注: 文中misc.h 与 misc.c 是删减版.
来源于https://github.com/raspberrypi-ui/lxpanel/tree/master/src

一.检测远程桌面连接方法如下:

```
#include "misc.h"

gboolean check_connect()
{
   gchar *client_name = NULL;
   gint client_count = 0;
   gint i = 0;

   Window *client_list = get_xaproperty(GDK_ROOT_WINDOW(),   //获得桌面所有窗口列表
                  a_NET_CLIENT_LIST,
                  XA_WINDOW,
                  &client_count);
   if (client_list != NULL)
   {
      for (i = 0; i < client_count; i++)
      {
         client_name = get_utf8_property(client_list[i],   //获得窗口的名字
                  a_NET_WM_VISIBLE_NAME);
         if (strcmp(client_name, HDP_CLIENT_NAME) == 0)   //与要检测的窗口连接名比较, 有相等的则证明已连接
         {
            return TRUE;
         }
      }
   }
   return FALSE;
}
```

二.添加头文件misc.h

代码如下

```
/**
 * Copyright (c) 2006-2014 LxDE Developers, see the file AUTHORS for details.
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
 */

#ifndef __MISC_H__
#define __MISC_H__ 1
```

```
#include <X11/Xatom.h>
#include <gtk/gtk.h>
#include <gdk/gdk.h>
#include <gdk/gdkx.h>
#include <stdio.h>


G_BEGIN_DECLS

extern Atom a_UTF8_STRING;
extern Atom a_XROOTPMAP_ID;

extern Atom a_WM_STATE;
extern Atom a_WM_CLASS;
extern Atom a_WM_DELETE_WINDOW;
extern Atom a_WM_PROTOCOLS;
extern Atom a_NET_WORKAREA;
extern Atom a_NET_CLIENT_LIST;
extern Atom a_NET_CLIENT_LIST_STACKING;
extern Atom a_NET_NUMBER_OF_DESKTOPS;
extern Atom a_NET_CURRENT_DESKTOP;
extern Atom a_NET_DESKTOP_VIEWPORT;
extern Atom a_NET_DESKTOP_NAMES;
extern Atom a_NET_ACTIVE_WINDOW;
extern Atom a_NET_CLOSE_WINDOW;
extern Atom a_NET_SHOWING_DESKTOP;
extern Atom a_NET_SUPPORTED;
extern Atom a_NET_WM_STATE;
extern Atom a_NET_WM_STATE_SKIP_TASKBAR;
extern Atom a_NET_WM_STATE_SKIP_PAGER;
extern Atom a_NET_WM_STATE_STICKY;
extern Atom a_NET_WM_STATE_HIDDEN;
extern Atom a_NET_WM_STATE_SHADED;

#define a_NET_WM_STATE_REMOVE        0    /* remove/unset property */
#define a_NET_WM_STATE_ADD           1    /* add/set property */
#define a_NET_WM_STATE_TOGGLE        2    /* toggle property  */

extern Atom a_NET_WM_WINDOW_TYPE;
extern Atom a_NET_WM_WINDOW_TYPE_DESKTOP;
extern Atom a_NET_WM_WINDOW_TYPE_DOCK;
extern Atom a_NET_WM_WINDOW_TYPE_TOOLBAR;
extern Atom a_NET_WM_WINDOW_TYPE_MENU;
extern Atom a_NET_WM_WINDOW_TYPE_UTILITY;
extern Atom a_NET_WM_WINDOW_TYPE_SPLASH;
extern Atom a_NET_WM_WINDOW_TYPE_DIALOG;
extern Atom a_NET_WM_WINDOW_TYPE_NORMAL;

extern Atom a_NET_WM_DESKTOP;
extern Atom a_NET_WM_NAME;
extern Atom a_NET_WM_VISIBLE_NAME;
extern Atom a_NET_WM_STRUT;
extern Atom a_NET_WM_STRUT_PARTIAL;
extern Atom a_NET_WM_ICON;
extern Atom a_KDE_NET_WM_SYSTEM_TRAY_WINDOW_FOR;

extern Atom a_NET_SYSTEM_TRAY_OPCODE;
extern Atom a_NET_SYSTEM_TRAY_MESSAGE_DATA;
extern Atom a_NET_SYSTEM_TRAY_ORIENTATION;
extern Atom a_MANAGER;

extern Atom a_LXPANEL_CMD; /* for private client message */

/* Decoded value of WM_STATE property. */
```

```
typedef struct {
    unsigned int modal : 1;
    unsigned int sticky : 1;
    unsigned int maximized_vert : 1;
    unsigned int maximized_horz : 1;
    unsigned int shaded : 1;
    unsigned int skip_taskbar : 1;
    unsigned int skip_pager : 1;
    unsigned int hidden : 1;
    unsigned int fullscreen : 1;
    unsigned int above : 1;
    unsigned int below : 1;
} NetWMState;

/* Decoded value of _NET_WM_WINDOW_TYPE property. */
typedef struct {
    unsigned int desktop : 1;
    unsigned int dock : 1;
    unsigned int toolbar : 1;
    unsigned int menu : 1;
    unsigned int utility : 1;
    unsigned int splash : 1;
    unsigned int dialog : 1;
    unsigned int normal : 1;
} NetWMWindowType;

void Xclimsg(Window win, Atom type, long l0, long l1, long l2, long l3, long l4);
void Xclimsgwm(Window win, Atom type, Atom arg);
void *get_xaproperty (Window win, Atom prop, Atom type, int *nitems);
char *get_textproperty(Window win, Atom prop);
void *get_utf8_property(Window win, Atom atom);
char **get_utf8_property_list(Window win, Atom atom, int *count);

void resolve_atoms();
//Window Select_Window(Display *dpy);
int get_net_number_of_desktops();
int get_net_current_desktop ();
int get_net_wm_desktop(Window win);
int get_wm_state (Window win);
void get_net_wm_state(Window win, NetWMState *nws);
void get_net_wm_window_type(Window win, NetWMWindowType *nwwt);
GPid get_net_wm_pid(Window win);

G_END_DECLS

#endif
```

三.添加misc.c文件

代码如下:

```
/**
 * Copyright (c) 2006-2014 LxDE Developers, see the file AUTHORS for details.
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software Foundation,
```

```
 * Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
 */

#include <X11/Xatom.h>
#include <X11/cursorfont.h>

#include <gtk/gtk.h>
#include <gdk/gdk.h>
#include <gdk/gdkx.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <stdarg.h>

#include "misc.h"

/* X11 data types */
Atom a_UTF8_STRING;
Atom a_XROOTPMAP_ID;

/* old WM spec */
Atom a_WM_STATE;
Atom a_WM_CLASS;
Atom a_WM_DELETE_WINDOW;
Atom a_WM_PROTOCOLS;

/* new NET spec */
Atom a_NET_WORKAREA;
Atom a_NET_CLIENT_LIST;
Atom a_NET_CLIENT_LIST_STACKING;
Atom a_NET_NUMBER_OF_DESKTOPS;
Atom a_NET_CURRENT_DESKTOP;
Atom a_NET_DESKTOP_VIEWPORT;
Atom a_NET_DESKTOP_NAMES;
Atom a_NET_ACTIVE_WINDOW;
Atom a_NET_CLOSE_WINDOW;
Atom a_NET_SHOWING_DESKTOP;
Atom a_NET_SUPPORTED;
Atom a_NET_WM_STATE;
Atom a_NET_WM_STATE_SKIP_TASKBAR;
Atom a_NET_WM_STATE_SKIP_PAGER;
Atom a_NET_WM_STATE_STICKY;
Atom a_NET_WM_STATE_HIDDEN;
Atom a_NET_WM_STATE_SHADED;
Atom a_NET_WM_WINDOW_TYPE;
Atom a_NET_WM_WINDOW_TYPE_DESKTOP;
Atom a_NET_WM_WINDOW_TYPE_DOCK;
Atom a_NET_WM_WINDOW_TYPE_TOOLBAR;
Atom a_NET_WM_WINDOW_TYPE_MENU;
Atom a_NET_WM_WINDOW_TYPE_UTILITY;
Atom a_NET_WM_WINDOW_TYPE_SPLASH;
Atom a_NET_WM_WINDOW_TYPE_DIALOG;
Atom a_NET_WM_WINDOW_TYPE_NORMAL;
Atom a_NET_WM_DESKTOP;
Atom a_NET_WM_PID;
Atom a_NET_WM_NAME;
Atom a_NET_WM_VISIBLE_NAME;
Atom a_NET_WM_STRUT;
Atom a_NET_WM_STRUT_PARTIAL;
Atom a_NET_WM_ICON;
Atom a_KDE_NET_WM_SYSTEM_TRAY_WINDOW_FOR;

/* SYSTEM TRAY spec */
```

```
Atom a_NET_SYSTEM_TRAY_OPCODE;
Atom a_NET_SYSTEM_TRAY_MESSAGE_DATA;
Atom a_NET_SYSTEM_TRAY_ORIENTATION;
Atom a_MANAGER;

Atom a_LXPANEL_CMD; /* for private client message */

/* if current window manager is EWMH conforming. */

enum{
    I_UTF8_STRING,
    I_XROOTPMAP_ID,
    I_WM_STATE,
    I_WM_CLASS,
    I_WM_DELETE_WINDOW,
    I_WM_PROTOCOLS,

    I_NET_WORKAREA,
    I_NET_CLIENT_LIST,
    I_NET_CLIENT_LIST_STACKING,
    I_NET_NUMBER_OF_DESKTOPS,
    I_NET_CURRENT_DESKTOP,
    I_NET_DESKTOP_VIEWPORT,
    I_NET_DESKTOP_NAMES,
    I_NET_ACTIVE_WINDOW,
    I_NET_SHOWING_DESKTOP,
    I_NET_SUPPORTED,

    I_NET_WM_STATE,
    I_NET_WM_STATE_SKIP_TASKBAR,
    I_NET_WM_STATE_SKIP_PAGER,
    I_NET_WM_STATE_STICKY,
    I_NET_WM_STATE_HIDDEN,
    I_NET_WM_STATE_SHADED,
    I_NET_WM_WINDOW_TYPE,

    I_NET_WM_WINDOW_TYPE_DESKTOP,
    I_NET_WM_WINDOW_TYPE_DOCK,
    I_NET_WM_WINDOW_TYPE_TOOLBAR,
    I_NET_WM_WINDOW_TYPE_MENU,
    I_NET_WM_WINDOW_TYPE_UTILITY,
    I_NET_WM_WINDOW_TYPE_SPLASH,
    I_NET_WM_WINDOW_TYPE_DIALOG,
    I_NET_WM_WINDOW_TYPE_NORMAL,
    I_NET_WM_DESKTOP,
    I_NET_WM_PID,
    I_NET_WM_NAME,
    I_NET_WM_VISIBLE_NAME,
    I_NET_WM_STRUT,
    I_NET_WM_STRUT_PARTIAL,
    I_NET_WM_ICON,
    I_KDE_NET_WM_SYSTEM_TRAY_WINDOW_FOR,

    I_NET_SYSTEM_TRAY_OPCODE,
    I_NET_SYSTEM_TRAY_MESSAGE_DATA,
    I_NET_SYSTEM_TRAY_ORIENTATION,
    I_MANAGER,

    I_LXPANEL_CMD,
    N_ATOMS
};

void resolve_atoms()
{
```

```
static const char* atom_names[ N_ATOMS ];

atom_names[ I_UTF8_STRING ] = "UTF8_STRING";
atom_names[ I_XROOTPMAP_ID ] = "_XROOTPMAP_ID";
atom_names[ I_WM_STATE ] = "WM_STATE";
atom_names[ I_WM_CLASS ] = "WM_CLASS";
atom_names[ I_WM_DELETE_WINDOW ] = "WM_DELETE_WINDOW";
atom_names[ I_WM_PROTOCOLS ] = "WM_PROTOCOLS";
atom_names[ I_NET_WORKAREA ] = "_NET_WORKAREA";
atom_names[ I_NET_CLIENT_LIST ] = "_NET_CLIENT_LIST";
atom_names[ I_NET_CLIENT_LIST_STACKING ] = "_NET_CLIENT_LIST_STACKING";
atom_names[ I_NET_NUMBER_OF_DESKTOPS ] = "_NET_NUMBER_OF_DESKTOPS";
atom_names[ I_NET_CURRENT_DESKTOP ] = "_NET_CURRENT_DESKTOP";
atom_names[ I_NET_DESKTOP_VIEWPORT ] = "_NET_DESKTOP_VIEWPORT";
atom_names[ I_NET_DESKTOP_NAMES ] = "_NET_DESKTOP_NAMES";
atom_names[ I_NET_ACTIVE_WINDOW ] = "_NET_ACTIVE_WINDOW";
atom_names[ I_NET_SHOWING_DESKTOP ] = "_NET_SHOWING_DESKTOP";
atom_names[ I_NET_SUPPORTED ] = "_NET_SUPPORTED";
atom_names[ I_NET_WM_DESKTOP ] = "_NET_WM_DESKTOP";
atom_names[ I_NET_WM_STATE ] = "_NET_WM_STATE";
atom_names[ I_NET_WM_STATE_SKIP_TASKBAR ] = "_NET_WM_STATE_SKIP_TASKBAR";
atom_names[ I_NET_WM_STATE_SKIP_PAGER ] = "_NET_WM_STATE_SKIP_PAGER";
atom_names[ I_NET_WM_STATE_STICKY ] = "_NET_WM_STATE_STICKY";
atom_names[ I_NET_WM_STATE_HIDDEN ] = "_NET_WM_STATE_HIDDEN";
atom_names[ I_NET_WM_STATE_SHADED ] = "_NET_WM_STATE_SHADED";
atom_names[ I_NET_WM_WINDOW_TYPE ] = "_NET_WM_WINDOW_TYPE";

atom_names[ I_NET_WM_WINDOW_TYPE_DESKTOP ] = "_NET_WM_WINDOW_TYPE_DESKTOP";
atom_names[ I_NET_WM_WINDOW_TYPE_DOCK ] = "_NET_WM_WINDOW_TYPE_DOCK";
atom_names[ I_NET_WM_WINDOW_TYPE_TOOLBAR ] = "_NET_WM_WINDOW_TYPE_TOOLBAR";
atom_names[ I_NET_WM_WINDOW_TYPE_MENU ] = "_NET_WM_WINDOW_TYPE_MENU";
atom_names[ I_NET_WM_WINDOW_TYPE_UTILITY ] = "_NET_WM_WINDOW_TYPE_UTILITY";
atom_names[ I_NET_WM_WINDOW_TYPE_SPLASH ] = "_NET_WM_WINDOW_TYPE_SPLASH";
atom_names[ I_NET_WM_WINDOW_TYPE_DIALOG ] = "_NET_WM_WINDOW_TYPE_DIALOG";
atom_names[ I_NET_WM_WINDOW_TYPE_NORMAL ] = "_NET_WM_WINDOW_TYPE_NORMAL";
atom_names[ I_NET_WM_DESKTOP ] = "_NET_WM_DESKTOP";
atom_names[ I_NET_WM_PID ] = "_NET_WM_PID";
atom_names[ I_NET_WM_NAME ] = "_NET_WM_NAME";
atom_names[ I_NET_WM_VISIBLE_NAME ] = "_NET_WM_VISIBLE_NAME";
atom_names[ I_NET_WM_STRUT ] = "_NET_WM_STRUT";
atom_names[ I_NET_WM_STRUT_PARTIAL ] = "_NET_WM_STRUT_PARTIAL";
atom_names[ I_NET_WM_ICON ] = "_NET_WM_ICON";
atom_names[ I_KDE_NET_WM_SYSTEM_TRAY_WINDOW_FOR ] = "_KDE_NET_WM_SYSTEM_TRAY_WINDOW_FOR";

atom_names[ I_NET_SYSTEM_TRAY_OPCODE ] = "_NET_SYSTEM_TRAY_OPCODE";
atom_names[ I_NET_SYSTEM_TRAY_MESSAGE_DATA ] = "_NET_SYSTEM_TRAY_MESSAGE_DATA";
atom_names[ I_NET_SYSTEM_TRAY_ORIENTATION ] = "_NET_SYSTEM_TRAY_ORIENTATION";
atom_names[ I_MANAGER ] = "MANAGER";

atom_names[ I_LXPANEL_CMD ] = "_LXPANEL_CMD";

Atom atoms[ N_ATOMS ];


if( ! XInternAtoms( GDK_DISPLAY_XDISPLAY(gdk_display_get_default()), (char**)atom_names,
      N_ATOMS, False, atoms ) )
{
   g_warning( "Error: unable to return Atoms" );
   return;
}

a_UTF8_STRING = atoms[ I_UTF8_STRING ];
a_XROOTPMAP_ID = atoms[ I_XROOTPMAP_ID ];
a_WM_STATE = atoms[ I_WM_STATE ];
```

```
    a_WM_CLASS = atoms[ I_WM_CLASS ];
    a_WM_DELETE_WINDOW = atoms[ I_WM_DELETE_WINDOW ];
    a_WM_PROTOCOLS = atoms[ I_WM_PROTOCOLS ];

    a_NET_WORKAREA = atoms[ I_NET_WORKAREA ];
    a_NET_CLIENT_LIST = atoms[ I_NET_CLIENT_LIST ];
    a_NET_CLIENT_LIST_STACKING = atoms[ I_NET_CLIENT_LIST_STACKING ];
    a_NET_NUMBER_OF_DESKTOPS = atoms[ I_NET_NUMBER_OF_DESKTOPS ];
    a_NET_CURRENT_DESKTOP = atoms[ I_NET_CURRENT_DESKTOP ];
    a_NET_DESKTOP_VIEWPORT = atoms[ I_NET_DESKTOP_VIEWPORT ];
    a_NET_DESKTOP_NAMES = atoms[ I_NET_DESKTOP_NAMES ];
    a_NET_ACTIVE_WINDOW = atoms[ I_NET_ACTIVE_WINDOW ];
    a_NET_SHOWING_DESKTOP = atoms[ I_NET_SHOWING_DESKTOP ];
    a_NET_SUPPORTED = atoms[ I_NET_SUPPORTED ];
    a_NET_WM_STATE = atoms[ I_NET_WM_STATE ];
    a_NET_WM_STATE_SKIP_TASKBAR = atoms[ I_NET_WM_STATE_SKIP_TASKBAR ];
    a_NET_WM_STATE_SKIP_PAGER = atoms[ I_NET_WM_STATE_SKIP_PAGER ];
    a_NET_WM_STATE_STICKY = atoms[ I_NET_WM_STATE_STICKY ];
    a_NET_WM_STATE_HIDDEN = atoms[ I_NET_WM_STATE_HIDDEN ];
    a_NET_WM_STATE_SHADED = atoms[ I_NET_WM_STATE_SHADED ];
    a_NET_WM_WINDOW_TYPE = atoms[ I_NET_WM_WINDOW_TYPE ];

    a_NET_WM_WINDOW_TYPE_DESKTOP = atoms[ I_NET_WM_WINDOW_TYPE_DESKTOP ];
    a_NET_WM_WINDOW_TYPE_DOCK = atoms[ I_NET_WM_WINDOW_TYPE_DOCK ];
    a_NET_WM_WINDOW_TYPE_TOOLBAR = atoms[ I_NET_WM_WINDOW_TYPE_TOOLBAR ];
    a_NET_WM_WINDOW_TYPE_MENU = atoms[ I_NET_WM_WINDOW_TYPE_MENU ];
    a_NET_WM_WINDOW_TYPE_UTILITY = atoms[ I_NET_WM_WINDOW_TYPE_UTILITY ];
    a_NET_WM_WINDOW_TYPE_SPLASH = atoms[ I_NET_WM_WINDOW_TYPE_SPLASH ];
    a_NET_WM_WINDOW_TYPE_DIALOG = atoms[ I_NET_WM_WINDOW_TYPE_DIALOG ];
    a_NET_WM_WINDOW_TYPE_NORMAL = atoms[ I_NET_WM_WINDOW_TYPE_NORMAL ];
    a_NET_WM_DESKTOP = atoms[ I_NET_WM_DESKTOP ];
    a_NET_WM_PID = atoms[ I_NET_WM_PID ];
    a_NET_WM_NAME = atoms[ I_NET_WM_NAME ];
    a_NET_WM_VISIBLE_NAME = atoms[ I_NET_WM_VISIBLE_NAME ];
    a_NET_WM_STRUT = atoms[ I_NET_WM_STRUT ];
    a_NET_WM_STRUT_PARTIAL = atoms[ I_NET_WM_STRUT_PARTIAL ];
    a_NET_WM_ICON = atoms[ I_NET_WM_ICON ];
    a_KDE_NET_WM_SYSTEM_TRAY_WINDOW_FOR = atoms[ I_KDE_NET_WM_SYSTEM_TRAY_WINDOW_FOR ];

    a_NET_SYSTEM_TRAY_OPCODE = atoms[ I_NET_SYSTEM_TRAY_OPCODE ];
    a_NET_SYSTEM_TRAY_MESSAGE_DATA = atoms [ I_NET_SYSTEM_TRAY_MESSAGE_DATA ];
    a_NET_SYSTEM_TRAY_ORIENTATION = atoms[ I_NET_SYSTEM_TRAY_ORIENTATION ];
    a_MANAGER = atoms[ I_MANAGER ];

    a_LXPANEL_CMD = atoms[ I_LXPANEL_CMD ];

}

void
Xclimsg(Window win, Atom type, long l0, long l1, long l2, long l3, long l4)
{
    XClientMessageEvent xev;
    xev.type = ClientMessage;
    xev.window = win;
    xev.message_type = type;
    xev.format = 32;
    xev.data.l[0] = l0;
    xev.data.l[1] = l1;
    xev.data.l[2] = l2;
    xev.data.l[3] = l3;
    xev.data.l[4] = l4;
    XSendEvent(GDK_DISPLAY_XDISPLAY(gdk_display_get_default()), GDK_ROOT_WINDOW(), False,
        (SubstructureNotifyMask | SubstructureRedirectMask),
        (XEvent *) &xev);
```

```
}

void
Xclimsgwm(Window win, Atom type, Atom arg)
{
   XClientMessageEvent xev;

   xev.type = ClientMessage;
   xev.window = win;
   xev.message_type = type;
   xev.format = 32;
   xev.data.l[0] = arg;
   xev.data.l[1] = GDK_CURRENT_TIME;
   XSendEvent(GDK_DISPLAY_XDISPLAY(gdk_display_get_default()), win, False, 0L, (XEvent *) &xev);
}

void *
get_utf8_property(Window win, Atom atom)
{
   Atom type;
   int format;
   gulong nitems;
   gulong bytes_after;
   gchar *val, *retval;
   int result;
   guchar *tmp = NULL;

   type = None;
   retval = NULL;
   result = XGetWindowProperty (GDK_DISPLAY_XDISPLAY(gdk_display_get_default()), win, atom, 0, G_MAXLONG, False,
        a_UTF8_STRING, &type, &format, &nitems,
        &bytes_after, &tmp);
   if (result != Success || type == None)
      return NULL;
   val = (gchar *) tmp;
   if (val) {
      if (type == a_UTF8_STRING && format == 8 && nitems != 0)
         retval = g_strndup (val, nitems);
      XFree (val);
   }
   return retval;

}

char **
get_utf8_property_list(Window win, Atom atom, int *count)
{
   Atom type;
   int format;
   gulong nitems, i;
   gulong bytes_after;
   gchar *s, **retval = NULL;
   int result;
   guchar *tmp = NULL;

   *count = 0;
   result = XGetWindowProperty(GDK_DISPLAY_XDISPLAY(gdk_display_get_default()), win, atom, 0, G_MAXLONG, False,
        a_UTF8_STRING, &type, &format, &nitems,
        &bytes_after, &tmp);
   if (result != Success || type != a_UTF8_STRING || tmp == NULL)
      return NULL;

   if (nitems) {
      gchar *val = (gchar *) tmp;
```

```
      g_printf("res=%d(%d) nitems=%d val=%s\n", result, Success, nitems, val);
      for (i = 0; i < nitems; i++) {
         if (!val[i])
            (*count)++;
      }
      retval = g_new0 (char*, *count + 2);
      for (i = 0, s = val; (int)i < *count; i++, s = s +  strlen (s) + 1) {
         retval[i] = g_strdup(s);
      }
      if (val[nitems-1]) {
         result = nitems - (s - val);
         g_printf("val does not ends by 0, moving last %d bytes\n", result);
         g_memmove(s - 1, s, result);
         val[nitems-1] = 0;
         g_printf("s=%s\n", s -1);
         retval[i] = g_strdup(s - 1);
         (*count)++;
      }
   }
   XFree (tmp);

   return retval;

}

void *
get_xaproperty (Window win, Atom prop, Atom type, int *nitems)
{
   Atom type_ret;
   int format_ret;
   unsigned long items_ret;
   unsigned long after_ret;
   unsigned char *prop_data;

   prop_data = NULL;
   if (XGetWindowProperty (GDK_DISPLAY_XDISPLAY(gdk_display_get_default()), win, prop, 0, G_MAXLONG, False,
         type, &type_ret, &format_ret, &items_ret,
         &after_ret, &prop_data) != Success)
   {
      if( G_UNLIKELY(prop_data) )
         XFree( prop_data );
      if( nitems )
         *nitems = 0;
   }

   if (nitems)
      *nitems = items_ret;
   return prop_data;
}

static char*
text_property_to_utf8 (const XTextProperty *prop)
{
  char **list;
  int count;
  char *retval;

  list = NULL;
  count = gdk_text_property_to_utf8_list_for_display (gdk_display_get_default(),
                                  gdk_x11_xatom_to_atom (prop->encoding),
                                  prop->format,
                                  prop->value,
                                  prop->nitems,
                                  &list);
```

```
  g_printf("count=%d\n", count);
  if (count == 0)
    return NULL;

  retval = list[0];
  list[0] = g_strdup (""); /* something to free */

  g_strfreev (list);

}

char *
get_textproperty(Window win, Atom atom)
{
   XTextProperty text_prop;
   char *retval;

   if (XGetTextProperty(GDK_DISPLAY_XDISPLAY(gdk_display_get_default()), win, &text_prop, atom)) {
      g_printf("format=%d enc=%d nitems=%d value=%s   \n",
           text_prop.format,
           text_prop.encoding,
           text_prop.nitems,
           text_prop.value);
      retval = text_property_to_utf8 (&text_prop);
      if (text_prop.nitems > 0)
         XFree (text_prop.value);

   }
}


int
get_net_number_of_desktops()
{
   int desknum;
   gulong *data;

   data = get_xaproperty (GDK_ROOT_WINDOW(), a_NET_NUMBER_OF_DESKTOPS,
        XA_CARDINAL, 0);
   if (!data)

   desknum = *data;
   XFree (data);
}


int
get_net_current_desktop ()
{
   int desk;
   gulong *data;

   data = get_xaproperty (GDK_ROOT_WINDOW(), a_NET_CURRENT_DESKTOP, XA_CARDINAL, 0);
   if (!data)

   desk = *data;
   XFree (data);
}

int
get_net_wm_desktop(Window win)
{
   int desk = 0;
```

```
    gulong *data;

    data = get_xaproperty (win, a_NET_WM_DESKTOP, XA_CARDINAL, 0);
    if (data) {
        desk = *data;
        XFree (data);
    }
}

GPid
get_net_wm_pid(Window win)
{
    GPid pid = 0;
    gulong *data;

    data = get_xaproperty (win, a_NET_WM_PID, XA_CARDINAL, 0);
    if (data) {
        pid = *data;
        XFree (data);
    }
}

void
get_net_wm_state(Window win, NetWMState *nws)
{
    Atom *state;
    int num3;


    memset(nws, 0, sizeof(*nws));
    if (!(state = get_xaproperty(win, a_NET_WM_STATE, XA_ATOM, &num3)))

    g_printf( "%x: netwm state = { ", (unsigned int)win);
    while (--num3 >= 0) {
        if (state[num3] == a_NET_WM_STATE_SKIP_PAGER) {
            g_printf("NET_WM_STATE_SKIP_PAGER ");
            nws->skip_pager = 1;
        } else if (state[num3] == a_NET_WM_STATE_SKIP_TASKBAR) {
            g_printf( "NET_WM_STATE_SKIP_TASKBAR ");
        nws->skip_taskbar = 1;
    } else if (state[num3] == a_NET_WM_STATE_STICKY) {
            g_printf( "NET_WM_STATE_STICKY ");
        nws->sticky = 1;
        } else if (state[num3] == a_NET_WM_STATE_HIDDEN) {
            g_printf( "NET_WM_STATE_HIDDEN ");
            nws->hidden = 1;
    } else if (state[num3] == a_NET_WM_STATE_SHADED) {
            g_printf( "NET_WM_STATE_SHADED ");
            nws->shaded = 1;
    } else {
        g_printf( "... ");
    }
    }
    XFree(state);
    g_printf( "}\n");
}

void
get_net_wm_window_type(Window win, NetWMWindowType *nwwt)
{
    Atom *state;
    int num3;
```

```
    memset(nwwt, 0, sizeof(*nwwt));
    if (!(state = get_xaproperty(win, a_NET_WM_WINDOW_TYPE, XA_ATOM, &num3)))

    g_printf( "%x: netwm state = { ", (unsigned int)win);
    while (--num3 >= 0) {
       if (state[num3] == a_NET_WM_WINDOW_TYPE_DESKTOP) {
          g_printf("NET_WM_WINDOW_TYPE_DESKTOP ");
          nwwt->desktop = 1;
       } else if (state[num3] == a_NET_WM_WINDOW_TYPE_DOCK) {
          g_printf( "NET_WM_WINDOW_TYPE_DOCK ");
       nwwt->dock = 1;
    } else if (state[num3] == a_NET_WM_WINDOW_TYPE_TOOLBAR) {
          g_printf( "NET_WM_WINDOW_TYPE_TOOLBAR ");
       nwwt->toolbar = 1;
       } else if (state[num3] == a_NET_WM_WINDOW_TYPE_MENU) {
          g_printf( "NET_WM_WINDOW_TYPE_MENU ");
          nwwt->menu = 1;
    } else if (state[num3] == a_NET_WM_WINDOW_TYPE_UTILITY) {
          g_printf( "NET_WM_WINDOW_TYPE_UTILITY ");
          nwwt->utility = 1;
    } else if (state[num3] == a_NET_WM_WINDOW_TYPE_SPLASH) {
          g_printf( "NET_WM_WINDOW_TYPE_SPLASH ");
          nwwt->splash = 1;
    } else if (state[num3] == a_NET_WM_WINDOW_TYPE_DIALOG) {
          g_printf( "NET_WM_WINDOW_TYPE_DIALOG ");
          nwwt->dialog = 1;
    } else if (state[num3] == a_NET_WM_WINDOW_TYPE_NORMAL) {
          g_printf( "NET_WM_WINDOW_TYPE_NORMAL ");
          nwwt->normal = 1;
    } else {
       g_printf( "... ");
    }
    }
    XFree(state);
    g_printf( "}\n");
}

int
get_wm_state (Window win)
{
    unsigned long *data;
    int ret = 0;

    data = get_xaproperty (win, a_WM_STATE, a_WM_STATE, 0);
    if (data) {
       ret = data[0];
       XFree (data);
    }
}
```