СОЗДАЕМ КРОССПЛАТФОРМЕННОЕ ПРИЛОЖЕНИЕ НА ОСНОВЕ FLTK

АНТОН БОРИСОВ

Четкого типажа IT-специалиста не существует, ему часто приходится вторгаться в области деятельности, смежные с его основным профилем. Например, программистам приходится иногда производить конфигурацию системы и выступать в роли администратора. В свою очередь, администраторы в рамках своих полномочий не всегда занимаются лишь администрированием – они умеют составлять программы для своих подзадач. Однако не стоит смешивать административную работу и работу программиста, хотя зачастую на многих предприятиях на сегодняшний день характерна ситуация, когда администратор и швец, и жнец и на дуде игрец.

Однако не буду концентрировать ваше внимание на неправильном подходе к организации IT-работы на предприятии. Замечу, что ситуации, когда необходимо обработать информацию и сформировать из нее новые данные, не являются единичными как для системных инженеров, администраторов, так и для программистов. Сооружать некий айсберг ради кратковременной и нетребовательной задачи я считаю нерациональным решением. Использовать готовые шаблоны также не всегда удобно (затрачивается время на изучение требований к этим шаблонам и т. п.). Кроме того, полученный код хотелось бы использовать на нескольких ОС. Поэтому я остановил свой выбор на продукте, предназначенном для создания кроссплатформенных приложений FLTK (Fast Light ToolKit). Почему он называется именно так, и чем интересен, вы узнаете, ознакомившись с данным материалом.

Краткая предыстория

Изначально пакет разрабатывал Bill Spitzak как удобную и быструю альтернативу существовавшим тогда монстрам. Во время работы в Sun Microsystems и Digital Domain концепция FLTK неоднократно перерабатывалась и дополнялась. Некоторые идеи были заимствованы из NeXT, NeWS, Forms. Само название несколько раз трансформировалось, была даже попытка назвать продукт как FL, но из-за того, что в поисковых машинах слово FL в первую очередь интерпретировалось как обозначение штата Флорида (FL), было решено придумать другое название. В итоге получилось то, что известно под акронимом FLTK (Full-Tick).

Для сегодняшнего обзора необходимо вооружиться следующим инструментарием – самим пакетом FLTK, компилятором gcc, интегрированной средой разработки Eclipse, продуктом для Win32 платформы Microsoft Visual Studio 6.0 и симпатичной кроссплатформенной библиотекой для работы с PNG-графикой – pngwriter. Будем использовать версию 1.1.5 (последняя стабильная ветка) [1].

Что касается компилятора, то надеюсь, что инструментарий разработчика вы уже поставили. Если нет, то следует поставить пакеты gcc и g++. А также пригодится make.

Следует сказать огромное спасибо фирме IBM за интегрированную среду разработки Eclipse. Мало того, что компания предоставляет исходники продукта на всеобщее обозрение, так он еще и предназначен для работы на нескольких аппаратных и программных платформах. Среди них: Windows 98/ME/2000/XP, Linux (x86/Motif, GTK 2), Linux (AMD 64/GTK 2), Solaris 8 (SPARC/Motif), AIX (PPC/Motif), HP-UX (HP9000/Motif), Mac OSX (Mac/Carbon).

Данное ПО основано на Java, поэтому убедитесь, что в вашей системе установлена версия JRE не ниже 1.4. Среди недостатков решений, основанных на Java, стоит отметить жадность к ресурсам и не блещущее быстродействие.

Я использовал при работе Eclipse версии 2.1.3. На момент написания статьи выпущена версия 3.0.1, но для наших ознакомительных целей вполне подойдет и вторая версия. Информация о том, с каких сайтов можно загрузить данное ПО, есть по адресу: http://download.eclipse.org/downloads, http://download.eclipse.org/downloads/drops/R-2.1.3-200403101828/eclipse-SDK-2.1.3-linux-gtk.zip (65 Мб).

Это только основное ядро. Теперь заберем дополнительный пакет для C/C++ разработки. Он называется CDT (C/C++ Development Tool): http://www.eclipse.org/tools/downloads.html, http://download.eclipse.org/tools/cdt/updates/release/dist/cdt-full-1.2-linux-gtk.zip (3 Мб).

Также для второй версии Eclipse существует пакет русификации. Насколько хорошо он переведен, не берусь судить, т.к. причин опробовать его пока не возникало. По умолчанию в Eclipse не используется CDT. Его нужно устанавливать дополнительно. То есть распаковать архив cdt-full-1.2-linux-gtk.zip и положить содержимое из каталогов «plugins» и «features» в установленную директорию с Eclipse. В главной директории Eclipse уже существуют такие каталоги – в них и нужно положить распакованные файлы.

Однако вышел вот какой казус – Eclipse не распознал добавленных расширений по неизвестной для меня причине. Поэтому я поступил следующим образом. Создал символические ссылки на файлы CDT, и вот что получилось в директории features:

```
drwxr-xr-x  org.eclipse.cdt_1.2.1
lrwxrwxrwx  org.eclipse.cdt_2.1.3 -> org.eclipse.cdt_1.2.1
drwxr-xr-x  org.eclipse.cdt.linux.gtk_1.2.1
lrwxrwxrwx  org.eclipse.cdt.linux.gtk_2.1.3 -> org.eclipse.cdt.linux.gtk_1.2.1
drwxr-xr-x  org.eclipse.cdt.linux.motif_1.2.1
lrwxrwxrwx  org.eclipse.cdt.linux.motif_2.1.3 -> org.eclipse.cdt.linux.motif_1.2.1
drwxr-xr-x  org.eclipse.cdt.make_1.2.1
lrwxrwxrwx  org.eclipse.cdt.make_2.1.3 -> org.eclipse.cdt.make_1.2.1
drwxr-xr-x  org.eclipse.cdt.managedbuilder_1.2.1
lrwxrwxrwx  org.eclipse.cdt.managedbuilder_2.1.3 -> org.eclipse.cdt.managedbuilder_1.2.1
drwxr-xr-x  org.eclipse.cdt.qnx.photon_1.2.1
lrwxrwxrwx  org.eclipse.cdt.qnx.photon_2.1.3 -> org.eclipse.cdt.qnx.photon_1.2.1
drwxr-xr-x  org.eclipse.cdt.solaris.motif_1.2.1
lrwxrwxrwx  org.eclipse.cdt.solaris.motif_2.1.3 -> org.eclipse.cdt.solaris.motif_1.2.1
drwxr-xr-x  org.eclipse.cdt.source_1.2.1
lrwxrwxrwx  org.eclipse.cdt.source_2.1.3 -> org.eclipse.cdt.source_1.2.1
drwxr-xr-x  org.eclipse.cdt.win32_1.2.1
lrwxrwxrwx  org.eclipse.cdt.win32_2.1.3 -> org.eclipse.cdt.win32_1.2.1
drwxrwxr-x  org.eclipse.jdt_2.1.3
drwxrwxr-x  org.eclipse.jdt.source_2.1.3
drwxrwxr-x  org.eclipse.pde_2.1.3
drwxrwxr-x  org.eclipse.platform_2.1.3
drwxrwxr-x  org.eclipse.platform.linux.gtk_2.1.3
drwxrwxr-x  org.eclipse.platform.linux.gtk.source_2.1.3
drwxrwxr-x  org.eclipse.platform.source_2.1.3
drwxrwxr-x  org.eclipse.sdk.linux.gtk_2.1.3
```

Аналогичным образом следует поступить и для директории plugins. При таком варианте уже возможно создание проекта на C/C++.

Что касается графики, то я решил выбрать pngwriter по следующим причинам: отсутствует проблема с лицензированием, формат PNG на сегодняшний день широко распространен, и понравились функции для работы с графикой в данном пакете.

Документация, идущая с pngwriter [2], достаточно широко освещает аспекты сборки как под Linux, так и под Win32. С другими ОС проблем не будет, если в системе есть компилятор: http://heanet.dl.sourceforge.net/sourceforge/pngwriter/pngwriter-0.5.0.tgz (640 Кб).

Также, надеюсь, у вас не возникнет проблем с Microsoft Visual Studio 6.0.

Теперь главный герой сегодняшнего эпоса – FLTK.

```
tar xzvf fltk-1.1.5-source.tar.bz2
cd fltk-1.1.5
./configure --prefix=/usr/local/fltk-1.1.5 --enable-xft ↵
    --enable-threads
```

Если мультипоточность в FLTK не нужна, то можно не использовать опцию «--enable-threads». То же самое касается и улучшенной поддержки шрифтов (антиалиасинг) – опция «--enable-xft».

```
make && make install
```

В итоге получились следующие библиотеки в директории /usr/local/fltk-1.1.5/lib:

```
-rw-r--r--  libfltk.a
-rw-r--r--  libfltk_forms.a
lrwxrwxrwx  libfltk_forms.so -> libfltk_forms.so.1.1
-rwxr-xr-x  libfltk_forms.so.1.1
-rw-r--r--  libfltk_gl.a
lrwxrwxrwx  libfltk_gl.so -> libfltk_gl.so.1.1
-rwxr-xr-x  libfltk_gl.so.1.1
-rw-r--r--  libfltk_images.a
lrwxrwxrwx  libfltk_images.so -> libfltk_images.so.1.1
-rwxr-xr-x  libfltk_images.so.1.1
lrwxrwxrwx  libfltk.so -> libfltk.so.1.1
-rwxr-xr-x  libfltk.so.1.1
```

Сборка pngwriter еще проще.

```
tar xzvf pngwriter-0.5.0.tgz
cd pngwriter
make && make install
```

Пакет устанавливается в /usr/local/lib/ и в /usr/local/include. Для более подробной информации обращайтесь к файлу README из этого пакета.

Теперь немного забежим вперед и создадим наше первое приложение на FLTK. В первую очередь следует обращаться к демо-программам, которые скомпилировались вместе с самим пакетом FLTK. Их можно найти в директории fltk-1.1.5/test. Написано доступно и просто. Тем, кто сталкивался с программированием на MFC или на QT/GTK, будет понятна идеология callback (т.е. реакция на события). В частности, в QT используется аналог системы callback – пара сигнал/слот. Фактически это более гибкое решение (нежели реальные callback-функции), которое позволяет любому объекту присвоить свойство реагировать на сигнал. В GTK также используется система callback, так называемая пара signal/callback. Более полную информацию можно почерпнуть из [7-10].

Если нужен справочник классов и методов, то fltk-1.1.5/documentation – хороший источник получения информации. В принципе на самом сайте FLTK можно забрать справочник в PDF-формате.

Рассмотрим самый простой файл на FLTK.

```
My_CPP_Test.cpp

//  HEADERS
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Menu_Bar.H>
#include <FL/Fl_File_Chooser.H>

//      GLOBALS
Fl_Menu_Bar             *m;
Fl_File_Chooser         *fc;

#define    FONT_SIZE     14

void fc_callback(Fl_File_Chooser *fc, void *data)
{
       if(fc->value())
       {
          printf( "Chosen file: \"%s\"\n", fc->value() );
       }

    return;
}

void open_cb(Fl_Widget*, void*)
{
       fc->callback(fc_callback);
       fc->show();

       while(fc->shown())
          Fl::wait();

       if( fc->count() == 1 )
       {
          printf("File was selected\n");
       }

}

void quit_cb(Fl_Widget*, void*)
{
    exit(0);
}

Fl_Menu_Item MenuEng[] = {

    { "&File",     0, 0, 0, FL_SUBMENU },
      { "&Open",   FL_CTRL + 'o', (Fl_Callback *)open_cb },
      { "&Save",   FL_CTRL + 's', 0 },
      { "E&xit",   FL_CTRL + 'q', (Fl_Callback *)quit_cb, 0 },
      { 0 },

    { "&Help",     0, 0, 0, FL_SUBMENU },
      { "&About", FL_CTRL + 'a' , 0 },
      { 0 },
    { 0 }
};

int main(int argc, char **argv) {

    Fl_Window *window = new Fl_Window(300, 300, "Sample");
    window->color(FL_WHITE);

    m = new Fl_Menu_Bar(0, 0, 640, 25);
    m->copy(MenuEng);
    m->box(FL_UP_BOX);

    m->textcolor(FL_BLUE);
    m->textfont(FL_TIMES);
    m->textsize(FONT_SIZE);

    fc = new Fl_File_Chooser(".", "*.{txt,cpp}", ↵
           Fl_File_Chooser::SINGLE, "File_Chooser_Dialog");

    window->show();
    window->callback( (Fl_Callback *)quit_cb, window );

    return Fl::run();

}
```

Кто есть кто в данном примере? Подключаются только необходимые файлы из комплекта FLTK. В нашем случае – это Fl.H, Fl_Window.H, Fl_Menu_Bar, Fl_File_Chooser.H. Первый файл необходимо включать при создании любого приложения на FLTK, второй отвечает за создание объекта Fl_Window *window. Понятно, что для создания объекта Fl_Menu_Bar *m мы подключили FL_Menu_Bar. Идеология достаточно понятна на данном этапе.

Далее мы объявляем глобальные переменные m, fc – доступ к классам этих объектов будет производиться не только из функции main().

Следующим шагом идет объявление callback-функций. Они отвечают за реакцию на события. В нашем примере fc_callback(), open_cb() участвуют в процессе открытия файлового диалога, а функция quit_cb() отвечает за завершение всего приложения.

Далее мы заполняем структуру меню.

```
    {"&File",      0, 0, 0, FL_SUBMENU },
      { "&Open",   FL_CTRL + 'o', (Fl_Callback *)open_cb },
      { "&Save",   FL_CTRL + 's', 0 },
      { "E&xit",   FL_CTRL + 'q', (Fl_Callback *)quit_cb, 0 },
      { 0 },
```

В частности данный элемент означает следующее – создается пункт в меню под названием «File». При этом быстрый вызов осуществляется по клавише <Alt+F> (на что указывает знак амперсанда – «&File»). В состав пункта «File» входят следующие подпункты – «Open», «Save», «Exit». Горячая клавиша указывается в шаблоне вида FL_CTRL + «X», где «X» – название клавиши.

В данном случае при нажатии на клавишу <Ctrl+O> будет вызвана callback-фунция open_cb(). Если на месте callback-функции «0», то никакая функция не привязана к данному событию.

Переходим к функции main(). В её теле мы создаем объект окна Fl_Window *window. Инициализируем его с параметрами (300, 300, «Sample») – соответственно ширина, высота, заголовок окна. Также создается и инициализируется

объект типа меню. В этом объекте изменяются тип шрифта и его размер. Инициализируем объект Fl_File_Chooser.

Показываем созданное творение – window → show(). Привязываем callback-фунцию quit_cb() на закрытие приложения. И наконец вызываем обработчик событий для всего созданного приложения – return Fl::run().

Если некоторые параметры остались для вас непонятными – обращайтесь к документации. Вполне возможно, что в версии, которую вы будете использовать, будут изменены некоторые параметры (это характерно для следующей версии FLTK 2.0).

Оформим данный файл как My_CPP_Test.cpp. И создадим make-файл для компиляции программы.

```
Makefile

FLTK=/usr/local/fltk/bin/fltk-config
OPTIONS=--compile

My_CPP_Test:
    $(FLTK)  $(OPTIONS)  My_CPP_Test.cpp
clean:
    rm My_CPP_Test
rebuild:
    make clean; make My_CPP_Test
```

Теперь, запустив команду make, вы скомпилируете это тестовое приложение. У меня оно получилось размером примерно в 400 Кб. Посмотрим на него чуточку внимательнее.

```
bash-2.05b$ file My_CPP_Test
My_CPP_Test: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), not stripped
```

Уберем лишнюю отладочную информацию и посмотрим, от каких библиотек зависит наше новое приложение.

```
bash-2.05b$ strip My_CPP_Test
```

В итоге приложение теперь «похудело» до 240 Кб.

```
bash-2.05b$ ldd My_CPP_Test
        libm.so.6 => /lib/libm.so.6 (0x40025000)
        libXext.so.6 => /usr/X11R6/lib/libXext.so.6 (0x40049000)
        libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x40057000)
        libc.so.6 => /lib/libc.so.6 (0x4011e000)
        libdl.so.2 => /lib/libdl.so.2 (0x40254000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

Достаточно симпатично, на мой взгляд. Вполне вероятно, что удастся запустить даже на дистрибутивах Linux, основанных на ядре 2.0.X, не говоря про современные.

Что и было протестировано на дистрибутиве SUSE Linux с ядром версии 2.2.18 и сопутствующими библиотеками (libc-2.1.3, ld-2.1.3). Радужные ожидания растаяли при запуске программы ldd:

```
bash-2.05b$ ldd -vr My_CPP_Test
        libm.so.6 => /lib/libm.so.6 (0x40016000)
        libXext.so.6 => /usr/X11R6/lib/libXext.so.6 (0x40033000)
        libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x4003f000)
        libc.so.6 => /lib/libc.so.6 (0x400e0000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
My_CPP_Test: /lib/libc.so.6: version `GLIBC_2.2.4' not found
(required by My_CPP_Test)
My_CPP_Test: /lib/libc.so.6: version `GLIBC_2.3' not found
(required by My_CPP_Test)
symbol __ctype_tolower_loc, version GLIBC_2.3 not defined in file
libc.so.6 with link time reference (My_CPP_Test)
symbol dl_iterate_phdr, version GLIBC_2.2.4 not defined in file
libc.so.6 with link time reference (My_CPP_Test)
symbol __ctype_b_loc, version GLIBC_2.3 not defined in file
libc.so.6 with link time reference (My_CPP_Test)
```

Как видите, некоторые символы дистрибутива того времени (4 года назад) не были определены, поэтому я вижу 2 варианта решения проблемы: собирать программу полностью со статическими библиотеками либо собирать FLTK на основе старого дистрибутива и там же компилировать. Насколько это удобно в каждом конкретном случае, решать всё-таки разработчику.

Итак, мы знаем, как написать программу, но по мере накопления дополнительных свойств она разрастается, и встает вопрос удобства и функциональности дальнейшей разработки. В этой связи удобнее будет использовать IDE Eclipse.

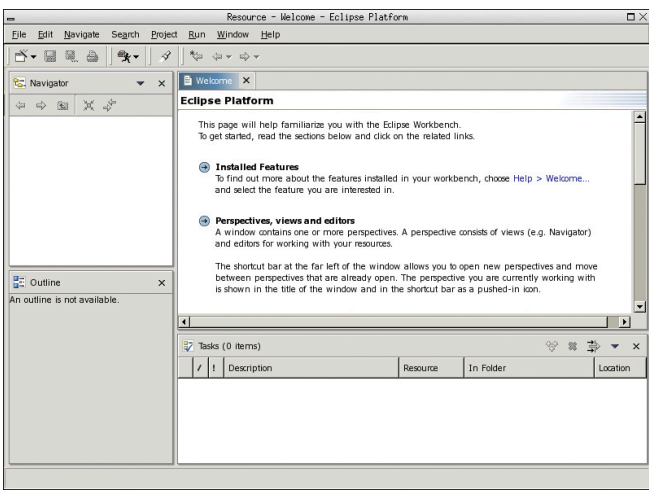

Рисунок 1. Общий вид Eclipse

Для создания проекта на C или C++ следует поступить следующим образом – выбрать меню «Window → Open Perspective → C/C++ Development».

Тем самым мы настраиваем в Eclipse подсветку синтаксиса, характерную для языковых конструкций C/C++.

Создадим пробный проект. «File → New → Project».

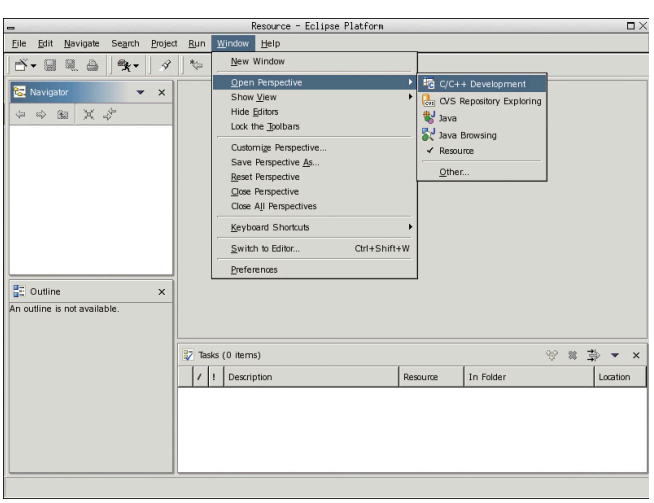

Рисунок 2

Выбираем создание проекта на C++ категории Managed Make C++ Project (см. рис. 3).

Выбираем имя для проекта «My_Test» (см. рис. 4).

А затем придумываем имя для главного файла проекта (см. рис. 5).

Проект сконструирован, файл My_CPP_Test.cpp создан, но он пуст и находится в следующей директории ~/eclipse/workspace.

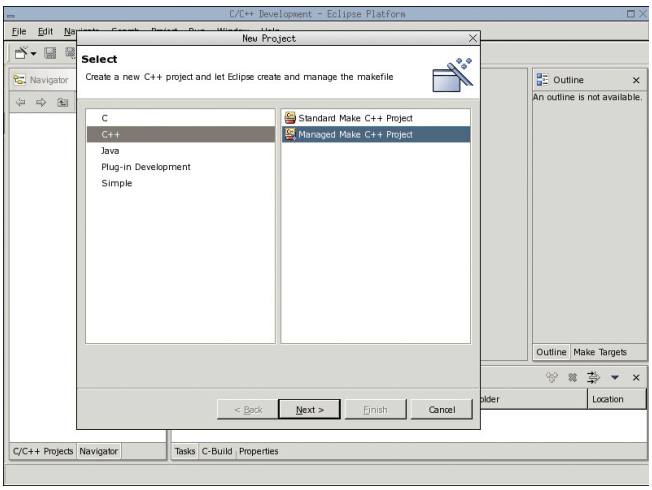

Рисунок 3

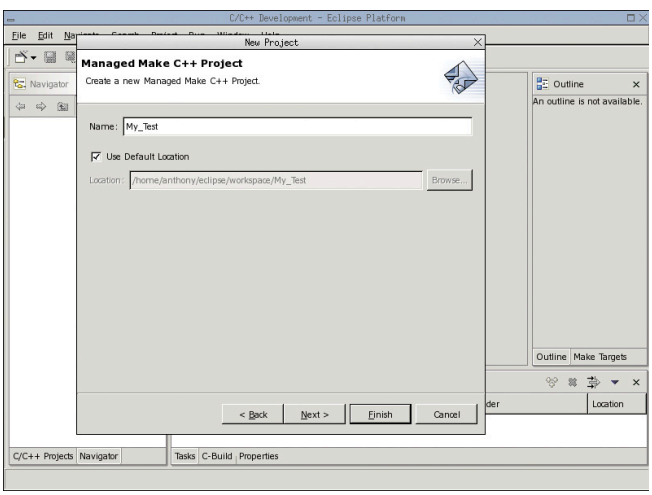

Рисунок 4

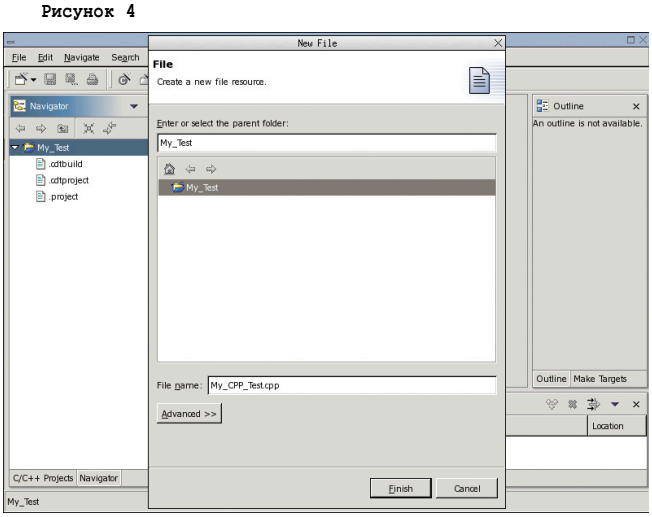

Рисунок 5

Файл My_CPP_Test.cpp создался пустым, поэтому либо скопируем в буфер содержимое нашего примера и вставим из буфера в этот файл. Либо перенесем файл примера в директорию нашего проекта. Eclipse высветит предупреждение о том, что содержимое файла изменилось.

Отлично, теперь в настройках проекта нам следует указать, где искать заголовочные файлы (в частности, где расположены заголовочные файлы для FLTK).

Заходим «Project → Properties». Заполняем поля, указанные на рис. 6 и 7.

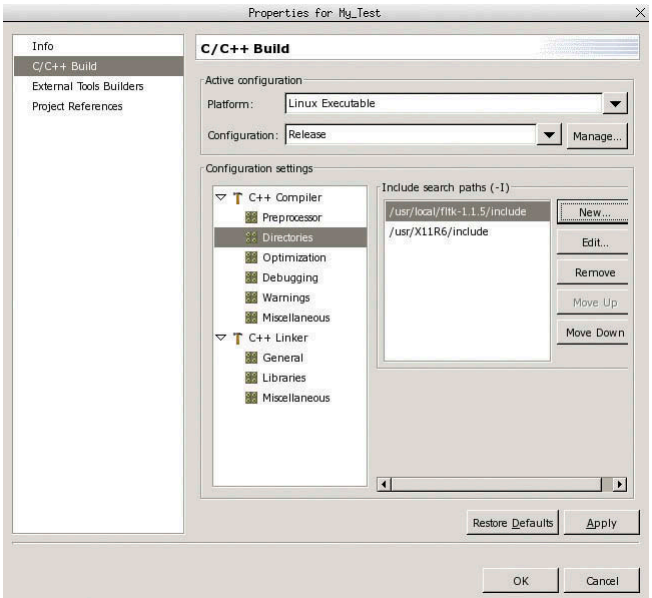

Рисунок 6

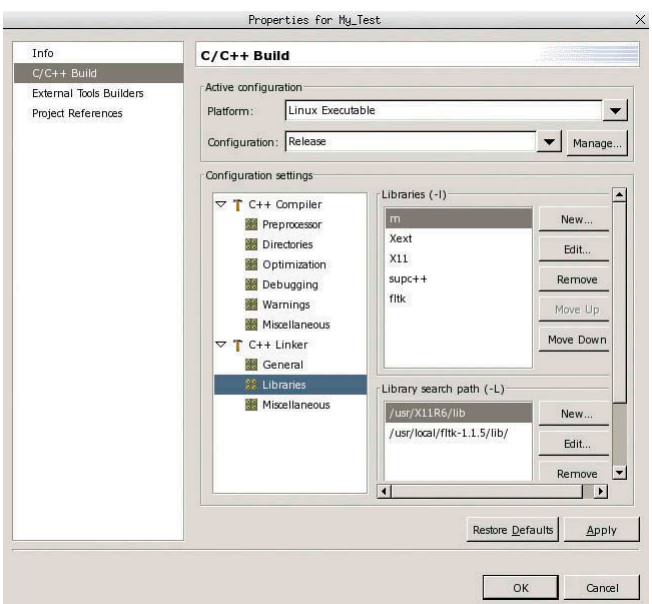

Рисунок 7

Спрашивается, откуда автор знает, какие именно библиотеки следует использовать. Очень просто. Необходимо запустить программу из комплекта FLTK под названием fltk-config со следующими флагами: «--cflags», «--ldflags» («--ldstaticflags»). В итоге при сборке проекта («Project → Rebuild All») вы должны увидеть показанное на рис. 8.

Для того чтобы запустить новое приложение из интегрированной среды, необходимо настроить пункт в меню «Run» (см. рис. 9). В поле «C/C++ Application» указать исполняемый файл (см. рис. 10).

Так как сборка производится не со статической библиотекой, а с динамической, то надо указать путь к библиотеке libfltk.so (см. рис. 11). В итоге сборка и дальнейшая разработка вашего продукта приобретает более удобные очертания.

Теперь добавим поддержку для pngwriter.

Добавим в My_CPP_Test.cpp следующие строки:

```
#include        <pngwriter.h>
```

```
void save_cb(Fl_Widget*, void*)
{

    pngwriter iris(300,300,0,"iris.png");

    for(int a = 1; a < 301; a++)
    for(int b = 1; b < 301; b++)
      iris.plotHSV( a, b, double(a)/500.0, double(b)/500.0, 1.0);

    iris.setgamma(0.7);
    iris.close();

}
```

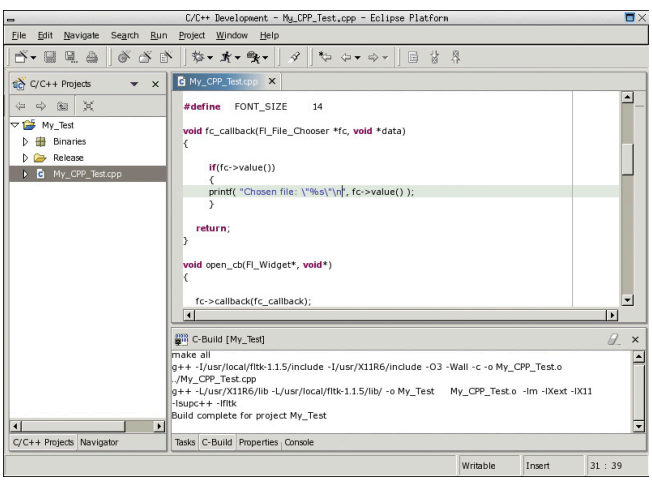

Рисунок 8

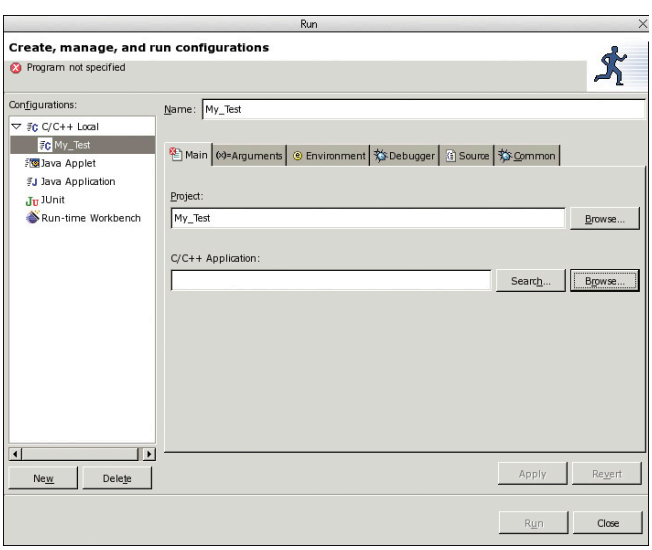

Рисунок 9

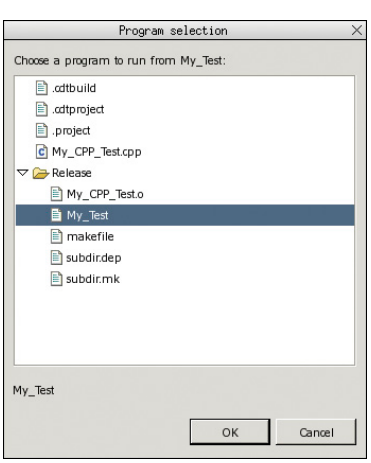

Рисунок 10

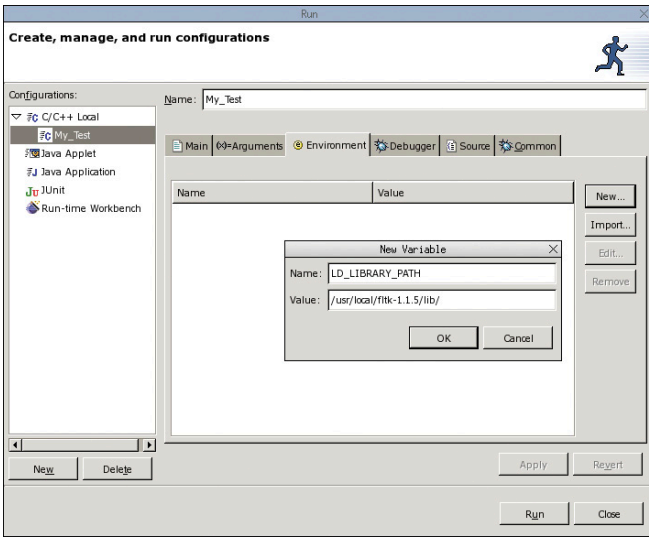

Рисунок 11

Потребуется для объявления функций библиотеки pngwriter. Файл фактически находится в /usr/local/include/, поэтому в Makefile надо будет добавить следующую конструкцию:

```
-I/usr/local/include -L/usr/local/lib
```

Добавляем новую процедуру сохранения файла PNG-формата. И меняем структуру меню.

```
{ "&Save", FL_CTRL + 's', (Fl_Callback *)save_cb },
```

Теперь осталось изменить Makefile, т.к. pngwriter использует C++ синтаксис. Новый Makefile выглядит следующим образом.

```
FLTK=/usr/local/fltk/bin/fltk-config

INCLUDE=`$(FLTK) -cflags`
INCLUDE2=-I/usr/local/include -L/usr/local/lib
LIBS=`$(FLTK) --ldstaticflags`
LIBS2=-lm -lXext -lX11 -lsupc++ -lpng -lpngwriter -lz ↵
    -lfreetype
FLAGS=-O3 -Wall -Wno-deprecated
FLAGS2=-lsupc++
CC=g++

My_CPP_Test:
    $(CC) $(INCLUDE) $(INCLUDE2) ↵
          -o My_CPP_Test My_CPP_Test.cpp $(LIBS) $(LIBS2)
clean:#
    rm My_CPP_Test

rebuild:
    make clean; make My_CPP_Test
```

Библиотека freetype потребовалась из-за того, что сам FLTK собрался с данной поддержкой. На вашей машине может быть по-другому.

Предлагаю переключить внимание на Windows и собрать пакет FLTK с использованием компилятора от Microsoft. Смысла компилировать с помощью Microsoft Visual Studio .NET я не увидел и решил воспользоваться предыдущей версией 6.0.

Приступим. С помощью WinRAR распакуем архив. Запускаем файл проектов fltk.dsw (он находится в каталоге fltk-1.1.5/visualc) в Microsoft Visual Studio. Выбираем для сборки нужный проект fltk (см. рис. 12).

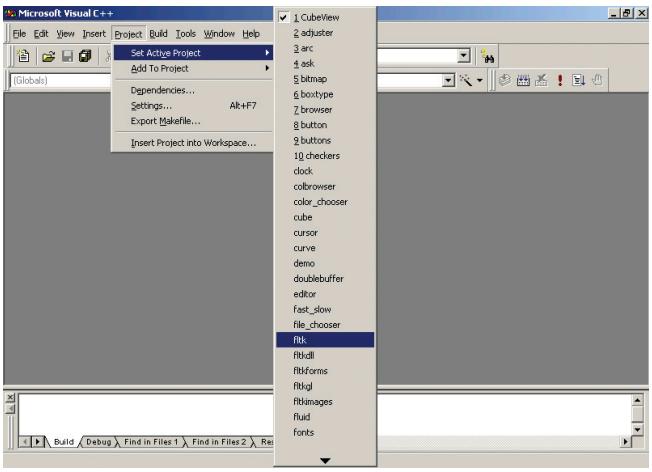

Рисунок 12

Если мы хотим создать DLL-библиотеку, то следует выбрать проект fltkdll. В этом случае программа после компиляции будет меньше, но при этом потребуется внешняя DLL-библиотека fltkdll.dll. Можно поступить другим образом – собрать LIB-библиотеку. При этом необходимый код будет включен в вашу программу при компиляции. И соответственно программа будет состоять фактически из одного EXE-файла. Следующим шагом нужно определить, какой тип библиотеки необходим – с отладочными символами («Debug») или без отладочных символов («Release»). Допустим, мы хотим собрать без отладочных символов, поэтому выбираем тип библиотеки «Release»:

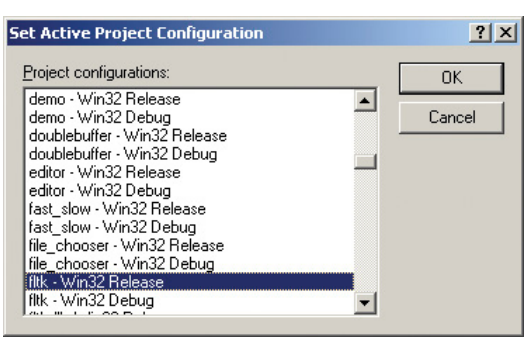

Рисунок 13

Запускаем процесс сборки. Конечный результат в виде файла fltk.lib смотрите в каталоге fltk-1.1.5/lib:

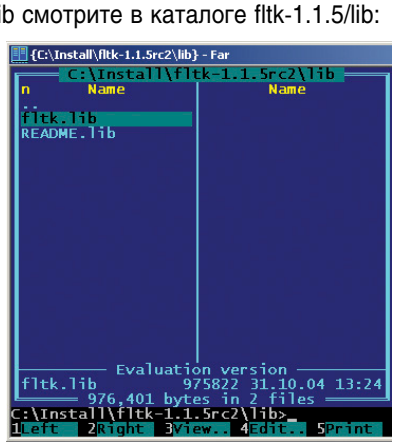

Рисунок 14

Компилировать библиотеку следует как статическую (расширение файла – .LIB). Полученный итог – файлы pngwriter.lib, zlib.lib, png.lib.

Полученные библиотеки необходимо положить в Program Files\Microsoft Visual Studio\VC98\Lib, а заголовочные файлы от FLTK следует положить в Program Files\Microsoft Visual Studio\VC98\Include (предварительно создав там директорию FL).

В общем-то, среда готова для написания программ.

Как именно вы организуете представление графической информации – дело вашего вкуса.

Импорт графических файлов с помощью FLTK возможен для файлов JPEG, PNG, BMP, XPM. Экспортировать в формате PNG можно в том случае, если вы подключите библиотеку libpngwriter.

На основе FLTK развилось несколько проектов. Это и расширение FLTK – проект EFLTK, который умеет работать с базами данных MySQL. Это и SPTK, разрабатываемый, кстати сказать, нашим соотечественником Алексеем Паршиным [5]. С помощью SPTK вы получаете доступ к данным как в Excel-, Access-форме, так и к MySQL-данным. FLU, обладающий некоторыми расширенными опциями для построения виджетов [6].

Среди достоинств FLTK хочу подчеркнуть особенность ясного и понятного построения кода программ. За счет простоты достигаются скорость составления каркаса программы и дальнейшая скорость её выполнения. В некоторой степени программы на основе FLTK являются «мобильными», т.к. вы четко представляете, от каких системных библиотек зависит FLTK. Поэтому разобраться, что требуется для запуска программы в новых условиях, получается намного быстрее, нежели, например, на основе QT.

Характерный минус разработки на основе FLTK заключается в его же простоте – некоторых конструкций, которые есть в других средствах разработки, просто нет. Или же они находятся на такой стадии разработки, когда продукт называется unstable (в частности, поддержка UTF8 всё еще находится на стадии разработки).

В целом FLTK можно предлагать как продукт для получения начального опыта программирования GUI-приложений. В частности, примеры, расположенные по адресам [11], [12], [13], помогут разобраться в составлении программ на FLTK.

Надеюсь, данный опыт покажет вам дальнейшие пути для поиска необходимого инструментария.

Ссылки:

1. http://www.fltk.org/software.php?SOFTWARE=v1_1
2. http://pngwriter.sourceforge.net
3. http://pngwriter.sourceforge.net/howto_msvc.php
4. http://yaroslav-v.chat.ru
5. http://sptk.tts-sf.com/index.php
6. http://www.osc.edu/~jbryan/FLU/
7. http://www.cs.wisc.edu/~cs559-1/tutorials/tutorial4.html
8. http://ib.cnea.gov.ar/~poo/gtkmm-devel-1.2.10/docs/tutorial/gtkmm-tut-html/tutorial-2.html
9. http://inti.sourceforge.net/tutorial/libinti/advanced eventand signalhandling.html
10. http://www.jtz.org.pl/Inne/QT-Tutorial/signalsandslots.html
11. http://www.cs.virginia.edu/~gfx/Courses/2004/Intro.Spring.04/ProgrammingAssignments/Exercise1
12. http://www.soe.ucsc.edu/classes/cmps160/Spring04
13. http://www.cs.wisc.edu/~cs559-1