NETWM

Материал из LorWiki

В этой статье мы рассмотрим вопросы, затрагивающие программирование с использованием спецификации NETWM.

Содержание

Введение

Первая спецификация, описывающая базовую функциональность общения X клиентов с сервером, и клиентов с оконным менеджером, называлась ICCCM (Inter-Client Communication Conventions Manual). Она описывает самые базовые принципы работы с X сервером.

Дальнейшее развитие эта спецификация получила в виде спецификации NETWM. NETWM описывает взаимодействие приложений с оконными менеджерами на более современном уровне. Тут вводятся понятия рабочего стола, рабочего пространства, захваченной области экрана, и др. Большинство популярных графических тулкитов и оконных менеджеров для X11 на данный момент являются NETWM совместимыми.

Важно знать, что все спецификации, работающие с X сервером (NETWM, Startup Notification, System Tray, XEMBED и др.), не вносят никаких новых прослоек, а их реализации не заменяют никаких X11 библиотек. Эти спецификации используют *стандартные* возможности X сервера, а их реализации являются обычными программами для X11, запускаемыми точно так же, как и любые другие.

Цель

К концу статьи наберётся несколько примеров, работающих с NETWM. По аналогии, имея на руках спецификацию NETWM, можно будет написать клиента, работающего с любыми атомами.

Атомы

Каждое окно в X11 может иметь набор некоторых свойств. Если говорить просто, то атомы - это идентификаторы таких свойств. С точки зрения программиста, атом - это ключ в ассоциативном массиве, в котором нас интересует значение, сопоставленное этому ключу. Каждое окно имеет свой

набор атомов. Каждому атому сопоставлено значение - бинарные данные определённого формата, которые хранятся на стороне сервера (?). Мы можем получить эти данные, обратившись к X серверу. Передав серверу идентификатор атома, значение которого мы хотим получить, идентификатор окна и и формат выходных данных, в ответ мы получаем значение этого атома для данного окна.

Атомы могут использоваться как для получения текущего значения свойства, так и для установки свойству нового значения.

При получении глобальных событий X11, приложение, основываясь на атомах, может определить, что конкретно произошло - сменился ли рабочий стол, появилось ли новое окно, и т.д.

Подготовка

Все примеры в качестве GUI библиотеки используют Qt4. Она используется в основном из-за простоты написания GUI приложения. Несмотря на это, все важные части примеров написаны на Xlib. Qt используется в основном для получения указателя на переменную типа Display (*QX11Info::display()*) и корневого окна (*QX11Info::appRootWindow()*). Эти функции можно заменить на соответствующие механизмы в других тулкитах (например *gdk_display_get_default()*). Для вывода сообщений используется *qDebug()* - аналог *printf()*.

Итак, чтобы начать работать с атомами, нужно получить идентификатор атома по его имени (пример без привязки к какому-либо тулкиту):

```
/*
 *  Функции работы с X сервером требуют переменную-указатель
 *  на тип Display, чтобы знать с каким дисплеем мы хотим работать.
 *  В обычной ситуации берём дисплей по умолчанию.
 *
 *  Qt аналог - QX11Info::display(), GDK - gdk_display_get_default()
 */
Display *dpy = XOpenDisplay(NULL);

/*
 *  Получаем идентификтор атома. False указывает на то,
 *  что если атом не существует, его нужно создать. При ошибке,
 *  возвращается None.
 *
 *  Atom - это просто typedef для unsigned long.
 */
Atom atom_wm_name = XInternAtom(dpy, "WM_NAME", False);

/*
 *  Мы получили от сервера идентификатор атома WM_NAME, теперь его сохраним куда-нибудь,
 *  чтобы постоянно не дёргать XInternAtom().
 */
```

Функция получения значений свойств, сопоставленных атому (тут уже используем Qt класс QX11Info):

```
void* property(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 = 0;

    // на основании атома и его типа берём значение свойства
    if(XGetWindowProperty(QX11Info::display(), win, prop, 0, 0x7fffffff, False,
                          type, &type_ret, &format_ret, &items_ret,
                          &after_ret, &prop_data) != Success)
        return 0;

    // количество полученных элементов
```

```
    if(nitems)
        *nitems = items_ret;

    return prop_data;
}
```

Функция отсылки сообщения какому-либо окну, принимает на вход идентификатор окна, атом и список аргументов.

```
void climsg(Window win, long type, long l0, long l1, long l2, long l3, long l4)
{
    XClientMessageEvent xev;

    // формируем сообщение (в качестве type передаётся атом)
    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;

    // и отсылаем, стандартным X11 механизмом
    XSendEvent(QX11Info::display(), QX11Info::appRootWindow(), False,
          (SubstructureNotifyMask | SubstructureRedirectMask),
          (XEvent *)&xev);
}
```

Для компиляции примеров необходима библиотека Qt4 с заголовками и заголовки X11. Также нужно дополнительно линковаться с -lX11.

Заметка: С помощью утилиты *xprop* можно получить значения всех атомов для выбранного окна.

ICCCM атомы

Рассмотрим атомы из ICCCM:

- **WM_NAME**

Имя, неинтерпретированная строка, которую необходимо отобразить оконному менеджеру. Обычно эта строка отображается как заголовок окна.

- **WM_ICON_NAME**

Тоже самое, что и WM_NAME, но для случая, когда окно свёрнуто. Обычно отображается в панели задач.

- **WM_NORMAL_HINTS**

Содержит структуру определённого формата (ICCCM 4.1.2.3. WM_NORMAL_HINTS Property), где указаны некоторые величины, относящиеся к размеру окна. Какие значения из этой структуры используются, указано в переменной-флаге.

- **WM_HINTS**

Содержит структуру определённого формата (4.1.2.4. WM_HINTS Property), где указаны некоторые величины, предназначенные для оконного менеджера. Какие значения из этой структуры используются, указано в переменной-флаге. Содержит иконку для свёрнутого состояния, расположение иконки и др.

- **WM_CLASS**

Содержит две последовательные строки (с \0 символом каждая). Может использоваться оконным менеджером для получения каких-нибудь ресурсов, специфичных для данного приложения. Например, KWin по значению этого атома загружает иконку из текущей темы иконок для тех окон, у которых нету встроенной иконки. Например, для Assistant из Qt4, не имеющего собственной иконки в свойствах, KWin загружает иконку assistant.png с диска. Первая строка определяет конкретное приложение, вторая - класс приложений, к которым относится окно. Например, для Konsole:

```
WM_CLASS(STRING) = "konsole", "Konsole"
```

- **WM_STATE**

Состояние окна. Содержит структуру (4.1.3.1. WM_STATE Property), в которой есть поле, определяющее текущее состояние окна - WithdrawnState, NormalState или IconicState.

- **WM_TRANSIENT_FOR**
- **WM_PROTOCOLS**
- **WM_COLORMAP_WINDOWS**
- **WM_CLIENT_MACHINE**
- **WM_ICON_SIZE**

NETWM атомы

Атомы, определённые на корневом окне:

- **_NET_SUPPORTED**

Список атомов, поддерживаемых текущим оконным менеджером. Массив идентификаторов Atom.

- **_NET_CLIENT_LIST**

Список всех окон, обслуживаемых оконным менеджером. Массив идентификаторов Window.

```
Window *win;
int num;

// получить указатель на массив идентификаторов Window
win = reinterpret_cast<Window *>(property(QX11Info::appRootWindow(),
                                    NET_CLIENT_LIST, XA_WINDOW, &num));

// ошибка
if(!win)
{
    qDebug("Cannot get window list");
    exit(1);
}

// num - число элементов в массиве
for(int i = 0;i < num;i++)
    qDebug("Window %ld", win[i]);

// освобождаем память
XFree(win);
```

Отобразятся идентификаторы всех окон, обслуживаемых оконным менеджером:

```
# ./test
Window 18874404
Window 20971555
Window 20971789
Window 23068704
Window 6293343
Window 33554472
Window 39845890
```

- **_NET_NUMBER_OF_DESKTOPS**

Число сконфигурированных рабочих столов.

```
ulong *u = reinterpret_cast<ulong *>(property(QX11Info::appRootWindow(),
                                    NET_NUMBER_OF_DESKTOPS, XA_CARDINAL));

if(!u)
{
    qDebug("Cannot get property");
    exit(1);
}

qDebug("Number of desktops: %ld", *u);

XFree(u);
```

```
# ./test
Number of desktops: 2
```

- **_NET_DESKTOP_GEOMETRY**

Общий (имеется ввиду не суммарный) размер рабочего стола.

- **_NET_DESKTOP_VIEWPORT**

Размер окна (не пространства) каждого рабочего стола. Доступное пространство может быть меньше окна стола.

- **_NET_CURRENT_DESKTOP**

Номер текущего рабочего стола.

```
ulong *u = reinterpret_cast<ulong *>(property(QX11Info::appRootWindow(),
                                    NET_CURRENT_DESKTOP, XA_CARDINAL));

if(!u)
{
    qDebug("Cannot get property");
    exit(1);
}

qDebug("Current desktop: %ld", *u);

XFree(u);
```

```
# ./test
```

```
Current desktop: 0
```

А теперь пример смены рабочего стола с индекса 0 на 1:

```
climsg(QX11Info::appRootWindow(), NET_CURRENT_DESKTOP, 1, CurrentTime);
```

- **_NET_DESKTOP_NAMES**

Имена рабочих столов в utf8 виде. Это строка, состоящая последовательно из нескольких строк, каждая с \0 в конце.

```
int num;
int ind = 0;

// сначала получим число сконфигурированных столов, т.к.
// NET_DESKTOP_NAMES может вернуть больше чем есть
ulong *u = reinterpret_cast<ulong *>(property(QX11Info::appRootWindow(),
                                      NET_NUMBER_OF_DESKTOPS, XA_CARDINAL));

if(!u)
{
    qDebug("Cannot get property");
    exit(1);
}

// число рабочих столов получено
num = *u;

XFree(u);

// имена столов
char *names = reinterpret_cast<char *>(property(QX11Info::appRootWindow(),
                                      NET_DESKTOP_NAMES, UTF8_STRING));

if(!names)
{
    qDebug("Cannot get property");
    exit(1);
}

for(int i = 0; i < num;i++)
{
    qDebug("Desktop name: %s", names + ind);

    ind += (strlen(names+ind) + 1);
}

XFree(names);
```

```
# ./test
Desktop name: Основной
Desktop name: Видео
```

- **_NET_ACTIVE_WINDOW**

Идентификатор текущего активного окна.

```
Window *w = reinterpret_cast<ulong *>(property(QX11Info::appRootWindow(),
                                       NET_ACTIVE_WINDOW, XA_WINDOW));

if(!w)
```

```
{
    qDebug("Cannot get property");
    exit(1);
}

qDebug("Active window id: %ld", *w);

XFree(w);
```

```
# ./test
Active window id: 23068704
```

- **_NET_WORKAREA**

Размер рабочей области рабочего стола, относительно *_NET_DESKTOP_VIEWPORT*. Приложения, захватывающие области рабочего стола через *_NET_WM_STRUT_PARTIAL*, уменьшают размер рабочей области.

- **_NET_SUPPORTING_WM_CHECK**

Атом для проверки является ли текущий оконный менеджер совместимым с NETWM. Содержит идентификатор Window окна-ребёнка, созданного для этой цели.

- **_NET_VIRTUAL_ROOTS**
- **_NET_DESKTOP_LAYOUT**

- **_NET_SHOWING_DESKTOP**

Атом, определяющий состояние "Показать/скрыть все окна на данном рабочем столе". Имеет значение 1 или 0. Если установить значение этого атома в 1, то все окна на текущем столе мгновенно скрываются. Это делает, например, апплет "Свернуть все окна" для панели KDE.

- **_NET_CLOSE_WINDOW**

Может использоваться для закрытия определённого окна.

- **_NET_MOVERESIZE_WINDOW**

Может использоваться для передвижения окна и для задания ему размеров.

- **_NET_WM_MOVERESIZE**

Тоже самое, что и *_NET_MOVERESIZE_WINDOW*, но с другими параметрами.

- **_NET_RESTACK_WINDOW**
- **_NET_REQUEST_FRAME_EXTENTS**

Атомы, которые могут быть определены на конкретном клиентском окне:

- **_NET_WM_NAME**

Аналогично WM_NAME, но в кодировке utf8. Если такой атом существует на данном окне, то ему (с большой вероятностью) отдаётся предпочтение.

- **_NET_WM_VISIBLE_NAME**

Отображаемое имя окна, наподобие *_NET_WM_NAME*. Это введено для того, чтобы WM мог отобразить другое имя вместо текущего, если, например, окно с таким именем уже существует (чтобы различать окна с одинаковыми *_NET_WM_NAME*).

- **_NET_WM_ICON_NAME**

Аналогично WM_ICON_NAME, но в кодировке utf8. Если такой атом существует на данном окне, то ему (с большой вероятностью) отдаётся предпочтение.

- **_NET_WM_VISIBLE_ICON_NAME**

Если используется другое имя иконки нежели *_NET_WM_ICON_NAME*, то *_NET_WM_VISIBLE_ICON_NAME* содержит это имя в кодировке utf8.

- **_NET_WM_DESKTOP**

Рабочий стол, на котором находится окно. Если окно прикреплено на все столы, то значение это атома - 0xFFFFFFFF (-1 в виде int32).

- **_NET_WM_WINDOW_TYPE**

Массив типов окна (каждый тип - атом). Каждое клиентское окно может быть одновременно нескольких типов. Самое обыкновенное окно имеет только один тип *_NET_WM_WINDOW_TYPE_NORMAL*. Функция получения типа окна может выглядеть так:

```
struct net_wm_window_type
{
    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;
};

void getNetwmWindowType(Window win, net_wm_window_type *nwwt)
{
    Atom *state;
    int num3;

    bzero(nwwt, sizeof(nwwt));

    // получим массив атомов с текущими типами окна
    if(!(state = (Atom *)property(win, NET_WM_WINDOW_TYPE, XA_ATOM, &num3)))
        return;

    // пройдёмся по массиву
    while(--num3 >= 0)
    {
        if(state[num3] == NET_WM_WINDOW_TYPE_DESKTOP)
            nwwt->desktop = 1;
        else if(state[num3] == NET_WM_WINDOW_TYPE_DOCK)
            nwwt->dock = 1;
        else if(state[num3] == NET_WM_WINDOW_TYPE_TOOLBAR)
            nwwt->toolbar = 1;
        else if(state[num3] == NET_WM_WINDOW_TYPE_MENU)
            nwwt->menu = 1;
        else if(state[num3] == NET_WM_WINDOW_TYPE_UTILITY)
            nwwt->utility = 1;
        else if(state[num3] == NET_WM_WINDOW_TYPE_SPLASH)
            nwwt->splash = 1;
        else if(state[num3] == NET_WM_WINDOW_TYPE_DIALOG)
            nwwt->dialog = 1;
        else if(state[num3] == NET_WM_WINDOW_TYPE_NORMAL)
            nwwt->normal = 1;
    }
```

```
    XFree(state);
}
```

- **_NET_WM_STATE**

Массив состояний окна (каждое состояние - атом). Каждое клиентское окно может быть одновременно нескольких типов. Например, атом *_NET_WM_STATE_SKIP_TASKBAR* указывает панели задач не отображать это окно у себя в списке, *_NET_WM_STATE_FULLSCREEN* - окно открыто на весь экран, *_NET_WM_STATE_ABOVE* - окно поддерживается поверх других и т.д. Функция получения состояния окна может выглядеть так:

```
// структура, хранящая все типы состояний
struct net_wm_state
{
    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;
};

void getNetwmState(Window win, net_wm_state *nws)
{
    Atom *state;
    int num3;

    bzero(nws, sizeof(nws));

    // получим массив атомов с текущими состояниями окна
    if(!(state = (Atom *)property(win, NET_WM_STATE, XA_ATOM, &num3)))
        return;

    // пройдёмся по массиву
    while(--num3 >= 0)
    {
        if(state[num3] == NET_WM_STATE_SKIP_PAGER)
            nws->skip_pager = 1;
        else if(state[num3] == NET_WM_STATE_SKIP_TASKBAR)
            nws->skip_taskbar = 1;
        else if(state[num3] == NET_WM_STATE_STICKY)
            nws->sticky = 1;
        else if(state[num3] == NET_WM_STATE_HIDDEN)
            nws->hidden = 1;
        else if(state[num3] == NET_WM_STATE_SHADED)
            nws->shaded = 1;
    }

    XFree(state);
}
```

- **_NET_WM_ALLOWED_ACTIONS**

Массив действий, которые могут быть применены пользователем к окну (каждое действе - атом). Например, *_NET_WM_ACTION_MOVE* - окно можно двигать, *_NET_WM_ACTION_MINIMIZE* -окно можно свернуть, *_NET_WM_ACTION_CHANGE_DESKTOP* - окно можно двигать с одного рабочего стола на другой и т.д.

- **_NET_WM_STRUT** и **_NET_WM_STRUT_PARTIAL**

Эти атомы используются для захвата приложением области экрана. Например, панель типа *kicker* или *ksmoothdock* используют струты для выделения себе особого места на экране. Доступное рабочее пространство сокращается на размер струта. Окна, развёрнутые на весь экран, не могут закрыть эту область, т.е. они разворачиваются только на всю доступную область. Как ведёт себя неразвёрнутое окно по отношению к струту не указано. Например, *KWin* не позволяет двигать окна в область струта. *OpenBox* напротив, позволяет сдвинуть окно так, что оно закроет собой струт, и панель, находящаяся в захваченной области, станет невидна.

_NET_WM_STRUT_PARTIAL появился позже *_NET_WM_STRUT*, поэтому для совместимости можно посылать два сообщения - одно с *_NET_WM_STRUT_PARTIAL*, другое с *_NET_WM_STRUT*.

- **_NET_WM_ICON_GEOMETRY**

Возможный размер иконки в свёрнутом виде. Значение этого атома может быть установлено сторонними утилитами, типа панели задач, для своих нужд.

- **_NET_WM_ICON**

Массив возможных иконок окна. В каждом элементе первыми unsigned long числами идут ширина и высота, далее - несжатый ARGB массив пикселей.

- **_NET_WM_PID**

Содержит PID приложения, владеющего окном. Например:

```
Window *win;
int num;

// получить указатель на массив идентификаторов Window
win = reinterpret_cast<Window *>(property(QX11Info::appRootWindow(),
                                    NET_CLIENT_LIST, XA_WINDOW, &num));

// ошибка
if(!win)
{
    qDebug("Cannot get window list");
    exit(1);
}

// num - число элементов в массиве
for(int i = 0;i < num;i++)
{
    // определим PID для этого окна
    ulong *u = reinterpret_cast<ulong *>(property(win[i], NET_WM_PID, XA_CARDINAL));

    if(!u)
    {
        qDebug("Cannot get property PID");
        continue;
    }

    qDebug("Window %ld belongs to app with pid %ld", win[i], *u);

    XFree(u);
}
```

```
# ./test
Window 18874404 belongs to app with pid 5512
Window 20971555 belongs to app with pid 5514
Window 20971789 belongs to app with pid 5514
Window 23068704 belongs to app with pid 5524
```

- **_NET_WM_HANDLED_ICONS**

- **_NET_WM_USER_TIME**

Время последней активности пользователя в этом окне.

- **_NET_WM_USER_TIME_WINDOW**

Идентификатор окна, содержащего атом *_NET_WM_USER_TIME*.

- **_NET_FRAME_EXTENTS**

Атомы, используемые в NETWM протоколе:

- **_NET_WM_PING**

Используется для того же, для чего и утилита *ping* - для проверки того, что окно всё ещё живо и обрабатывает сообщения. Окно, получившее событие с этим атомом должно сразу же отослать это же сообщение назад корневому окну, не трогая другие поля сообщения, кроме окна назначения. Оконный менеджер может убить приложение, владеющее этим окном, если оно не ответило на данный запрос в течение какого-то промежутка времени.

- **_NET_WM_SYNC_REQUEST**
- **_NET_WM_FULLSCREEN_MONITORS**

Другие атомы:

- **_NET_WM_FULL_PLACEMENT**
- **_NET_WM_SUPPORTED_LIST**

- **_NET_WM_WINDOW_OPACITY**

Используется для задания степени прозрачности окна (при использовании композитного менеджера окон наподобие *xcompmgr*). За примером работы этого атома можно обратиться к коду утилиты *transset*. Этот атом отсутствует в спецификации, тем не менее он есть и работает.

- **_NET_WM_STATE_ABOVE** и **_NET_WM_STATE_STAYS_ON_TOP**

Используются для установки свойства "Поверх всех окон". Также отсутствуют в спецификации.

Практическое использование

Пример общего кода

В разделе ссылок есть ссылка на рабочий проект Qt4, части кода которого использовались ранее. В нём содержится namespace *netwm*, в котором объеденены различные атомы и функции работы с NETWM. Этот namespace легко может быть преобразован в C структуру, или что-то ещё.

Шпион

NETWM находит своё практическое применение в такой программе, как панель задач. Архитектура X-сервера подразумевает, что любые клиенты могут получать любые X11-сообщения. И панели задач только это и нужно - получая все события X11, она сможет анализировать атомы в пришедших событиях, а значит точно знать, что сейчас происходит на рабочем столе - создалось ли новое окно, сменился ли текущий рабочий стол, активизировалось ли какое-то окно и т.д. При использовании различных тулкитов типа Qt может появится проблема с получением всех событий X11. Может произойти так, что до конечного окна, в котором обрабатываются X11-сообщения, некоторые

сообщения просто не дойдут, а будут отфильтрованы ещё на пути (например объектом *QApplication*). В таком случае, нужно переопределить (унаследовать *QApplication* и переопределить метод *x11EventFilter()*) фильтр событий так, чтобы он дополнительно отправлял все события в нашу панель задач. В этом случае наша панель становится X-шпионом - окном, получающем все события X11, несмотря на ограничения тулкита.

Теперь код для Qt виджета, получающего все события X11, может выглядеть так:

```
using namespace netwm;

bool Spy::x11Event(XEvent *event)
{
    // это не то, что нам надо. Нам нужны только события
    // изменения свойств окон.
    if(event->type != PropertyNotify)
        return false;

    // получаем атом и окно, где произошло событие.
    // Атом определяет тип события.
    Atom atom = event->xproperty.atom;
    Window win = event->xproperty.window;

    // глобальное событие - на корневом окне
    if(win == QX11Info::appRootWindow())
    {
        // список окон изменился
        if(atom == NET_CLIENT_LIST)
        { }
        // сменился текущий рабочий стол
        else if(atom == NET_CURRENT_DESKTOP)
        { }
        // появилось новое активное окно
        else if(atom == NET_ACTIVE_WINDOW)
        { }
        // расположение окон "вглубь" относительно друг друга
        // изменилось
        else if(atom == NET_CLIENT_LIST_STACKING)
        { }
    }
    // события, относящиеся к конкретному клиентскому окну
    else
    {
        // окно перешло на другой рабочий стол
        // (или прикреплено на все столы)
        if(atom == NET_WM_DESKTOP)
        { }
        // WM_NAME сменился (обычно заголовок окна)
        else if(atom == XA_WM_NAME) // если делать совсем хорошо,
                                    // то NET_WM_NAME тоже надо отлавливать
        { }
        // состояние окна сменилось
        else if(atom == NET_WM_STATE)
        { }
        // тип окна сменился
        else if(atom == NET_WM_WINDOW_TYPE)
        { }
        // иконка окна сменилась
        else if(atom == NET_WM_ICON)
        { }
        // сменился WM_HINTS, возможно надо перечитать иконку
        else if(atom == XA_WM_HINTS)
        { }
    }
```

Далее, в каждом конкретном if-е, используя ранее приведенные примеры кода, можно перечитать список окон, иконку окна, его заголовок, тип, состояние и т.д.

Естественно, X11-шпионом может быть любая программа.

Панель

Теперь в качестве примера сделаем простую панель, которая будет занимать 40 пикселей сверху экрана. Полный проект есть в разделе ссылок. Метод *winId()* возвращает X11 идентификтор окна. Алгоритм действий: создаём окно, задаём ему координаты и размер, захватываем для него область экрана, устанавливаем дополнительные NETWM свойства.

```
#include <QDesktopWidget>
#include <QX11Info>
#include <QPainter>

#include "panel.h"
#include "netwm.h"

#include <X11/Xlib.h>
#include <X11/Xatom.h>
#include <X11/Xutil.h>

using namespace netwm;

Panel::Panel() : QWidget(0, Qt::FramelessWindowHint)
{
    setAttribute(Qt::WA_OpaquePaintEvent);
    setAttribute(Qt::WA_DeleteOnClose);

    QDesktopWidget desktop;
    QRect rc = desktop.availableGeometry();

    move(rc.x(), 0);

    setFixedWidth(rc.width());
    setFixedHeight(40);

    // захватим область экрана
    setStrut();

    // установим другие важные NETWM свойства
    initX11Relationship();
}

void Panel::setStrut()
{
    // по спецификации, у нас должен быть массив из 12-ти элементов,
    // в котором будут прописаны все необходимые координаты
    unsigned long data[12] = { 0 };
    int i = 2;

    // координаты захвата
    data[i] = height();
    data[4 + i*2] = x();
    data[5 + i*2] = x() + width();

    // учитываем старую и новую спецификацию захваченных областей
    XChangeProperty(QX11Info::display(), winId(), NET_WM_STRUT_PARTIAL,
          XA_CARDINAL, 32, PropModeReplace, (unsigned char *)data, 12);

    XChangeProperty(QX11Info::display(), winId(), NET_WM_STRUT,
          XA_CARDINAL, 32, PropModeReplace, (unsigned char *)data, 4);
}

void Panel::initX11Relationship()
{
    Atom state[3];
    unsigned int val;

    // ставим тип нашей панели - док
    state[0] = NET_WM_WINDOW_TYPE_DOCK;

    // меняем свойство
    XChangeProperty(QX11Info::display(), winId(), NET_WM_WINDOW_TYPE, XA_ATOM,
              32, PropModeReplace, (unsigned char *)state, 1);

    // панель будет видима на всех рабочих столах
    val = 0xffffffff;

    XChangeProperty(QX11Info::display(), winId(), NET_WM_DESKTOP, XA_CARDINAL, 32,
          PropModeReplace, (unsigned char *)&val, 1);
```

```
    // панель не будет видима в pager
    state[0] = NET_WM_STATE_SKIP_PAGER;

    // панель не будет видима в панели задач
    state[1] = NET_WM_STATE_SKIP_TASKBAR;

    // оконный менеджер постарается держать нашу панель
    // прикреплённой к фиксированной позиции, даже во время прокрутки
    // виртуального пространства
    state[2] = NET_WM_STATE_STICKY;

    // меняем свойство
    XChangeProperty(QX11Info::display(), winId(), NET_WM_STATE, XA_ATOM,
          32, PropModeReplace, (unsigned char *)state, 3);
}

void Panel::paintEvent(QPaintEvent *)
{
    // белый фон
    QPainter p(this);
    p.fillRect(rect(), Qt::white);
}
```

Проблемы прозрачности

С помощью композитного менеджера окон и атома *NET_WM_WINDOW_OPACITY* мы можем регулировать прозрачность окна. В Qt даже есть поддержка этого атома в виде метода setWindowOpacity().

В качестве композитного менеджера может выступать *xcompmgr*. Но этот менеджер имеет плохой сторонний эффект - т.к. он не подразумевает замещения текущего менеджера окон, а работу параллельно с ним, то программное задание прозрачности будет работать некорректно. Устанавливая прозрачность на своё top-level окно, следует иметь в виду, что оно на самом деле может не быть реальным top-level окном, видимым на экране, т.е. Qt функция winId() не будет возвращать корректный идентификатор. Объясняется это просто - обычный оконный менеджер создаёт окно-рамку, и делает reparent для каждого top-level окна в это своё окно (для целей отрисовки рамки вокруг окон). Qt это знает и winId() возвратит правильный идентификатор. Теперь *xcompmgr* делает второй reparent для целей композитинга. Таким образом, именно в случае двух менеджеров окон мы не достигнем желаемого эффекта прозрачности, т.к. идентификатор окна, возвращаемый winId(), будет неверным.

Решение: используйте XQueryTree() чтобы найти реального родителя окна и устанавливайте прозрачность именно на него средствами X11 и NETWM.

Ссылки

- Спецификация NETWM (http://standards.freedesktop.org/wm-spec/wm-spec-latest.html)
- Спецификация ICCCM (http://www.x.org/docs/ICCCM)
- libqnetwm (http://code.google.com/p/libqnetwm)
- Qt проект с namespace netwm (http://rusfolder.com/34174585)
- Qt проект, пример панели (http://rusfolder.com/34174587)

Источник — «https://lorwiki.ru/index.php?title=NETWM&oldid=268»

Категория: Net WM
