Getting Started With Microwindows & Nano-X

Gary James
Sr Software Engineer
Critical Link, LLC (http://www.criticallink.com), Personel Homepage (http://home.twcny.rr.com/embedded)

gary.james@criticallink.com
gjames@twcny.rr.com

Getting Started With Microwindows & Nano-X

by Gary James

Table of Contents

List of Figures

List of Examples

Release Information

Release Date. June 21, 2001

Nano-X Revision. 0.89-pre7

Chapter 1. Getting Started With Microwindows

1.1. Installing Microwindows

This document will guide you through installation of Microwindows onto a Linux host. At the end of this tutorial you will be able to execute the Microwindows demo applications in an X window that simulates the screen of your embedded device. You will also be able to create and execute a simple hello world application using the nano-X API.

In the following chapters you will also learn how to work with images and TrueType fonts.

1.1.1. Download The Latest Sources

Download the latest source code from the Microwindows ftp site (ftp://microwindows.org/pub/microwindows/). In the following example I'll be building revision 0.89pre7. The sources come in a tarball named `microwindows-0.89pre7.tar.gz`. Copy this file into a convenient spot. I copied the file into my home directory `/home/gary`.

Extract the sources. The tarball is a compressed tar file. You must first uncompress the file using gunzip, then extract the sources from the tar file using tar. This creates a new directory named `microwin` that contains the Microwindows source tree. Type the following commands.

```
$ tar -xzf microwindows-0.89pre7.tar.gz
```

1.1.2. Configure The Build Scripts

Change directories to the new Microwindows source directory.

```
$ cd microwin/src
```

You should take the time to read the file `microwin/src/INSTALL` and then look through the file `microwin/src/config`. There are many different options that you can compile into Microwindows. For now the only difference that you want from the default is to build for X11 display. When your application goes into your embedded system you will most likely want to use FRAMBUFFER drivers. But for now we will be prototyping our system on a desktop machine running X windows. This will let us see our output in a window on the same desktop as our compiler and debugger.

To build for X windows you need to first set the configuration file for the proper X11 settings. Luckily there are preset configurations in the `microwin/src/Configs` directory. Copy the file `config.x11` over the file `config`.

```
$ cp Configs/config.x11 config
```

1.1.3. Build The Library And Demos

After making modifications to the configuration file you can build the Microwindows library and demo applications. Build the libraries and demos by typing:

```
$ make clean; make
```

You should add the microwindows binary directory to your path. We will be running applications in that directly quite a bit, especially when we start using nano-X. Run the following command, you may also want to add the command to your `~/.bashrc` file.

```
$ export PATH=~/microwin/src/bin:$PATH
```

1.1.3.1. Run the WIN32 demos

Test the build by running a few of the Microwindows demo applications. Press the **Pause** key to exit each of these applications.

```
$ mine
$ mtest
$ mdemo
$ malpha
```

Currently the WIN32 layer of Microwindows can only support one application running on the desktop at a time. If you are interested in running multiple applications simultaneously you will need to use the nano-X layer of microwindows rather than the WIN32 layer.

1.1.3.2. Run the NANO-X demos

Also run a few of the nano-X demo applications. To run the nano-X applications you must first run the nano-X server, then the application. The following three commands will start a nano-X server, then start the nxclock application, then the nxscribble application. You will see a black screen with two windows, one for each application. Press the **Break** key to exit each of these applications.

```
$ nano-X & sleep 1
$ nxclock &
$ nxscribble &
```

An alternative command line to accomplish the same thing is shown below. The sleep command is required to give the nano-X server a bit of time to initialize before starting the client applications.

```
$ nano-X& sleep 1; nxclock& nxscribble&
```

You can’t move the windows around within the nano-X screen, without the aid of a window manager. Luckily a window manager is included in the demo directory. Kill

the previous demo by pressing the **Break** key. This time run the nanowm window manager before running the nxclock or nxscribble applications.

```
$ nano-X& sleep 1; nanowm& sleep 1; nxclock& nxscribble&
```

Now you will see a cyan screen with two decorated windows, one for each application. The significant difference is not that we have a cyan background. The significant difference is that we have title bars on the two applications. Now you can move the windows around the screen by dragging their title bars. You can also exit an individual application. Try this by clicking the X button on the top of the nxclock window. The nxclock application will quit, while the nxscribble application continues to run. You can restart the nxclock application back at the command line, and a new clock window will appear.

1.1.4. Install The Libraries

If everything went as planned and the demo applications ran then you should install the libraries now. Switch to *root* user id and type:

```
# make install
```

This will copy the appropiate files to `/usr/include/microwin` and `/usr/lib`. You don't need to install the libraries to run the demo applications, but you will need them to build your own applications. The libraries that will be installed are:

```
usr/lib/libmwdrivers.a
usr/lib/libmwengine.a
usr/lib/libmwfonts.a
usr/lib/libmwimages.a
usr/lib/libmwin.a
usr/lib/libmwinlib.a
usr/lib/libmwobjects.a
usr/lib/libnano-X.a
usr/lib/libnwidget.a
```

```
usr/lib/libvncauth.a
```

1.2. Hello World Example

In this section I present a simple nano-X application. This application is the classic "hello world" in nano-X style. When you run the application you will see a single white window with the text "Hello World". If you run the application with nanowm the application’s window will have a title bar, and a resizable border.

1.2.1. hello.c

Copy the source shown below into a file named "`hello.c`". Compile the application with the following command.

```
$ gcc hello.c -o hello -lnano-X
```

Example 1-1. hello.c

```
#include <stdio.h>
#define MWINCLUDECOLORS
#include "microwin/nano-X.h"

GR_WINDOW_ID  wid;
GR_GC_ID      gc;

void event_handler (GR_EVENT *event);

int main (void)
{
    if (GrOpen() < 0)
    {
        fprintf (stderr, "GrOpen failed");
```

```
        exit (1);
    }

    gc = GrNewGC();
    GrSetGCUseBackground (gc, GR_FALSE);
    GrSetGCForeground (gc, RED);

    wid = GrNewWindowEx (GR_WM_PROPS_APPFRAME |
                         GR_WM_PROPS_CAPTION  |
                         GR_WM_PROPS_CLOSEBOX,
                         "Hello Window",
                         GR_ROOT_WINDOW_ID,
                         50, 50, 200, 100, WHITE);

    GrSelectEvents (wid, GR_EVENT_MASK_EXPOSURE |
                         GR_EVENT_MASK_CLOSE_REQ);

    GrMapWindow (wid);
    GrMainLoop (event_handler);
}

void event_handler (GR_EVENT *event)
{
    switch (event->type)
    {
    case GR_EVENT_TYPE_EXPOSURE:
        GrText (wid, gc, 50, 50,
                "Hello World",  -1, GR_TFASCII);
        break;

    case GR_EVENT_TYPE_CLOSE_REQ:
        GrClose();
        exit (0);
    }
}
```

1.2.2. Include Files

The header file "`microwin/nano-X`" defines the Microwindows and nano-X data structures, variables and functions. This file will be included in all source files that make nano-X API calls.

If we start at the top with the include files you will first notice the define for `MWINCLUDECOLORS`. This definition enables the definition of common system colors. The following color names can be used if `MWINCLUDECOLORS` is defined before the nano-X header files.

BLACK	BLUE	GREEN	CYAN	RED
MAGENTA	BROWN	LTGRAY	LTBLUE	LTGREEN
LTCYAN	LTRED	LTMAGENTA	YELLOW	WHITE
DKGRAY				

1.2.3. Library Initialization

A single function call, `GrOpen()`, will open and initialize the nano-X library. The function sets up the screen, keyboard and mouse device interfaces. This must be the first nano-X function that your application calls.

1.2.4. Create a Graphics Context

Nano-X uses objects called graphics contexts to describe drawing attributes. Among other things a graphics context (GC) will describe the colors to use when drawing graphical objects using nano-X.

Your application may allocate as many graphics contexts as you wish. Each drawing function call takes a GC as a parameter. For example if you wanted to draw red and blue text on a white background you might create one GC. You could set the

foreground color to red and draw the red text. Then set the foreground color to blue and draw the blue text. Another approach is to create two GCs, one with a red foreground and the other with a blue foreground. With two GCs you would use the first GC for drawing red text and the second GC for drawing blue text.

In the "hello world" example I create one GC using the `GrNewGC()` function. Then I configure the GC so that it does not use a background color and I set the foreground color to red. I save the ID of the GC for use later when I start drawing onto the application window.

1.2.5. Create a Window

Now you're going to need a window to draw onto. The next section of the example creates a main window for the application. The `GrNewWindowEx()` function creates our "hello world" application's main window. `GrNewWindowEx()` is the preferred method to create windows. Another function `GrNewWindow()` has been depreciated since it can not specify window decoration options to a window manager.

In this example we have a single main window with a title bar. The title bar caption reads "Hello Window".

1.2.6. Select Events

In nano-X you must select the types of events that you want a window to receive. After you create the window, you must make a call to the `GrSelectEvents()` function to choose the events that the window will receive. In our example we choose to receive exposure events and close request events.

By selecting exposure events you will know when the window needs to be redrawn. By selecting close request events, you will know when the window is closed.

1.2.7. Show The Window

To make the window visible your application must "map" the window. You will call the function `GrMapWindow()` to make the window visible.

1.2.8. Handle Events

After creating the main window, selecting events and mapping the window, the application can enter it's main event dispatch loop. The nano-X library provides several ways to implement the application's event dispatch loop. The easiest of these methods is the `GrMainLoop()` function. This function takes as a parameter, a pointer to your application's event handler. The event handler function will be invoked each time that the nano-X event queue receives a selected event.

In the example the function `event_handler()` serves as the event handler. Within this function is a switch on the event type. The two events that we select in the example are the exposure event and the close request event.

1.2.8.1. `GR_EVENT_TYPE_EXPOSURE`

Exposure events are nano-X's means of asking the application to redraw the contents of a window. Your application must redraw the window contents each time it gets an exposure event. You can not draw the window once and then forget about what's in there.

You will receive an exposure event after the window is mapped for the first time. You will also receive exposure events when the window is re-exposed. For example, let's imagine another window within your application or another application covers your window. When that window is moved exposing a portion, or all of, your window, nano-X will send your application an exposure event.

In our example we handle the exposure event by drawing the text "hello world" onto the window. Notice that when we call the function `GrText()` we specify a window ID and a graphics context ID. These are the IDs that we received earlier when we created the window and the GC.

1.2.8.2. `GR_EVENT_TYPE_CLOSE_REQ`

When you close the application window nano-X sends a close request event. The hello world application calls `GrClose()` to close the connection to the nano-X server. Then we exit the application.

1.2.9. Drum Roll Please...

Run the "hello world" application with the following command. You will see a window appear as shown below.

```
$ nano-X& sleep 1; nanowm& sleep 1; ./hello&
```

Figure 1-1. Hello World Example

1.2.10. Summary

This simple example program shows the structure of most, even much more complicated, nano-X applications. You will almost always connect to server, create windows and GCs, select events, map the windows and then process events.

Chapter 2. Working With Images

2.1. Embeddeding Images in an Application

Microwindows comes with a utility program called convbmp. This utility is used to convert Windows™ style bitmap files into C source code. This allows an your application to have a small number of images embedded within its program memory. This method is very useful for small embedded systems with no file system to store images.

This first image example will show you how to embed an image wthin your application. After the first example more examples will be given to show you how to read the same image from a file at run time.

2.1.1. Convert a Bitmap Image to C Source

Create a directory in which to build this example. Then copy the bitmap image of Tux from the Microwindows sources to this directory. Lastly use convbmp to convert the bitmap file to a C source file that we can compile into our application.

Note: In this example assume that Microwindows is installed to `~/microwin` and the examples are built in `~/mymw/ex_image_01`. If you have different locations then you will have to modify the paths used in the example accordingly.

```
$ cd ~/mymw/ex_image_01
$ cp ~/microwin/src/mwin/bmp/penguin.bmp penguin.bmp
$ convbmp penguin.bmp
```

Examine the contents of the file `penguin.c` that you just created. The file contains three structures. The first static structure is a color palette of up to 256 unique colors that convbmp found within the image. The second static structure is an array of the bits

from the image. The last structure is the public MWIMAGEHDR structure that your application will reference. This structure is named image_penguin in this example. convbmp will always use the naming convention "image_" followed by the base filename. In this example we started with `penguin.bmp`, therefore our image structure name "image_penguin".

2.1.2. Nano-X Client Fixup

The current version of nano-X (0.89pre7) has an arbitrary limit of 10,000 bytes per message that may be sent through the socket from the client to the server. Some functions, such as `GrArea()` will break up messages that exceed this size into smaller chunks. Unfortunately the function `GrDrawImageBits()` does not break up large messages into smaller chunks. The penguin bitmap is larger than 10,000 bytes so this example is not going to work as is. You could run the example with a smaller image, or increase the maximum message length. To increase the maximum image length you need to edit the nano-X source file `~/microwin/src/nanox/nxproto.h`. Change the line:

```
#define MAXREQUESTSZ 10000 /* max request size (65532)*/
```

to:

```
#define MAXREQUESTSZ 30000 /* max request size (65532)*/
```

Then rebuild and reinstall the Microwindows package.

2.1.3. Example Drawing Tux I

Copy the source shown below into a file named "`image_01.c`". Compile the application with the following command.

Note: If you did not install Microwindows you will need to change the path `/usr/include/microwin` so that it points to the include directory where you

extracted the Microwindows source.

```
$ gcc image_01.c penguin.c \
> -I/usr/include/microwin \
> -o image_01 -lnano-X
```

Example 2-1. image_01.c

```
#include <stdio.h>
#define MWINCLUDECOLORS
#include "microwin/nano-X.h"

GR_WINDOW_ID  wid;
GR_GC_ID      gc;

extern GR_IMAGE_HDR  image_penguin;

void event_handler (GR_EVENT *event);

int main (void)
{
    if (GrOpen() < 0)
    {
        fprintf (stderr, "GrOpen failed");
        exit (1);
    }

    gc = GrNewGC();

    wid = GrNewWindowEx (GR_WM_PROPS_APPFRAME |
                         GR_WM_PROPS_CAPTION  |
                         GR_WM_PROPS_CLOSEBOX,
                         "Tux Window I",
                         GR_ROOT_WINDOW_ID, 50, 50,
                         image_penguin.width,
```

```
                        image_penguin.height,
                        WHITE);

    GrSelectEvents (wid, GR_EVENT_MASK_EXPOSURE |
                        GR_EVENT_MASK_CLOSE_REQ);

    GrMapWindow (wid);
    GrMainLoop (event_handler);
    return 0;
}

void event_handler (GR_EVENT *event)
{
    switch (event->type)
    {
    case GR_EVENT_TYPE_EXPOSURE:
        GrDrawImageBits (wid , gc , 0 , 0, &image_penguin);
        break;

    case GR_EVENT_TYPE_CLOSE_REQ:
        GrClose();
        exit (0);
    }
}
```

Run the example application with the following command. You will see a window appear as shown below.

```
$ nano-X& sleep 1; nanowm& sleep 1; ./image_01&
```

Figure 2-1. Tux I Example

2.2. Drawing Images Fom Files

The nano-X API contains a function, `GrDrawImageFromFile()`, which will read images from a file and draw the image onto a window or pixmap. Multiple image formats (GIF, JPEG, BMP, PNG, XPM, PBM, PGM and PPM) are supported by `GrDrawImageFromFile()`. The image type is automatically determined when the file is read.

The image file must reside within the nano-X server's file system. The client application just passes the filename to the server then the server reads the file. This is no problem as long as the the client and server are on the same machine. Another thing to be aware of is that since the server is opening the file, all relative paths in the image file name are relative to the nano-X server's current directory rather than the client's current working directory.

The following example shows how to display Tux as an image loaded from file at run time. The file is read each time an exposure event is received. This approach is rather slow, in the next example we will look at a method to read the file once into memory and draw from memory during the exposure event.

2.2.1. Example Drawing Tux II

Create a directory in which to build this example. Then copy the bitmap image of Tux from the Microwindows sources to this directory. Also copy the example source shown below into a file named "`image_02.c`".

Note: In this example assume that Microwindows is installed to `~/microwin` and the examples are built in `~/mymw/ex_image_02`. If you have different locations then you will have to modify the paths used in the example accordingly.

```
$ cd ~/mymw/ex_image_02
$ cp ~/microwin/src/mwin/bmp/penguin.bmp penguin.bmp
```

Compile the application with the following command.

> **Note:** If you did not install Microwindows you will need to change the path `/usr/include/microwin` so that it points to the include directory where you extracted the Microwindows source.

```
$ gcc image_02.c \
> -I/usr/include/microwin \
> -o image_02 -lnano-X
```

Example 2-2. image_02.c

```
#include <stdio.h>
#define MWINCLUDECOLORS
#include "microwin/nano-X.h"

GR_WINDOW_ID  wid;
GR_GC_ID      gc;

void event_handler (GR_EVENT *event);

int main (void)
{
     if (GrOpen() < 0)
     {
          fprintf (stderr, "GrOpen failed");
          exit (1);
     }

     gc = GrNewGC();

     wid = GrNewWindowEx (GR_WM_PROPS_APPFRAME |
                          GR_WM_PROPS_CAPTION  |
                          GR_WM_PROPS_CLOSEBOX,
                          "Tux Window II",
                          GR_ROOT_WINDOW_ID, 50, 50,
                          100, 200,
```

```
                         WHITE);

     GrSelectEvents (wid, GR_EVENT_MASK_EXPOSURE |
                         GR_EVENT_MASK_CLOSE_REQ);

     GrMapWindow (wid);
     GrMainLoop (event_handler);
     return 0;
}

void event_handler (GR_EVENT *event)
{
     switch (event->type)
     {
     case GR_EVENT_TYPE_EXPOSURE:
     {
          GR_WINDOW_INFO info;

          GrGetWindowInfo (wid, &info);
          GrDrawImageFromFile (wid, gc, 0, 0,
                               info.width, info.height,
                               "penguin.bmp", 0);
          break;
     }
     case GR_EVENT_TYPE_CLOSE_REQ:
          GrClose();
          exit (0);
     }
}
```

Run the example application with the following command.

```
$ nano-X& sleep 1; nanowm& sleep 1; ./image_02&
```

2.3. Loading Images Fom Files

The nano-X API contains a function, `GrLoadImageFromFile()`, which will read an image from a file into the nano-X server's memory and return a GR_IMAGE_ID ID. With this ID the image can be drawn to a window at a later time using the `GrDrawImageToFit()` function. Multiple image formats (GIF, JPEG, BMP, PNG, XPM, PBM, PGM and PPM) are supported by `GrLoadImageFromFile()`. The image type is automatically determined when the file is read.

Just as with the `GrDrawImageFromFile()` function the image file must reside within the nano-X server's file system. The client application just passes the filename to the server then the server reads the file. This is no problem as long as the the client and server are on the same machine. Another thing to be aware of is that since the server is opening the file, all relative paths in the image file name are relative to the nano-X server's current directory rather than the client's current working directory.

The following example shows how to display Tux as an image loaded from file at run time. This approach is a little quicker than the previous example since the program does not go out to disk during each exposure event.

2.3.1. Example Drawing Tux III

Create a directory in which to build this example. Then copy the bitmap image of Tux from the Microwindows sources to this directory. Also copy the example source shown below into a file named "`image_03.c`".

Note: In this example assume that Microwindows is installed to `~/microwin` and the examples are built in `~/mymw/ex_image_03`. If you have different locations then you will have to modify the paths used in the example accordingly.

```
$ cd ~/mymw/ex_image_03
$ cp ~/microwin/src/mwin/bmp/penguin.bmp penguin.bmp
```

Compile the application with the following command.

Note: If you did not install Microwindows you will need to change the path `/usr/include/microwin` so that it points to the include directory where you extracted the Microwindows source.

```
$ gcc image_03.c \
> -I/usr/include/microwin \
> -o image_03 -lnano-X
```

Example 2-3. image_03.c

```
#include <stdio.h>
#define MWINCLUDECOLORS
#include "microwin/nano-X.h"

GR_WINDOW_ID  wid;
GR_GC_ID      gc;
GR_IMAGE_ID   image;

void event_handler (GR_EVENT *event);

int main (void)
{
     if (GrOpen() < 0)
     {
          fprintf (stderr, "GrOpen failed");
          exit (1);
     }

     image = GrLoadImageFromFile ("penguin.bmp", 0);

     gc = GrNewGC();

     wid = GrNewWindowEx (GR_WM_PROPS_APPFRAME |
                          GR_WM_PROPS_CAPTION  |
                          GR_WM_PROPS_CLOSEBOX,
```

```
                              "Tux Window III",
                              GR_ROOT_WINDOW_ID, 50, 50,
                              100, 200,
                              WHITE);

     GrSelectEvents (wid, GR_EVENT_MASK_EXPOSURE |
                              GR_EVENT_MASK_CLOSE_REQ);

     GrMapWindow (wid);
     GrMainLoop (event_handler);
     return 0;
}

void event_handler (GR_EVENT *event)
{
     switch (event->type)
     {
     case GR_EVENT_TYPE_EXPOSURE:
     {
          GR_WINDOW_INFO info;

          GrGetWindowInfo (wid, &info);
          GrDrawImageToFit (wid, gc, 0, 0,
                            info.width, info.height,
                            image);
          break;
     }
     case GR_EVENT_TYPE_CLOSE_REQ:
          GrClose();
          exit (0);
     }
}
```

Run the example application with the following command.

```
$ nano-X& sleep 1; nanowm& sleep 1; ./image_03&
```

Chapter 3. Working With Fonts

3.1. Using TrueType Fonts in Your Application

Microwindows comes with two built in raster fonts. A rather frequent question to the mail list goes something like *"I can only set two font sizes..."*. To gain additional font styles and sizes you could compile additional raster fonts into Microwindows. There are some tools in the `../microwin/src/fonts/` directory that can be used to create additional raster fonts for Microwindows.

This section will not cover these raster font tools. Instead I will describe how to install and use TrueType fonts within Microwindows.

3.2. Installing The FreeType Library

3.2.1. Download The FreeType Library

Microwindows uses the open source FreeType (http://www.freetype.org) library to render TrueType fonts. Microwindows works with version 1.3.1 of the FreeType library. At the time of this article the FreeType library is at version 2.0.3. I do not know if this version works with Microwindows. FreeType version 1.3.1 is known to work and can be downloaded from the Microwindows ftp site (ftp://microwindows.org/pub/microwindows/). The sources come in a tarball named `freetype-1.3.1.tar.gz`. Copy this file into a convenient spot. I copied the file into my home directory `/home/gary`.

3.2.2. Extract The Sources

The tarball is a compressed tar file. You must first uncompress the file using gunzip,

then extract the sources from the tar file using tar. This creates a new directory named `freetype-1.3.1` that contains the FreeType source tree. Type the following commands.

```
$ tar -xzf freetype-1.3.1.tar.gz
```

3.2.3. Build & Install The Library

Change directories to the new FreeType source directory.

```
$ cd freetype-1.3.1
```

Build the FreeType library by typing:

```
$ ./configure
$ make
```

If everything went as planned and the library built without errors then you should install the libraries now. Switch to *root* user id and type:

```
# make install
```

3.2.4. Modify The Microwindows Configuration File

Modify the Microwindows config file `.../microwin/src/config`. Change `HAVE_FREETYPE_SUPPORT` to `Y`. Modify `INCFTLIB` and `LIBFTLIB` to point to the directories that contain the FreeType libraries that you just built. On my system it's as shwon below:

```
#####################################################################
# TrueType font support thru FreeType
#####################################################################
HAVE_FREETYPE_SUPPORT    = Y
INCFTLIB                 = /usr/local/include
```

```
LIBFTLIB                = /usr/local/lib/libttf.so
FREETYPE_FONT_DIR       = "/usr/local/microwin/fonts"
```

Symbol	DescriptionPurpose
HAVE_FREETYPE_SUPPORT	This symbol controls the conditional compilation of the FreeType code within Microwindows. If set to "Y" then FreeType support will be included. If set to "N" the FreeType support is not compiled in.
INCFTLIB	This symbol defines the path to the FreeType include files (on the development system). This path will be added to the include file search path during compilation of Microwindows.
LIBFTLIB	This symbol defines the path to the FreeType library files (on th development system). This path will be added to the include file search path during linking of Microwindows.
FREETYPE_FONT_DIR	This symbol defines the path to the TrueType fonts on the target system. When you setup your target system you will use this directory on the target system to hold the TrueType fonts.

3.2.5. Rebuild Microwindows With FreeType Support

After you modify the config file, re-build and re-install Microwindows.

3.2.6. Get Some TrueType Fonts

Create a directory "/usr/local/microwin/fonts/" on your target machine to hold your TrueType fonts. When you get some TrueType fonts, you will put the `*.ttf` files in this directory.

> **Note:** This directory must match the directory that you specified with the symbol `FREETYPE_FONT_DIR` in your Microwindows configuration file.

You need to pay attention to the license on the TrueType fonts that you plan to use. For quick evaluation purposes you can grab some TrueType fonts from the nearest Windows machine. But you should not use these font on a production system unless you get the proper permissions from the copyright holders. You can get some TrueType fonts from the following sites. If you do a quick seach on the web, dozens of additional sites offering free TrueType fonts will show up.

ftp://microwindows.censoft.com/pub/microwindows/microwindows-fonts-truetype-0.89pre2.tar.gz

http://www.microsoft.com/typography/fontpack/default.htm

3.3. Font Example

In this section I present a simple nano-X TrueType font application.

3.3.1. Example Using Fonts

Copy the source shown below into a file named "`fonts.c`". Compile the application with the following command.

```
$ gcc fonts.c -I/usr/include/microwin \
> -o fonts -lnano-X
```

Example 3-1. fonts.c

```
#include <stdio.h>
#define MWINCLUDECOLORS
#include "microwin/nano-X.h"

GR_WINDOW_ID  wid;
GR_GC_ID      gc;
```

```
GR_FONT_ID    font_a, font_b, font_c, font_d;

void event_handler (GR_EVENT *event);

int main (void)
{
    if (GrOpen() < 0)
    {
        fprintf (stderr, "GrOpen failed");
        exit (1);
    }

    gc = GrNewGC();
    GrSetGCUseBackground (gc, GR_FALSE);
    GrSetGCForeground (gc, RED);

    wid = GrNewWindowEx (GR_WM_PROPS_APPFRAME |
                         GR_WM_PROPS_CAPTION  |
                         GR_WM_PROPS_CLOSEBOX,
                         "Font Test Window",
                         GR_ROOT_WINDOW_ID, 50, 50,
                         200, 130, WHITE);

    GrSelectEvents (wid, GR_EVENT_MASK_EXPOSURE |
                         GR_EVENT_MASK_CLOSE_REQ);

    font_a = GrCreateFont ("arial", 12, NULL);
    font_b = GrCreateFont ("comic", 16, NULL);
    font_c = GrCreateFont ("comic", 24, NULL);
    font_d = GrCreateFont ("arial", 36, NULL);

    GrMapWindow (wid);
    GrMainLoop (event_handler);
    return 0;
}

void event_handler (GR_EVENT *event)
```

```
{
    switch (event->type)
    {
    case GR_EVENT_TYPE_EXPOSURE:
        GrSetGCFont (gc, font_a);
        GrText (wid, gc, 20, 20, "Arial 12",  -1, GR_TFASCII);
        GrSetGCFont (gc, font_b);
        GrText (wid, gc, 20, 40, "Comic 16",  -1, GR_TFASCII);
        GrSetGCFont (gc, font_c);
        GrText (wid, gc, 20, 70, "Comic 24",  -1, GR_TFASCII);
        GrSetGCFont (gc, font_d);
        GrText (wid, gc, 20, 110, "Arial 36",  -1, GR_TFASCII);
        break;

    case GR_EVENT_TYPE_CLOSE_REQ:
        GrClose();
        exit (0);
    }
}
```

Run the example application with the following command. You will see a window appear as shown below.

```
$ nano-X& sleep 1; nanowm& sleep 1; ./fonts&
```

Figure 3-1. TrueType Font Example