Notes for the DOS port of Microwindows/Nano-X

Nano-X is primarily intended to be used with Linux. So it comes as source code which shall be compiled on the target system using the gcc compiler.

DJGPP is a DOS port of the gcc compiler which comes with a lot of Linux GNU utilities ported to DOS, such as the Make program. So I used that for the port of Nano-X to DOS.

Nano-X comes with two different APIs: one that is compatible to the Microsoft Windows-API and one that is compatible to the X-Windows API. So after compiling Nano-X you will have two libraries you can link with your application program: libmwin.a if you want to use the Windows-API and libnano-X.a if you want to use the X-Windows API.

1. Setup DJGPP

If you have not installed DJGPP yet, you should download the following files, if you have DJGPP already make sure you have them installed:

First enter the address below into your browsers address field or just click on this link:

ftp://**ftp.delorie**.com/pub/djgpp/current/

In this FTP directory there are several subdirectories. You should download the following files:

```
V2: readme.1st, djtzn203.zip, djdev204.zip
V2gnu: gcc453b.zip, bnu281b.zip, mak3791b.zip
V2misc: pmode11b.zip
```

To display JPEG, TIFF and PNG images you will need:

```
V2tk: jpeg8bb.zip, png152b.zip, zlib125b, tiff395b.zip
```

I recommend to install these files too which are required to port other Linux programs:

```
V2gnu: gpp453b.zip, acnf257b.zip, amak175b.zip, bsh203b.zip,
dif30b,zip, fil41b,zip, find41b.zip, grep28b.zip, pat261b.zip,
perl552.b.zip, sed421b.zip, shl2011b.zip, txt20b.zip,
pdcur34a.zip, txi40b.zip,
```

Unzip these files so you get a c:djgpp directory which contains all these in the appropiate subdirectories.

2. Setup the DJGPP environment

To compile Nano-X you will need long filename support for DOS. So I compiled it in a DOS window of Windows XP. If you compile in real mode DOS, you have to load DOSLFN to get long filename support. However, compilation takes much longer than in a DOS window of Windows XP since Windows has a large disk buffer which helps a lot with these large files.

Before using DJGPP I run the following batch file to setup the environment as necessary:

```
rem Put the djgpp path before the DOS command path or GNU find will not work
path=c:\djgpp\bin;%path%
rem Sets e.g. the include directory path
set DJGPP=c:\djgpp\djgpp.env
rem Shall redirect STDERR to STDOUT
rem set GO32=2r1
rem need to set this to get truetype fonts to work
set TTFONTDIR=e:\djgpp\mwin\src\fonts\truetype
```

3. Install and compile the Nano-X sources

After downloading the latest version of Nano-X, use 7zip to unpack it into a directory e.g.:

c:\djgpp\microwin

In the microwin directory there is a src directory which is the base directory from where you have to run the Make utility to compile Nano-X. So if you are in the command line window you go into that directory and enter Make on the command line.

The Nano-X package currently has several depreciated makefiles and config settings for DOS which will not work. Therefore I wrote new Makefiles which can be used with DJGPP.

Before running Make you have to install the thesel Makefiles I have written for DJGPP in their respective directories:

```
Makefile -> c:\djgpp\microwin\src
Makefile-mwin -> c:\djgpp\microwin\src\mwin (rename Makefile-mwin to Makefile)
Makefile-nanox -> c:\djgpp\microwin\src\nanox (rename Makefile-nanox to Makefile)
Makefile-bmp -> c:\djgpp\microwin\src\mwin\bmp (rename Makefile-bmp to Makefile)
Makefile-engine -> c:\djgpp\microwin\src\engine (rename Makefile-engine to Makefile)
Makefile-fonts -> c:\djgpp\microwin\src\fonts (rename Makefile-fonts to Makefile)
Makefile-drivers -> c:\djgpp\microwin\src\drivers (rename Makefile-drivers to Makefile)
```

There is a makecopy.bat file in the djgpp directory which shall do these copy operations for you.

The Makefile you have copied into c:\djgpp\microwin\src is the main Makefile you will run when you start Make from the command line in that directory. This file has several configuration options you can set.

Before running make make sure the three drivers mentioned in section four below are installed in the "c:\djgpp\microwin\src\drivers" directory. Also copy the X6x13.c font into the fonts directory.

This Makefile will first build the libmwin.a, the libnano-X.a and the libimages.a libraries. For this it calls the different Makefiles you copied into the subdirectories. The new libraries will be in the "lib" subdirectory when done.

Then the Makefile will compile the demos for the Windows-API and the XWindows-API. These will be in the "bin" directory as exe files where you can run them.

4. Screen drivers for DOS

The screen driver interface did change considerably between version 0.92 and 0.93. So I wrote two new DOS screen drivers for version 0.93.

One is based on the GRX graphics library like the DOS screen driver in the previous versions.

Since only a very tiny fraction of the GRX library's functionality is used by that screen driver I wrote a second one which uses VESA interface of the video graphics cards directly. This driver needs less memory space since it does not load the GRX library and seems to work a little bit faster too.

The driver using the GRX library is called scr_djgrx.c and the other one scr_djvesa.c / djvesa.h. Both have to be in the c:\djgpp\microwin\src\drivers directory.

These drivers work with 16bit, 24bit and 32bit TrueColor settings. I could not test the 8bit palette setting.

You have to set the desired screen resolution and pixel format in the main Makefile before compiling Nano-X. If you change the settings, do a "Make clean" before the next Make, since several modules need to read the new settings and adjust themselves accordingly.

I also wrote a modified keyboard driver for DOS since the interface for that also changed with version 0.93. Its name is "kbd_dj.c".

5. Images

Nano-X supports to display BMP, GIF, PNM and XPM images directly. For other image types such as JPEG, PNG and TIFF you need to add appropiate libraries which Nano-X will call. You will probably have already downloaded these libraries as described in section one.

The BMP images in the c:\djgpp\microwin\src\mwin\bmp directory are converted by executing the Makefile using the convbmp.c program to an internal Nano-X image format and then compiled into the libimages.a library to be used by the demo programs. The malpha demo uses the car8.bmp image this way.

You can also use the GrDrawImageFromFile() function to read an image from a file and display it in a window. The GrLoadImageFromFile() function on the other hand will load the image from file and return an image id number. This allows this image to be displayed later in the program. The nxview demo uses this function.

Unfortunately I could only get BMP and PPM images to work. The devimage.c code needs to be adapted for DJGPP. Therefore support for GIF, JPEG, PNG and TIFF images is not enabled as standard in the makefile for DJGPP. Just uncomment the lines provided for these image formats if you want to test this.

6. Fonts

Nano-X supports "bdf" bitmap fonts, Windows "fnt" bitmap fonts, X11 "pcf" fonts, Adobe Type1 or postscript fonts and "ttf" truetype fonts. Plus japanese and chinese fonts.

The "pcf" fonts can be stored used using z compression and will have a "gz" ending then. Nano-X then needs the "libz.a" library and its header files to read these. The "fnt" fonts could also be stored as gz compressed files.

Nano-X is compiled with two compiled-in system fonts. This are "winFreeSansSerif11x13" as variable font and "X6x13" as fixed font in the fonts directory. The demos usually use these fonts.

You can also extend these compiled-in fonts. For this you have to modify the drivers/genfont.c file. Add the desired fonts to the "gen_fonts" array there and modify NUMBER_FONTS number accordingly. Then change the Makefile in the fonts directory to include that font. For a file to be compiled-in you have to use a bdf font file and the convbdf program to convert that into a C program file. Then you compile that file and may add it to a library. The Makefile in the fonts directory will do this for several fonts.

The pcfdemo.c in demos/nanox allows to display a PCF font. The pcf font file can also be stored in a GZ archive since it takes quite some space on disk. If you specify such a file for pcfdemo on the command line, Nano-X will use the libz.a library to decompress this font file before using it. Furthermore this demo program allows to display a FNT plus a TrueType font in spite of its name too.

Demo2.c can also be compiled with various other fonts. If demo2.exe is in the bin directory add e.g. "../fonts/fnt" before the name of the font here.

Type 1 support is made with the library "libt1.a".

For TrueType support Nano-X needs the external library "libfreetype.a" plus a long list of header files. Set the environment variable as described above in section two for TrueType support. The font directory cannot be set in the makefile here successfully.

You also need to modify the Makefile in the engines directory to include font_freetype2.o in the objects to compile. Just uncomment that line provided.

I downloaded mupdf for DOS which comes with a compiled "libfreetype.a" library.
This is the link: http://www.ulozto.cz/7940392/mupdf-07-djgpp-rar . You have to guess Czech words to download it from there though.

The fontdemo.c program will show the different type sizes of a TrueType font on the screen.

7. Developing programs using Nano-X

To develop your own programs using Nano-X you could take the following steps:

1. create directory for your files e.g. c:myprogs.
2. copy libmwin.a, libnano-x and libimages.a from "src\lib" into c:\djgpp\lib.
3. make subdirectory "mwin" in c:\djgpp\include and put the files from "src\include" in that.
4. copy mtest.c from "src\demos\mwin" into c:myprogs since we use that as example.
5. run start.bat in c:\djgpp as described in section one to set path and environment.
6. edit mtest.c and change the include line to: "#include <mwin\windows.h>".

7. compile with: "gcc -Wall -g -s -O3 -o mtest.exe mtest.c -lmwin -lfreetype -lz"
8. run the mtest.exe program.

For other programs you may have to include fonts or other libraries too. Step 7 could be put into a batch file.

The resulting programs have a large size on disk compared with other DOS programs. I compiled with the "-s" switch which reduced the size by over 70%. This port has not worked further on reducing the size. Here are some recommendations to do that, e.g. remove the functions to read command line options:

http://www.delorie.com/djgpp/v2faq/faq8_14.html

Also the fonts and images could be compiled as DXE files, DJGPP's way to make a dynamic loadable library: http://www.delorie.com/djgpp/v2faq/faq22_15.html

Some additional remarks regarding program development:

If you need GUI widgets like buttons etc. you can use the Microwindows API which includes these just like MS-Windows. The Nano-X API just has the functionality of a graphics library. There were NXWidgets in earlier version but these are not longer included since FLTK or GTK can be used with Nano-X/NXLIB when used with a Linux operating system. Porting FLTK or GTK to DJGPP still has to be done.

Nano-X uses DPRINTF(), EPRINTF() and FPRINTF() for output of error messages. These are not properly displayed when the screen is in TrueColor mode. You can use the DJGPP redir program to redirect standard error and standard output to files. E.g.
"redir -e err.log -o text.log nxview tux.gif" will write standard error messages into err.log and messages for standard out (the default screen in text mode) to text.log. You can use redir also with the make statement. The warnings will then be in err.log and the executed statements in text.log.
For debugging you can insert "printf()" statements in the code. You can either use redir as described above or run e.g. "mtest.exe >test.log" and the ouput of these printf() statements will be written into the file test.log.

Some demos will not return to text mode if they terminate due to an error. The DOS command line will then result in colored pixels on the screen.

Read the following page about handling of slashes "/" by DJGPP and converting Linux device names to DOS: http://www.delorie.com/djgpp/doc/libc/libc_626.html

This page also mentions that getting a program to compile with DJGPP is usually not enough, you also have to check if it works as it does under Linux. Then you have to apply the necessary changes. I did not do that when porting to

If you are using the microwindows interface, there is a makedlg.c program in the "mwin\src\contrib\makedlg" directory which will make a dialog for your program using a resource file.

8. Compiling NXLIB

If you have a package which uses X11 on Linux and you want to use that with Nano-X you can adapt the code of this package to work with Nano-X.

Or you use NXLIB which is a X11 Conversion Library for Nano-X. This provides a full X11 interface to application programs so these do not require code changes to work with Nano-X. The NXLIB library converts X11 calls to nano-X calls. Functions which are not implemented just return 0. This is done in the stub.c program, where you can add functions that may be called by your application and which are not implemented yet.

With DJGPP you will compile a static library of NXLIB and then compile the Linux application including this library. However, you will probably also need the GNU utilities converted to DJGPP as recommended for download in section one. Perl is required to compile NXLIB.

So download NXLIB from the Microwindows website and use 7zip to unpack it into myprogs directory.

To compile NXLIB with DJGPP you have to set the directories in the makefile along the lines of this example, which assumes that you have made the steps as described in section 7 before:

```
MWIN_INCLUDE=/djgpp/include/mwin -I/myprogs/nxlib
MWIN_LIB=/djgpp/lib

X11_INCLUDE=/usr/include

X11_RGBTXT=fonts/rgb.txt

INSTALL_DIR=/myprogs/nxlib/lib
SHAREDLIB=N
INCLUDE_XRM=N
```

Then remove "./" in front of keysymstr.h in the makefile:

```
keysymstr.h:
      perl ./keymap.pl $(X11_INCLUDE)/X11 > keysymstr.h
```

Then make a directory called "lib" in "/myprogs/nxlib/"

I recommend to compile NXLIB with the original X11 header files. So copy the directory "usr/include/X11" from a Linux installation and put this as "usr/include/X11" on the disk where you compile NXLIB.

You will also need the files "keyboard.h" and "kb.h" which you can copy from any linux distribution from the directory "usr/lib/linux". Make a directory "/usr/include/linux" and put the files into that. Then:

In kd.h you have to comment out the line:
`//#include <linux/types.h>`
and in keyboard.h the line:
`//#include <linux/wait.h>`

Then patch the "StrKeysym.c" file by removing all:
`//#if linux`
and the corresponding
`//#endif /* linux*/`

Now you can enter "make" and a new libNX11.a will be generated in the nxlib directory.

If you have to repeat make, you have to enter "make distclean" before that. A Perl script will generate the keysymstr.h file needed for the compilation and this is removed again by "make distclean".

Now that we have successfully made NXLIB we can copy libNX11.a into /djgpp/lib and "usr/include/X11" into /djgpp/include/X11.

9. Compiling a program using NXLIB

There are only a few application programs which are based on X directly. Usually GTK, FLTK, Enlightenment, Xt etc. is used to have GUI widgets. In Linux the FLTK would run with NXLIB but I did not get that to run with DJGPP.

For a test of our newly compiled NXLIB we will compile a demo program which depends on X. Here is a very simple X program which I call cross.c. Compile this with:

gcc -I/djgpp/include/mwin -o cross.exe cross.c -lNX11 -lnano-X -lz -lfreetype

If you have compiled nano-X without z and freetype support, you probably will not need these libraries on the command line.

Please keep the order of the -l statements here since the libraries depend on each other.

Now the code for cross.c:

```
/*
* Xlib programming demo - display a window with a cross on the screen
* based on:
* http://xopendisplay.hilltopia.ca/2009/Jan/Xlib-tutorial-part-1----Beginnings.html
*/

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xresource.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char ** argv){
       int screen_num, width, height;
       unsigned long fcolor, bcolor;
       Window win;
       XEvent ev;
       Display *dpy;

       /* First connect to the display server */
       dpy = XOpenDisplay(NULL);
       if (!dpy) {fprintf(stderr, "unable to connect to display");return 7;}

       /* these are macros that pull useful data out of the display object */
       /* we use these bits of info enough to want them in their own variables */
       screen_num = DefaultScreen(dpy);
```

```
      bcolor = BlackPixel(dpy, screen_num);
      fcolor = WhitePixel(dpy, screen_num);

      // get the default colormap
      Colormap colmap = DefaultColormap(dpy, DefaultScreen(dpy));

      /* allocate a "red" color map entry. */
      XColor system_blue, system_red;
      Status rc;

      /* allocate blue color with values (0, 0, 0x8000) in RGB. */
      system_blue.red = system_blue.green =0;
      system_blue.blue = 0x8000;
      rc = XAllocColor(dpy,colmap,&system_blue);
      /* make sure the allocation succeeded. */
      if (rc == 0) {printf("XAllocColor blue failed.\n"); return 2;}
      /* allocate red color with values (0xFFFF, 0, 0) in RGB. */
      system_red.red = 0xFFFF;
      system_red.green = system_red.blue = 0;
      rc = XAllocColor(dpy, colmap, &system_red);
      if (rc == 0) {printf("XAllocColor red.\n"); return 3;}

      width = height = 400; /* size of window in pixel */
      win = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy), /* display, parent */
            20,20, /* place of window on screen */
            width, height,
            5, fcolor, /* border width & color */
            system_blue.pixel); /* background color */

      /* tell the display server what kind of events we would like to see */
      XSelectInput(dpy, win, ButtonPressMask);

    // Get the fixed font.
      XFontStruct *font = XLoadQueryFont( dpy, "System" );
    if( !font ){printf("Error, couldn't load font\n" ); return 1 ;}

      // in order to draw, we need a graphics context.
      XGCValues gv;
       GC gc = XCreateGC(dpy,screen_num,GCFunction | GCForeground | GCBackground,&gv);

      XSetForeground(dpy, gc, system_red.pixel);

      /* put the window on the screen */
      XMapWindow(dpy, win);

    // get the geometry of our window
    XWindowAttributes attr;
    XGetWindowAttributes( dpy, win, &attr );
    int w = attr.width, h = attr.height;

    // Draw a big cross that covers the whole window.
    XDrawLine( dpy, win, gc, 0, 0, w, h );
    XDrawLine( dpy, win, gc, 0, h, w, 0 );

    const char *text = "Hello World!";
    XDrawString( dpy, win, gc, 170, 60, text, strlen( text ) );

      /* Exit if button is pressed inside window */
      while(1){
            XNextEvent(dpy, &ev);
            switch(ev.type){
            case ButtonPress:
                  XCloseDisplay(dpy);
                  GrClose(); //Nano-X function - return to text screen mode
                  return 0;
            }
      }
}
```