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 programnming. 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 the WWW address http://git.ankarstrom.se/re/.

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

Xm/Label.h contains the definitions for Motif's label widget, which is used to display the text. That file, in turn, includes *X11/Xlib.h* for us.

machine/apmvar.h has definitions needed in order to inspect the battery status on NetBSD.

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

By default, the battery status is checked every five seconds.

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

By default, the battery level is considered to be low if it is below 30 percent.

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

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

Two structures are created to access the application's resources: *res*, which will hold the values of the resources, and *res_opts*, which defines how those resources should be retrieved.

The *res* structure is going to be filled by the function *XtGetApplicationResources* using the information defined in *res_opts*.[1]

```
/* 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…
```

[1] 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.

```
                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},
};
```

The *change* variable represents the possible changes in battery state. *xbattext* updates the font and color of the battery display whenever one of these state changes occur.

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

There is not space to explain the function of all variables used by the program, but some of them deserve a special mention:

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.

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

int
main(int argc, char* argv[])
{
```

Many Xt functions have two variants: a non-variadic variant, which uses *XtSetArg* to collect arguments, and a variadic variant, marked by the *Va* component of its name.

toplevel is the “main” widget that contains all actual widgets.

```
        toplevel = XtVaAppInitialize(
                &app_context,
                "XBat",
                NULL, 0,
                &argc, argv,
                NULL,
                NULL);
```

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

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

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

As mentioned above, *XtGetApplicationResources* uses the *XtResource* list defined earlier to fill the *res* structure with the corresponding resources.

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

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

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

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

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

The *update* function, which is also called at the end of each timeout, is responsible for checking the battery status and updating the label.

The first argument is a pointer to a value set by the user when the timeout is registered. The second argument contains a pointer to the timeout ID. Neither argument is used in this program.

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

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

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).

```
        /* 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;
        }
```

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.

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

The *wargs* array mentioned above starts to be 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 widget's label string is set to the *XmString* value defined earlier.

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

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

The bits SET_CHARGE and SET_NOCHARGE are added to the *charge* bitmap when a change in *info.ac_state* is detected.

Note that the value of *charging* is used in order to prevent these font and color changes from unnecessarily being applied every timeout regardless of whether the charging status has changed.

```
/* 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, which is set to 30 by default.

```
/* 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.

```
/* 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 is set accordingly.

Note that if a *Pixel* (color) resource is not defined, *XtGetApplicationResources* gives it the integer value zero, which also signifies the color black (thus, the program cannot differentiate between a missing value and a value of *black*). Font lists, however, are set to null if undefined. If a font list resource is undefined, *xbattext* uses the default 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 scared of. While Xlib, Xt and Xm tend nowadays not to be considered "best practice", they have a low barrier to entry, require little resources from the computer and are often installed by default on UNIX systems. It is better to use worse practices to create *something* than to use best practices and create nothing at all; if those are the alternatives, then perhaps best practices aren't.

If, after reading the source code, you are still wondering why anyone would want a small window displaying the current battery level, then you should get acquainted with the X11 window manager *jwm* and its "swallowing" feature, which removes the border from a given X11 program and displays it in the tray. You can look at it as tray icons according to the UNIX philosophy. In my tray, *xbattext* sits right beside *xclock* and *xload*.

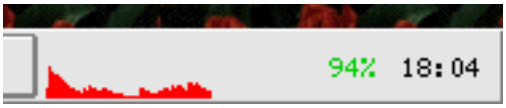

Figure 1. *xload*, *xbattext* and *xclock*.

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/.