xbattext, an X11 battery monitor for NetBSD

John AnkarstrÃ¶m

ABSTRACT

xbattext is a simple X11 program that displays, in text, the current battery level. Its source code serves as a good introduction to X11 programming. It is short, simple and easy to follow, as it accounts only for a single system – NetBSD. It makes use of both Xt (X toolkit intrinsics) and Xm (Motif), two of the most popular X libraries. It demonstrates how to access resources from *~/.Xdefaults*, how to display text in various colors and fonts and how to set timers outside of the main event loop to perform asynchronous tasks that are not triggered by user interaction.

This document is a commented version of the *xbattext* source code. It is generated with *re*, a reference-based literate programming system available at http://git.ankarstrom.se/re/. The uncommented source code is hosted at http://git.ankarstrom.se/x11/xbattext/.

If you want to see a screenshot of *xbattext*, skip ahead to the last page.

Definitions

Most of the includes regard standard C features, but there are two special ones worth mentioning:

Xm/Label.h contains the definitions for Motif's label widget, which is used to display the battery percentage. It includes the rest of the relevant X11 headers for us.

machine/apmvar.h defines the *apm_power_info* structure, which is needed in order to inspect the battery status on NetBSD. It will be retrieved via an *ioctl* request, so *sys/ioctl.h* is included as well.

```
#include <err.h>
#include <fcntl.h>
#include <machine/apmvar.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <Xm/Label.h>
```

The battery status is checked every five seconds by default.

```
/* interval in seconds */
#define INTERVAL 5
```

The battery level is considered to be "low" if it is below 30 percent.

The user is encouraged to modify this and the previous constant according to his own preferences.

```
/* low battery level */
#define ALERT 30
```

```
void update(XtPointer, XtIntervalId *);

/* resources */
struct res {
        XmFontList font_list;
        XmFontList alert_font_list;
        XmFontList charge_font_list;
        Pixel foreground;
        Pixel alert_foreground;
        Pixel charge_foreground;
} res;
static XtResource res_opts[] = {
        {"fontList", "FontList", XmRFontList, sizeof(XmFontList),
        XtOffset(struct res*, font_list), XtRImmediate, (caddr_t…
                )NULL},
        {"alertFontList", "AlertFontList", XmRFontList, sizeof(X…
                mFontList),
        XtOffset(struct res*, alert_font_list), XtRImmediate, (c…
                addr_t)NULL},
        {"chargeFontList", "ChargeFontList", XmRFontList, sizeof…
                (XmFontList),
        XtOffset(struct res*, charge_font_list), XtRImmediate, (…
                caddr_t)NULL},
        {"foreground", "foreground", XmRPixel, sizeof(Pixel),
        XtOffset(struct res*, foreground), XtRImmediate, (caddr_…
                t)NULL},
        {"alertForeground", "AlertForeground", XmRPixel, sizeof(…
                Pixel),
        XtOffset(struct res*, alert_foreground), XtRImmediate, (…
                caddr_t)NULL},
        {"chargeForeground", "ChargeForeground", XmRPixel, sizeo…
                f(Pixel),
        XtOffset(struct res*, charge_foreground), XtRImmediate, …
                (caddr_t)NULL},
};

/* state changes */
enum {
        SET_NONE = 0,
        SET_NOALERT = 1 << 0,
        SET_NOCHARGE = 1 << 1,
        SET_ALERT = 1 << 2,
        SET_CHARGE = 1 << 3
} change;
```

Two structures are needed to access the application's resources: *res*, which will hold the values of the resources, and *res_opts*, which defines the manner in which the resources should be assigned to the members of the *res* strcture.[1]

At the beginning of the program's execution, the *res* structure is filled by the function *XtGetApplicationResources* according to the definitions in *res_opts*.

The *Pixel* type is an unsigned long value representing a color, like *black* or *red3*. The *XmFontList* type technically represents a list of fonts, but for all intents and purposes, it will be used here only to represent a single font selection.

As you might guess from these resource definitions, *xbattext* allows the user to control the color and font of the battery display depending on the battery status.

xbattext inspects the state of the battery every five seconds. On every iteration, it updates the percentage displayed, but the font and color are changed only when it detects a relevant change in the battery's state.

When such a change is detected, the *change* variable is set. Its possible values represent the range of relevant changes in battery state. The program then changes the color and font depending on the value of *change*.

[1] "Resources" are the settings set by the user in *~/.Xdefaults* or *~/.Xresources*. For more information about resource management and the structure of the *XtResource* type, see http://lesstif.sourceforge.net/doc/super-ux/g1ae03e/part1/chap9.html.

wargs is an array used by *XtSetArg*, which stores arguments in it, and *XtSetValues*, which applies new settings to a given widget according to the arguments stored in it.

The boolean values *alerting* and *charging* that are set to true whenever *xbattext* detects that the battery is low or that the AC adapter is plugged in.

The other variables will be explained as we go along.

```
/* application state */
Arg wargs[10];
char *s;
int apmfd, alerting, charging;
struct apm_power_info info;
Widget toplevel, label;
XmString xms;
XtAppContext app_context;
```

Initial setup

```
/* program start */
int
main(int argc, char* argv[])
{
        toplevel = XtVaAppInitialize(
                &app_context,
                "xbattext",
                NULL, 0,
                &argc, argv,
                NULL,
                NULL);
```

The application is initialized. *toplevel* is set, which will serve as the parent for all widgets.

Note that many Xt functions have two variants: one that uses *XtSetArg* to collect arguments, and a variadic variant, marked by *Va*.

```
        if ((apmfd = open("/dev/apm", O_RDONLY)) == -1)
                err(1, "open");

        if ((s = malloc(5*sizeof(char))) == NULL)
                err(1, "malloc");
```

The battery level is queried through *ioctl* requests to */dev/apm*. The file descriptor is closed by the kernel when the program exits.

```
        /* load application resources */
        XtGetApplicationResources(toplevel,
                &res, res_opts, XtNumber(res_opts), NULL, 0);
```

XtGetApplicationResources fills the *res* structure with the resources set in *res_opts*.

```
        /* create motif label */
        label = XtVaCreateManagedWidget("label",
                xmLabelWidgetClass, toplevel,
                XmNlabel, "",
                NULL);
        alerting = 0;
        charging = 0;
```

The battery level will be displayed in a Motif label widget. It starts out containing an empty string.

```
        update(NULL, NULL);
        XtRealizeWidget(toplevel);
        XtAppMainLoop(app_context);
}
```

Before starting the main event loop, the *update* function is called, which creates a timer that will run independently of the event loop.[2]

[2] For more information about timeouts, see http://motifdeveloper.com/tips/tip16.html.

Operation

The *update* both sets up the timer and is called at the end of each timeout. Along the way, it checks the battery status and updates the text displayed in the label widget.

The first argument to *update* is a pointer to an arbitrary value set by the user when the timeout is registered. The second argument contains a pointer to the timeout identifier. Neither argument is used here.

As mentioned above, the battery status is retrieved through an *ioctl* request, `APM_IOC_GETPOWER`. It returns an *apm_power_info* structure (which must be zeroed first).

The battery percentage, contained in *info.battery_life*, is written to an *XmString*, a special type of string used by Motif. It is associated with a "font list element tag", containing information about the visual characteristics of the text. We just use the default.

The *wargs* array mentioned above now starts being filled with arguments that determine the state of the label widget. The number of arguments set is kept track of in the *i* variable. To begin with, the label string is set to the *XmString* value defined earlier.

The `SET_CHARGE` or `SET_NO CHARGE` bit is added to the *change* bitmap when a change in *info.ac_ state* is detected.

(The value of *charging* is checked and updated in order to prevent the font and color changes from unnecessarily being applied at every timeout regardless of whether the charging status has changed.)

```
/* update battery status and (re-)add timer */
void
update(XtPointer client_data, XtIntervalId *t)
{
        int i;

        /* reset temporary variables */
        i = 0;
        change = SET_NONE;



        /* get battery info */
        memset(&info, 0, sizeof(info));
        if (ioctl(apmfd, APM_IOC_GETPOWER, &info) == -1) {
                fprintf(stderr, "ioctl APM_IOC_GETPOWER failed\n");
                sprintf(s, "?");
                goto end;
        }

        /* put battery status into label */
        sprintf(s, "%d%%", info.battery_life);
        xms = XmStringCreate(s, XmFONTLIST_DEFAULT_TAG);



        XtSetArg(wargs[i], XmNlabelString, xms); i++;



        /* check charging status */
        if (!charging && info.ac_state == APM_AC_ON)
                change |= SET_CHARGE;
        else if (charging && info.ac_state != APM_AC_ON)
                change |= SET_NOCHARGE;
        charging = info.ac_state == APM_AC_ON;
```

The same applies when the program checks whether the battery level is below `ALERT`.

```
/* check low battery */
if (!alerting && info.battery_life < ALERT)
        change |= SET_ALERT;
else if (alerting && info.battery_life >= ALERT)
        change |= SET_NOALERT;
alerting = info.battery_life < ALERT;
```

Before the bits in *change* are acted upon, some prioritization is necessary. The charging indication overrides any other indication. The low battery indication is activated if the AC adapter is plugged out, but the battery is still low. Likewise, the charging indication is activated if the battery rises above the `ALERT` threshold, but the AC adapter is still plugged in.

(Remember that the *change* bitmap reflects a **change** in state. It does not reflect the current **state**.)

```
/* prioritize charging and low battery indications */
if (change & SET_CHARGE) change = SET_CHARGE;
if (change & SET_NOCHARGE && alerting) change = SET_ALERT;
if (change & SET_NOALERT && charging) change = SET_CHARGE;
```

After collecting and prioritizing the state changes, the foreground color and font of the label widget are set accordingly.

Note that if a *Pixel* resource is not defined, *XtGetApplicationResources* gives it the integer value zero, which also signifies the color black. Thus, the program cannot tell the difference between a missing value and a value of *black*.

Font lists, however, are set to null if undefined. If the *alert* or *charge* font list is undefined, *xbattext* uses the main font list instead.

```
/* act on state changes */
switch(change & 0xf) {
case SET_NOCHARGE:
case SET_NOALERT:
        XtSetArg(wargs[i], XtNforeground, res.foreground);
        i++;
        if (res.font_list != NULL) {
                XtSetArg(wargs[i], XmNfontList, res.font…
                        _list);
                i++;
        }
        break;
case SET_ALERT:
        XtSetArg(wargs[i], XtNforeground, res.alert_fore…
                ground);
        i++;
        if (res.alert_font_list != NULL) {
                XtSetArg(wargs[i], XmNfontList, res.aler…
                        t_font_list);
                i++;
        }
        break;
case SET_CHARGE:
        XtSetArg(wargs[i], XtNforeground, res.charge_for…
                eground);
        i++;
        if (res.charge_font_list != NULL) {
                XtSetArg(wargs[i], XmNfontList, res.char…
                        ge_font_list);
                i++;
        } else if (res.font_list != NULL) {
                XtSetArg(wargs[i], XmNfontList, res.font…
                        _list);
                i++;
        }
        break;
```

```
        }

set:    XtSetValues(label, wargs, i);
        XmStringFree(xms);

        /* add new timer */
end:    XtAppAddTimeOut(app_context, INTERVAL * 1000, update, NULL);
}
```

Finally, the values collected in *wargs* are associated with the label widget through the *XtSetValues* function. The *XmString* is freed, as a new one will be created on the next call to *update*, which at the end is registered through the *XtAppAddTimeOut* function to occur in `INTERVAL` seconds.

That is the totality of the *xbattext* source (almost; a long comment at the beginning of the file, explaining what the program does and how it should be compiled, was excluded). Hopefully, it shows that graphical UNIX programming is nothing to be afraid of. While Xlib, Xt and Xm tend nowadays not to be considered "best practice", they have a low barrier to entry, require little resources of the computer and are often installed by default on UNIX systems. It is better to use worse practices to create *something* than it is to use best practices and create nothing at all; if those are the alternatives, then perhaps best practices aren't.

If, after reading all this, you are still wondering why anyone would want a small window on their screen displaying the current battery level, then you should get acquainted with the X11 window manager *jwm*[3] and its "swallowing" feature, which removes the border from a given X11 program and displays it in the tray. You can regard it as tray icons according to the UNIX philosophy.

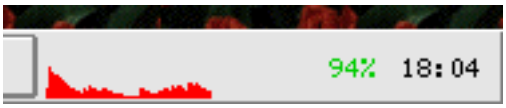

Figure 1. In my tray, *xbattext* sits right beside *xclock* and *xload*.

[3] *jwm* is available at https://joewing.net/projects/jwm/. My personal fork of *jwm* 1.8, which has a more traditional visual appearance, is available at http://git.ankarstrom.se/jwm/.