active


```
#--------------------------------------------------------------------------
# Author:       phillipsme
# Copyright:    (c) phillipsme 2014
# Licence:      Free to use, free to have fun!
# Developer:    Goncalo Bejinha
#--------------------------------------------------------------------------
```

Modules

argparse socket sys

Functions

bin2ip(b)

dec2bin(n, d=None)

errormsg(msg)

ip2bin(ip)

iprange(addressrange)

main()

portscan(target, ports, tcp, udp, verbose)

printmsg(msg)

returnCIDR(c)

FirewallData

index
/root/Área de Trabalho/lpd/FirewallData.py

```
#/usr/bin/env python
#-*- coding: utf-8 -*-
#
# Author: Goncalo Bejinha 13428
#
```

Modules

GeoIP	numpy	re
reportlab.lib.colors	os	sqlite3
csv	matplotlib.pyplot	sys

Classes

Firewall_Data

class **Firewall_Data**

```
    Class that allows to process the firewall log (/var/log/ufw.log).
```

Methods defined here:

build_bars_graph(self, ip_list)

```
Method to create a simple horizontal bar chart.
@param ip_list -> List of all IP's read from the log.
@reference: http://matplotlib.org/examples/lines_bars_and_markers/barh_demo.html
```

geo_ip_lookup(self, ip_address)

```
Method to seach the coordinates of a given IP address.
@param ip_address -> IP to search.
@return record_list -> List containing the latitude and longitude.
```

read_firewall_log(self)

```
Method to read the firewall log and obtain data from it.
@return info -> Data obtained from the log.
@return ip_list -> List of number of attacks by IP.
```

Functions

mapa()

Data

inch = 72.0

gui_tkinter

index
/root/Área de Trabalho/lpd/gui_tkinter.py

```
# -*- coding: utf-8 -*-
# Developer: Goncalo Bejinha
```

Modules

FirewallData active tkMessageBox
Tkinter lan_scan

Functions

LPDInfo()

Data

Area = <Tkinter.Canvas instance>
B_info = <Tkinter.Button instance>
B_lan = <Tkinter.Button instance>
B_mapa = <Tkinter.Button instance>
B_nmap = <Tkinter.Button instance>
top = <Tkinter.Tk instance>

login

index
/root/Área de Trabalho/lpd/login.py

```
#/usr/bin/env python
#-*- coding: utf-8 -*-
#
# Author: Goncalo Bejinha 13428
#
```

Modules

FirewallData active lan_scan tkMessageBox
Tkinter gui_tkinter sys _tkinter

Classes

Tkinter.Tk(Tkinter.Misc, Tkinter.Wm)

LoginMenu

class **LoginMenu**(Tkinter.Tk)

Method resolution order:

LoginMenu
Tkinter.Tk
Tkinter.Misc
Tkinter.Wm

Methods defined here:

__init__(self)

login(self)

Methods inherited from Tkinter.Tk:

__getattr__(self, attr)

```
    Delegate attribute access to the interpreter object
```

destroy(self)
```
    Destroy this and all descendants widgets. This will
    end the application of this Tcl interpreter.
```

loadtk(self)

readprofile(self, baseName, className)
```
    Internal function. It reads BASENAME.tcl and CLASSNAME.tcl into
    the Tcl Interpreter and calls execfile on BASENAME.py and CLASSNAME.py if
    such a file exists in the home directory.
```

report_callback_exception(self, exc, val, tb)
```
    Internal function. It reports exception on sys.stderr.
```

Methods inherited from Tkinter.Misc:

__contains__(self, key)

__getitem__ = cget(self, key)
```
    Return the resource value for a KEY given as string.
```

__setitem__(self, key, value)

__str__(self)
```
    Return the window path name of this widget.
```

after(self, ms, func=None, *args)
```
    Call function once after given time.

    MS specifies the time in milliseconds. FUNC gives the
    function which shall be called. Additional parameters
    are given as parameters to the function call.  Return
    identifier to cancel scheduling with after_cancel.
```

after_cancel(self, id)
```
Cancel scheduling of function identified with ID.

Identifier returned by after or after_idle must be
given as first parameter.
```

after_idle(self, func, *args)
```
Call FUNC once if the Tcl main loop has no event to
process.

Return an identifier to cancel the scheduling with
after_cancel.
```

bbox = grid_bbox(self, column=None, row=None, col2=None, row2=None)
```
Return a tuple of integer coordinates for the bounding
box of this widget controlled by the geometry manager grid.

If COLUMN, ROW is given the bounding box applies from
the cell with row and column 0 to the specified
cell. If COL2 and ROW2 are given the bounding box
starts at that cell.

The returned integers specify the offset of the upper left
corner in the master widget and the width and height.
```

bell(self, displayof=0)
```
Ring a display's bell.
```

bind(self, sequence=None, func=None, add=None)
```
Bind to this widget at event SEQUENCE a call to function FUNC.

SEQUENCE is a string of concatenated event
patterns. An event pattern is of the form
<MODIFIER-MODIFIER-TYPE-DETAIL> where MODIFIER is one
of Control, Mod2, M2, Shift, Mod3, M3, Lock, Mod4, M4,
Button1, B1, Mod5, M5 Button2, B2, Meta, M, Button3,
```

```
B3, Alt, Button4, B4, Double, Button5, B5 Triple,
Mod1, M1. TYPE is one of Activate, Enter, Map,
ButtonPress, Button, Expose, Motion, ButtonRelease
FocusIn, MouseWheel, Circulate, FocusOut, Property,
Colormap, Gravity Reparent, Configure, KeyPress, Key,
Unmap, Deactivate, KeyRelease Visibility, Destroy,
Leave and DETAIL is the button number for ButtonPress,
ButtonRelease and DETAIL is the Keysym for KeyPress and
KeyRelease. Examples are
<Control-Button-1> for pressing Control and mouse button 1 or
<Alt-A> for pressing A and the Alt key (KeyPress can be omitted).
An event pattern can also be a virtual event of the form
<<AString>> where AString can be arbitrary. This
event can be generated by event_generate.
If events are concatenated they must appear shortly
after each other.

FUNC will be called if the event sequence occurs with an
instance of Event as argument. If the return value of FUNC is
"break" no further bound function is invoked.

An additional boolean parameter ADD specifies whether FUNC will
be called additionally to the other bound function or whether
it will replace the previous function.

Bind will return an identifier to allow deletion of the bound function with
unbind without memory leak.

If FUNC or SEQUENCE is omitted the bound function or list
of bound events are returned.
```

bind_all(self, sequence=None, func=None, add=None)

```
Bind to all widgets at an event SEQUENCE a call to function FUNC.
An additional boolean parameter ADD specifies whether FUNC will
be called additionally to the other bound function or whether
it will replace the previous function. See bind for the return value.
```

bind_class(self, className, sequence=None, func=None, add=None)

```
Bind to widgets with bindtag CLASSNAME at event
SEQUENCE a call of function FUNC. An additional
boolean parameter ADD specifies whether FUNC will be
called additionally to the other bound function or
whether it will replace the previous function. See bind for
the return value.
```

bindtags(self, tagList=None)

```
Set or get the list of bindtags for this widget.

With no argument return the list of all bindtags associated with
this widget. With a list of strings as argument the bindtags are
set to this list. The bindtags determine in which order events are
processed (see bind).
```

cget(self, key)

```
Return the resource value for a KEY given as string.
```

clipboard_append(self, string, **kw)

```
Append STRING to the Tk clipboard.

A widget specified at the optional displayof keyword
argument specifies the target display. The clipboard
can be retrieved with selection_get.
```

clipboard_clear(self, **kw)

```
Clear the data in the Tk clipboard.

A widget specified for the optional displayof keyword
argument specifies the target display.
```

clipboard_get(self, **kw)

```
Retrieve data from the clipboard on window's display.

The window keyword defaults to the root window of the Tkinter
```

```
    application.

    The type keyword specifies the form in which the data is
    to be returned and should be an atom name such as STRING
    or FILE_NAME.  Type defaults to STRING, except on X11, where the default
    is to try UTF8_STRING and fall back to STRING.

    This command is equivalent to:

    selection_get(CLIPBOARD)
```

colormodel(self, value=None)

```
    Useless. Not implemented in Tk.
```

columnconfigure = grid_columnconfigure(self, index, cnf={}, **kw)

```
    Configure column INDEX of a grid.

    Valid resources are minsize (minimum size of the column),
    weight (how much does additional space propagate to this column)
    and pad (how much space to let additionally).
```

config = configure(self, cnf=None, **kw)

```
    Configure resources of a widget.

    The values for resources are specified as keyword
    arguments. To get an overview about
    the allowed keyword arguments call the method keys.
```

configure(self, cnf=None, **kw)

```
    Configure resources of a widget.

    The values for resources are specified as keyword
    arguments. To get an overview about
    the allowed keyword arguments call the method keys.
```

deletecommand(self, name)

Internal function.

Delete the Tcl command provided in NAME.

event_add(self, virtual, *sequences)

Bind a virtual event VIRTUAL (of the form <<Name>>)
to an event SEQUENCE such that the virtual event is triggered
whenever SEQUENCE occurs.

event_delete(self, virtual, *sequences)

Unbind a virtual event VIRTUAL from SEQUENCE.

event_generate(self, sequence, **kw)

Generate an event SEQUENCE. Additional
keyword arguments specify parameter of the event
(e.g. x, y, rootx, rooty).

event_info(self, virtual=None)

Return a list of all virtual events or the information
about the SEQUENCE bound to the virtual event VIRTUAL.

focus = focus_set(self)

Direct input focus to this widget.

If the application currently does not have the focus
this widget will get the focus if the application gets
the focus through the window manager.

focus_displayof(self)

Return the widget which has currently the focus on the
display where this widget is located.

Return None if the application does not have the focus.

focus_force(self)

Direct input focus to this widget even if the

application does not have the focus. Use with
caution!

focus_get(self)
Return the widget which has currently the focus in the
application.

Use focus_displayof to allow working with several
displays. Return None if application does not have
the focus.

focus_lastfor(self)
Return the widget which would have the focus if top level
for this widget gets the focus from the window manager.

focus_set(self)
Direct input focus to this widget.

If the application currently does not have the focus
this widget will get the focus if the application gets
the focus through the window manager.

getboolean(self, s)
Return a boolean value for Tcl boolean values true and false given as parameter.

getvar(self, name='PY_VAR')
Return value of Tcl variable NAME.

grab_current(self)
Return widget which has currently the grab in this application
or None.

grab_release(self)
Release grab for this widget if currently set.

grab_set(self)
```
    Set grab for this widget.

    A grab directs all events to this and descendant
    widgets in the application.
```

grab_set_global(self)
```
    Set global grab for this widget.

    A global grab directs all events to this and
    descendant widgets on the display. Use with caution -
    other applications do not get events anymore.
```

grab_status(self)
```
    Return None, "local" or "global" if this widget has
    no, a local or a global grab.
```

grid_bbox(self, column=None, row=None, col2=None, row2=None)
```
    Return a tuple of integer coordinates for the bounding
    box of this widget controlled by the geometry manager grid.

    If COLUMN, ROW is given the bounding box applies from
    the cell with row and column 0 to the specified
    cell. If COL2 and ROW2 are given the bounding box
    starts at that cell.

    The returned integers specify the offset of the upper left
    corner in the master widget and the width and height.
```

grid_columnconfigure(self, index, cnf={}, **kw)
```
    Configure column INDEX of a grid.

    Valid resources are minsize (minimum size of the column),
    weight (how much does additional space propagate to this column)
    and pad (how much space to let additionally).
```

grid_location(self, x, y)
```
    Return a tuple of column and row which identify the cell
    at which the pixel at position X and Y inside the master
    widget is located.
```

grid_propagate(self, flag=['_noarg_'])
```
    Set or get the status for propagation of geometry information.

    A boolean argument specifies whether the geometry information
    of the slaves will determine the size of this widget. If no argument
    is given, the current setting will be returned.
```

grid_rowconfigure(self, index, cnf={}, **kw)
```
    Configure row INDEX of a grid.

    Valid resources are minsize (minimum size of the row),
    weight (how much does additional space propagate to this row)
    and pad (how much space to let additionally).
```

grid_size(self)
```
    Return a tuple of the number of column and rows in the grid.
```

grid_slaves(self, row=None, column=None)
```
    Return a list of all slaves of this widget
    in its packing order.
```

image_names(self)
```
    Return a list of all existing image names.
```

image_types(self)
```
    Return a list of all available image types (e.g. phote bitmap).
```

keys(self)
```
    Return a list of all resource names of this widget.
```

lift = tkraise(self, aboveThis=None)
```
    Raise this widget in the stacking order.
```

lower(self, belowThis=None)
```
    Lower this widget in the stacking order.
```

mainloop(self, n=0)
```
    Call the mainloop of Tk.
```

nametowidget(self, name)
```
    Return the Tkinter instance of a widget identified by
    its Tcl name NAME.
```

option_add(self, pattern, value, priority=None)
```
    Set a VALUE (second parameter) for an option
    PATTERN (first parameter).

    An optional third parameter gives the numeric priority
    (defaults to 80).
```

option_clear(self)
```
    Clear the option database.

    It will be reloaded if option_add is called.
```

option_get(self, name, className)
```
    Return the value for an option NAME for this widget
    with CLASSNAME.

    Values with higher priority override lower values.
```

option_readfile(self, fileName, priority=None)
```
    Read file FILENAME into the option database.

    An optional second parameter gives the numeric
```

```
    priority.
```

pack_propagate(self, flag=['_noarg_'])
```
    Set or get the status for propagation of geometry information.

    A boolean argument specifies whether the geometry information
    of the slaves will determine the size of this widget. If no argument
    is given the current setting will be returned.
```

pack_slaves(self)
```
    Return a list of all slaves of this widget
    in its packing order.
```

place_slaves(self)
```
    Return a list of all slaves of this widget
    in its packing order.
```

propagate = pack_propagate(self, flag=['_noarg_'])
```
    Set or get the status for propagation of geometry information.

    A boolean argument specifies whether the geometry information
    of the slaves will determine the size of this widget. If no argument
    is given the current setting will be returned.
```

quit(self)
```
    Quit the Tcl interpreter. All widgets will be destroyed.
```

register = _register(self, func, subst=None, needcleanup=1)
```
    Return a newly created Tcl function. If this
    function is called, the Python function FUNC will
    be executed. An optional function SUBST can
    be given which will be executed before FUNC.
```

rowconfigure = grid_rowconfigure(self, index, cnf={}, **kw)
```
    Configure row INDEX of a grid.
```

Valid resources are minsize (minimum size of the row),
weight (how much does additional space propagate to this row)
and pad (how much space to let additionally).

selection_clear(self, **kw)

Clear the current X selection.

selection_get(self, **kw)

Return the contents of the current X selection.

A keyword parameter selection specifies the name of
the selection and defaults to PRIMARY. A keyword
parameter displayof specifies a widget on the display
to use. A keyword parameter type specifies the form of data to be
fetched, defaulting to STRING except on X11, where UTF8_STRING is tried
before STRING.

selection_handle(self, command, **kw)

Specify a function COMMAND to call if the X
selection owned by this widget is queried by another
application.

This function must return the contents of the
selection. The function will be called with the
arguments OFFSET and LENGTH which allows the chunking
of very long selections. The following keyword
parameters can be provided:
selection - name of the selection (default PRIMARY),
type - type of the selection (e.g. STRING, FILE_NAME).

selection_own(self, **kw)

Become owner of X selection.

A keyword parameter selection specifies the name of
the selection (default PRIMARY).

selection_own_get(self, **kw)
```
    Return owner of X selection.

    The following keyword parameter can
    be provided:
    selection - name of the selection (default PRIMARY),
    type - type of the selection (e.g. STRING, FILE_NAME).
```

send(self, interp, cmd, *args)
```
    Send Tcl command CMD to different interpreter INTERP to be executed.
```

setvar(self, name='PY_VAR', value='1')
```
    Set Tcl variable NAME to VALUE.
```

size = grid_size(self)
```
    Return a tuple of the number of column and rows in the grid.
```

slaves = pack_slaves(self)
```
    Return a list of all slaves of this widget
    in its packing order.
```

tk_bisque(self)
```
    Change the color scheme to light brown as used in Tk 3.6 and before.
```

tk_focusFollowsMouse(self)
```
    The widget under mouse will get automatically focus. Can not
    be disabled easily.
```

tk_focusNext(self)
```
    Return the next widget in the focus order which follows
    widget which has currently the focus.

    The focus order first goes to the next child, then to
    the children of the child recursively and then to the
    next sibling which is higher in the stacking order.  A
```

widget is omitted if it has the takefocus resource set
to 0.

tk_focusPrev(self)
Return previous widget in the focus order. See tk_focusNext for details.

tk_menuBar(self, *args)
Do not use. Needed in Tk 3.6 and earlier.

tk_setPalette(self, *args, **kw)
Set a new color scheme for all widget elements.

A single color as argument will cause that all colors of Tk
widget elements are derived from this.
Alternatively several keyword parameters and its associated
colors can be given. The following keywords are valid:
activeBackground, foreground, selectColor,
activeForeground, highlightBackground, selectBackground,
background, highlightColor, selectForeground,
disabledForeground, insertBackground, troughColor.

tk_strictMotif(self, boolean=None)
Set Tcl internal variable, whether the look and feel
should adhere to Motif.

A parameter of 1 means adhere to Motif (e.g. no color
change if mouse passes over slider).
Returns the set value.

tkraise(self, aboveThis=None)
Raise this widget in the stacking order.

unbind(self, sequence, funcid=None)
Unbind for this widget for event SEQUENCE the
function identified with FUNCID.

unbind_all(self, sequence)
```
    Unbind for all widgets for event SEQUENCE all functions.
```

unbind_class(self, className, sequence)
```
    Unbind for a all widgets with bindtag CLASSNAME for event SEQUENCE
    all functions.
```

update(self)
```
    Enter event loop until all pending events have been processed by Tcl.
```

update_idletasks(self)
```
    Enter event loop until all idle callbacks have been called. This
    will update the display of windows but not process events caused by
    the user.
```

wait_variable(self, name='PY_VAR')
```
    Wait until the variable is modified.

    A parameter of type IntVar, StringVar, DoubleVar or
    BooleanVar must be given.
```

wait_visibility(self, window=None)
```
    Wait until the visibility of a WIDGET changes
    (e.g. it appears).

    If no parameter is given self is used.
```

wait_window(self, window=None)
```
    Wait until a WIDGET is destroyed.

    If no parameter is given self is used.
```

waitvar = wait_variable(self, name='PY_VAR')
```
    Wait until the variable is modified.
```

A parameter of type IntVar, StringVar, DoubleVar or
BooleanVar must be given.

winfo_atom(self, name, displayof=0)
Return integer which represents atom NAME.

winfo_atomname(self, id, displayof=0)
Return name of atom with identifier ID.

winfo_cells(self)
Return number of cells in the colormap for this widget.

winfo_children(self)
Return a list of all widgets which are children of this widget.

winfo_class(self)
Return window class name of this widget.

winfo_colormapfull(self)
Return true if at the last color request the colormap was full.

winfo_containing(self, rootX, rootY, displayof=0)
Return the widget which is at the root coordinates ROOTX, ROOTY.

winfo_depth(self)
Return the number of bits per pixel.

winfo_exists(self)
Return true if this widget exists.

winfo_fpixels(self, number)
Return the number of pixels for the given distance NUMBER
(e.g. "3c") as float.

winfo_geometry(self)
    Return geometry string for this widget in the form "widthxheight+X+Y".

winfo_height(self)
    Return height of this widget.

winfo_id(self)
    Return identifier ID for this widget.

winfo_interps(self, displayof=0)
    Return the name of all Tcl interpreters for this display.

winfo_ismapped(self)
    Return true if this widget is mapped.

winfo_manager(self)
    Return the window mananger name for this widget.

winfo_name(self)
    Return the name of this widget.

winfo_parent(self)
    Return the name of the parent of this widget.

winfo_pathname(self, id, displayof=0)
    Return the pathname of the widget given by ID.

winfo_pixels(self, number)
    Rounded integer value of winfo_fpixels.

winfo_pointerx(self)
    Return the x coordinate of the pointer on the root window.

winfo_pointerxy(self)

Return a tuple of x and y coordinates of the pointer on the root window.

winfo_pointery(self)
Return the y coordinate of the pointer on the root window.

winfo_reqheight(self)
Return requested height of this widget.

winfo_reqwidth(self)
Return requested width of this widget.

winfo_rgb(self, color)
Return tuple of decimal values for red, green, blue for
COLOR in this widget.

winfo_rootx(self)
Return x coordinate of upper left corner of this widget on the
root window.

winfo_rooty(self)
Return y coordinate of upper left corner of this widget on the
root window.

winfo_screen(self)
Return the screen name of this widget.

winfo_screencells(self)
Return the number of the cells in the colormap of the screen
of this widget.

winfo_screendepth(self)
Return the number of bits per pixel of the root window of the
screen of this widget.

winfo_screenheight(self)

Return the number of pixels of the height of the screen of this widget
in pixel.

winfo_screenmmheight(self)
Return the number of pixels of the height of the screen of
this widget in mm.

winfo_screenmmwidth(self)
Return the number of pixels of the width of the screen of
this widget in mm.

winfo_screenvisual(self)
Return one of the strings directcolor, grayscale, pseudocolor,
staticcolor, staticgray, or truecolor for the default
colormodel of this screen.

winfo_screenwidth(self)
Return the number of pixels of the width of the screen of
this widget in pixel.

winfo_server(self)
Return information of the X-Server of the screen of this widget in
the form "XmajorRminor vendor vendorVersion".

winfo_toplevel(self)
Return the toplevel widget of this widget.

winfo_viewable(self)
Return true if the widget and all its higher ancestors are mapped.

winfo_visual(self)
Return one of the strings directcolor, grayscale, pseudocolor,
staticcolor, staticgray, or truecolor for the
colormodel of this widget.

winfo_visualid(self)
```
Return the X identifier for the visual for this widget.
```

winfo_visualsavailable(self, includeids=0)
```
Return a list of all visuals available for the screen
of this widget.

Each item in the list consists of a visual name (see winfo_visual), a
depth and if INCLUDEIDS=1 is given also the X identifier.
```

winfo_vrootheight(self)
```
Return the height of the virtual root window associated with this
widget in pixels. If there is no virtual root window return the
height of the screen.
```

winfo_vrootwidth(self)
```
Return the width of the virtual root window associated with this
widget in pixel. If there is no virtual root window return the
width of the screen.
```

winfo_vrootx(self)
```
Return the x offset of the virtual root relative to the root
window of the screen of this widget.
```

winfo_vrooty(self)
```
Return the y offset of the virtual root relative to the root
window of the screen of this widget.
```

winfo_width(self)
```
Return the width of this widget.
```

winfo_x(self)
```
Return the x coordinate of the upper left corner of this widget
in the parent.
```

winfo_y(self)
```
Return the y coordinate of the upper left corner of this widget
in the parent.
```

Data and other attributes inherited from Tkinter.Misc:

getdouble = <type 'float'>
```
float(x) -> floating point number

Convert a string or number to a floating point number, if possible.
```

getint = <type 'int'>
```
int(x[, base]) -> integer

Convert a string or number to an integer, if possible.  A floating point
argument will be truncated towards zero (this does not include a string
representation of a floating point number!)  When converting a string, use
the optional base.  It is an error to supply a base when converting a
non-string.  If base is zero, the proper base is guessed based on the
string content.  If the argument is outside the integer range a
long object will be returned instead.
```

Methods inherited from Tkinter.Wm:

aspect = wm_aspect(self, minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)
```
Instruct the window manager to set the aspect ratio (width/height)
of this widget to be between MINNUMER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple
of the actual values if no argument is given.
```

attributes = wm_attributes(self, *args)
```
This subcommand returns or sets platform specific attributes

The first form returns a list of the platform specific flags and
their values. The second form returns the value for the specific
```

option. The third form sets one or more of the values. The values
are as follows:

On Windows, -disabled gets or sets whether the window is in a
disabled state. -toolwindow gets or sets the style of the window
to toolwindow (as defined in the MSDN). -topmost gets or sets
whether this is a topmost window (displays above all other
windows).

On Macintosh, XXXXX

On Unix, there are currently no special attribute values.

client = wm_client(self, name=None)

Store NAME in WM_CLIENT_MACHINE property of this widget. Return
current value.

colormapwindows = wm_colormapwindows(self, *wlist)

Store list of window names (WLIST) into WM_COLORMAPWINDOWS property
of this widget. This list contains windows whose colormaps differ from their
parents. Return current list of widgets if WLIST is empty.

command = wm_command(self, value=None)

Store VALUE in WM_COMMAND property. It is the command
which shall be used to invoke the application. Return current
command if VALUE is None.

deiconify = wm_deiconify(self)

Deiconify this widget. If it was never mapped it will not be mapped.
On Windows it will raise this widget and give it the focus.

focusmodel = wm_focusmodel(self, model=None)

Set focus model to MODEL. "active" means that this widget will claim
the focus itself, "passive" means that the window manager shall give
the focus. Return current focus model if MODEL is None.

frame = wm_frame(self)
```
Return identifier for decorative frame of this widget if present.
```

geometry = wm_geometry(self, newGeometry=None)
```
Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return
current value if None is given.
```

grid = wm_grid(self, baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)
```
Instruct the window manager that this widget shall only be
resized on grid boundaries. WIDTHINC and HEIGHTINC are the width and
height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the
number of grid units requested in Tk_GeometryRequest.
```

group = wm_group(self, pathName=None)
```
Set the group leader widgets for related widgets to PATHNAME. Return
the group leader of this widget if None is given.
```

iconbitmap = wm_iconbitmap(self, bitmap=None, default=None)
```
Set bitmap for the iconified widget to BITMAP. Return
the bitmap if None is given.

Under Windows, the DEFAULT parameter can be used to set the icon
for the widget and any descendents that don't have an icon set
explicitly.  DEFAULT can be the relative path to a .ico file
(example: root.iconbitmap(default='myicon.ico') ).  See Tk
documentation for more information.
```

iconify = wm_iconify(self)
```
Display widget as icon.
```

iconmask = wm_iconmask(self, bitmap=None)
```
Set mask for the icon bitmap of this widget. Return the
mask if None is given.
```

iconname = wm_iconname(self, newName=None)

```
    Set the name of the icon for this widget. Return the name if
    None is given.
```

iconposition = wm_iconposition(self, x=None, y=None)

```
    Set the position of the icon of this widget to X and Y. Return
    a tuple of the current values of X and X if None is given.
```

iconwindow = wm_iconwindow(self, pathName=None)

```
    Set widget PATHNAME to be displayed instead of icon. Return the current
    value if None is given.
```

maxsize = wm_maxsize(self, width=None, height=None)

```
    Set max WIDTH and HEIGHT for this widget. If the window is gridded
    the values are given in grid units. Return the current values if None
    is given.
```

minsize = wm_minsize(self, width=None, height=None)

```
    Set min WIDTH and HEIGHT for this widget. If the window is gridded
    the values are given in grid units. Return the current values if None
    is given.
```

overrideredirect = wm_overrideredirect(self, boolean=None)

```
    Instruct the window manager to ignore this widget
    if BOOLEAN is given with 1. Return the current value if None
    is given.
```

positionfrom = wm_positionfrom(self, who=None)

```
    Instruct the window manager that the position of this widget shall
    be defined by the user if WHO is "user", and by its own policy if WHO is
    "program".
```

protocol = wm_protocol(self, name=None, func=None)

```
    Bind function FUNC to command NAME for this widget.
    Return the function bound to NAME if None is given. NAME could be
    e.g. "WM_SAVE_YOURSELF" or "WM_DELETE_WINDOW".
```

resizable = wm_resizable(self, width=None, height=None)

```
Instruct the window manager whether this width can be resized
in WIDTH or HEIGHT. Both values are boolean values.
```

sizefrom = wm_sizefrom(self, who=None)

```
Instruct the window manager that the size of this widget shall
be defined by the user if WHO is "user", and by its own policy if WHO is
"program".
```

state = wm_state(self, newstate=None)

```
Query or set the state of this widget as one of normal, icon,
iconic (see wm_iconwindow), withdrawn, or zoomed (Windows only).
```

title = wm_title(self, string=None)

```
Set the title of this widget.
```

transient = wm_transient(self, master=None)

```
Instruct the window manager that this widget is transient
with regard to widget MASTER.
```

withdraw = wm_withdraw(self)

```
Withdraw this widget from the screen such that it is unmapped
and forgotten by the window manager. Re-draw it with wm_deiconify.
```

wm_aspect(self, minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)

```
Instruct the window manager to set the aspect ratio (width/height)
of this widget to be between MINNUMER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple
of the actual values if no argument is given.
```

wm_attributes(self, *args)

```
This subcommand returns or sets platform specific attributes

The first form returns a list of the platform specific flags and
their values. The second form returns the value for the specific
option. The third form sets one or more of the values. The values
```

are as follows:

On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a topmost window (displays above all other windows).

On Macintosh, XXXXX

On Unix, there are currently no special attribute values.

wm_client(self, name=None)

Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.

wm_colormapwindows(self, *wlist)

Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

wm_command(self, value=None)

Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the application. Return current command if VALUE is None.

wm_deiconify(self)

Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget and give it the focus.

wm_focusmodel(self, model=None)

Set focus model to MODEL. "active" means that this widget will claim the focus itself, "passive" means that the window manager shall give the focus. Return current focus model if MODEL is None.

wm_frame(self)

Return identifier for decorative frame of this widget if present.

wm_geometry(self, newGeometry=None)
Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return
current value if None is given.

wm_grid(self, baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)
Instruct the window manager that this widget shall only be
resized on grid boundaries. WIDTHINC and HEIGHTINC are the width and
height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the
number of grid units requested in Tk_GeometryRequest.

wm_group(self, pathName=None)
Set the group leader widgets for related widgets to PATHNAME. Return
the group leader of this widget if None is given.

wm_iconbitmap(self, bitmap=None, default=None)
Set bitmap for the iconified widget to BITMAP. Return
the bitmap if None is given.

Under Windows, the DEFAULT parameter can be used to set the icon
for the widget and any descendents that don't have an icon set
explicitly. DEFAULT can be the relative path to a .ico file
(example: root.iconbitmap(default='myicon.ico')). See Tk
documentation for more information.

wm_iconify(self)
Display widget as icon.

wm_iconmask(self, bitmap=None)
Set mask for the icon bitmap of this widget. Return the
mask if None is given.

wm_iconname(self, newName=None)
Set the name of the icon for this widget. Return the name if
None is given.

wm_iconposition(self, x=None, y=None)

```
Set the position of the icon of this widget to X and Y. Return
a tuple of the current values of X and X if None is given.
```

wm_iconwindow(self, pathName=None)

```
Set widget PATHNAME to be displayed instead of icon. Return the current
value if None is given.
```

wm_maxsize(self, width=None, height=None)

```
Set max WIDTH and HEIGHT for this widget. If the window is gridded
the values are given in grid units. Return the current values if None
is given.
```

wm_minsize(self, width=None, height=None)

```
Set min WIDTH and HEIGHT for this widget. If the window is gridded
the values are given in grid units. Return the current values if None
is given.
```

wm_overrideredirect(self, boolean=None)

```
Instruct the window manager to ignore this widget
if BOOLEAN is given with 1. Return the current value if None
is given.
```

wm_positionfrom(self, who=None)

```
Instruct the window manager that the position of this widget shall
be defined by the user if WHO is "user", and by its own policy if WHO is
"program".
```

wm_protocol(self, name=None, func=None)

```
Bind function FUNC to command NAME for this widget.
Return the function bound to NAME if None is given. NAME could be
e.g. "WM_SAVE_YOURSELF" or "WM_DELETE_WINDOW".
```

wm_resizable(self, width=None, height=None)

Instruct the window manager whether this width can be resized
in WIDTH or HEIGHT. Both values are boolean values.

wm_sizefrom(self, who=None)
Instruct the window manager that the size of this widget shall
be defined by the user if WHO is "user", and by its own policy if WHO is
"program".

wm_state(self, newstate=None)
Query or set the state of this widget as one of normal, icon,
iconic (see wm_iconwindow), withdrawn, or zoomed (Windows only).

wm_title(self, string=None)
Set the title of this widget.

wm_transient(self, master=None)
Instruct the window manager that this widget is transient
with regard to widget MASTER.

wm_withdraw(self)
Withdraw this widget from the screen such that it is unmapped
and forgotten by the window manager. Re-draw it with wm_deiconify.

Data

ACTIVE = 'active'
ALL = 'all'
ANCHOR = 'anchor'
ARC = 'arc'
Area = <Tkinter.Canvas instance>
BASELINE = 'baseline'

BEVEL = 'bevel'
BOTH = 'both'
BOTTOM = 'bottom'
BROWSE = 'browse'
BUTT = 'butt'
B_info = <Tkinter.Button instance>
B_lan = <Tkinter.Button instance>
B_mapa = <Tkinter.Button instance>
B_nmap = <Tkinter.Button instance>
CASCADE = 'cascade'
CENTER = 'center'
CHAR = 'char'
CHECKBUTTON = 'checkbutton'
CHORD = 'chord'
COMMAND = 'command'
CURRENT = 'current'
DISABLED = 'disabled'
DOTBOX = 'dotbox'
E = 'e'
END = 'end'
EW = 'ew'
EXCEPTION = 8
EXTENDED = 'extended'
FALSE = 0
FIRST = 'first'
FLAT = 'flat'
GROOVE = 'groove'
HIDDEN = 'hidden'
HORIZONTAL = 'horizontal'
INSERT = 'insert'

INSIDE = 'inside'
LAST = 'last'
LEFT = 'left'
MITER = 'miter'
MOVETO = 'moveto'
MULTIPLE = 'multiple'
N = 'n'
NE = 'ne'
NO = 0
NONE = 'none'
NORMAL = 'normal'
NS = 'ns'
NSEW = 'nsew'
NUMERIC = 'numeric'
NW = 'nw'
OFF = 0
ON = 1
OUTSIDE = 'outside'
PAGES = 'pages'
PIESLICE = 'pieslice'
PROJECTING = 'projecting'
RADIOBUTTON = 'radiobutton'
RAISED = 'raised'
READABLE = 2
RIDGE = 'ridge'
RIGHT = 'right'
ROUND = 'round'
S = 's'
SCROLL = 'scroll'
SE = 'se'

SEL = 'sel'
SEL_FIRST = 'sel.first'
SEL_LAST = 'sel.last'
SEPARATOR = 'separator'
SINGLE = 'single'
SOLID = 'solid'
SUNKEN = 'sunken'
SW = 'sw'
StringTypes = (<type 'str'>, <type 'unicode'>)
TOP = 'top'
TRUE = 1
TclVersion = 8.5
TkVersion = 8.5
UNDERLINE = 'underline'
UNITS = 'units'
VERTICAL = 'vertical'
W = 'w'
WORD = 'word'
WRITABLE = 4
X = 'x'
Y = 'y'
YES = 1
top = <Tkinter.Tk instance>
wantobjects = 1