (e.g. but restricting focus to a cancel button).

```
pack [frame .frame]
pack [text .frame.text]
tk busy hold .frame
pack [button .cancel -text "Cancel" -command exit]
focus .cancel
bind .cancel <Tab> {break}
bind .cancel <Shift-Tab> {break}
update
```

The above example moves the focus from .frame immediately after invoking the **hold** so that no keyboard events will be sent to **.frame** or any of its descendants. It also makes sure it's not possible to leave button **.cancel** using the keyboard.

PORTABILITY

Note that the **tk busy** command does not currently have any effect on macOS when Tk is built using Aqua support.

SEE ALSO

grab(n)

KEYWORDS

busy, keyboard events, pointer events, window