Subject: Mac OS X port architecture
Posted by daveremba on Sun, 17 Jul 2011 08:02:24 GMT
View Forum Message <> Reply to Message

Step A. Complete and test theide under X11 on MacOSX that is able to build itself.

Manual makefile exists now.
Need to try the makefile generation.
Building the examples have a visual problem:
flat buttons and garbage pixels under menus

PROPOSED:

Step B. Either use theide or Xcode on Mac to build up a Cocoa bridge into UPP framework.

n.b. Carbon lib with C++ direct linking still exists for 32 bit apps, but I think we want to support 64 bit arch, and this will require using Objective-C
and Cocoa.

For step B:

Here is how I think the Cocoa/MacOSX port would be architected, after I looked at wxWidgets and Firefox source:

1. For the Cocoa/MacOSX port, in CtrlCore, there will be new files like:

DrawCocoa.cpp
DrawWinCocoa.cpp
DrawTextCocoa.cpp
...

and
CocoaApp.cpp
CocoaProc.cpp
CocoaWnd.cpp
...

(as there is currently DrawX11... and DrawWin32... etc)

In particular: the top-level window is created through Cocoa, the App and Window are registered with MacOSX, events are captured from Cocoa and directed into UPP, drawing of specific buttons etc. are redirected out through Cocoa, font information obtained through Cocoa, etc.

2. Also in CtrlLib there will be new .m or .mm files for gcc to process; these are Objective-C source files that are part of the MacOS build that get linked into Theide. These files connect the DrawCocoa... files above into MacOSX Cocoa via the object messaging system.

n.b. It is likely that the new .cpp files will be small (or maybe not even needed) as most processing

could done inside the .m files. This is TBD.

The build step for #2 cannot be done on a Linux machine AFAIK, because there are Apple-specific headers and libs that are needed (hence 'legal').

n.b. Xcode doesn't seem to show the actual flags to gcc, but there should be a gcc command line equivalent that could be added to theide,
thus eliminating the need for Xcode to build the Objective-C and Cocoa parts. This is TBD.

3. The last step of generating an installable image is optional, and would give UPP a polished functionality so it installs like an .MSI or RPM; but is not essential. Step #3 not possible to generate on Linux AFAIK, because the .dmg disk
image file is MacOSX specific.

TBD: how to make installable disk image as a nightly build

4. These new files (in #1 and #2) will be switched on by a new flag, PLATFORM_MACOSX.

The existing flag PLATFORM_OSX11 will remain, and means to build a MacOSX app using the X11 emulator (this is the step A above).

n.b. Or PLATFORM_OSX11 could mean to use Cocoa, and the X11 version might be phased out, but I think it will have value for a while until the Cocoa port is completed, tested, accepted, etc.

Recommendations:

1. The Mac build of UPP does need to occur on a machine that runs true MacOSX (virtually or physically).

2. New files will need to be added to CtrlCore, as well as some minor patches to existing files.

note: this post was treated as a separate topic from a related post on the backend development process: architecture meaning what to do, process meaning how to do it.
But, I did mix both topics in this message.

-Dave

Subject: Re: Mac OS X port architecture
Posted by mirek on Sun, 17 Jul 2011 21:26:25 GMT
View Forum Message <> Reply to Message

daveremba wrote on Sun, 17 July 2011 04:02
1. For the Cocoa/MacOSX port, in CtrlCore, there will be new files like:

Actually, for macosx backend development, use rainbow. Rainbow basically makes possible to develop GUI backend without changing CtrlCore.

Mirek

Subject: Re: Mac OS X port architecture
Posted by daveremba on Tue, 19 Jul 2011 00:11:28 GMT
View Forum Message <> Reply to Message

OK, yes, it does look much easier and a lot
less code, and platforms are separated!

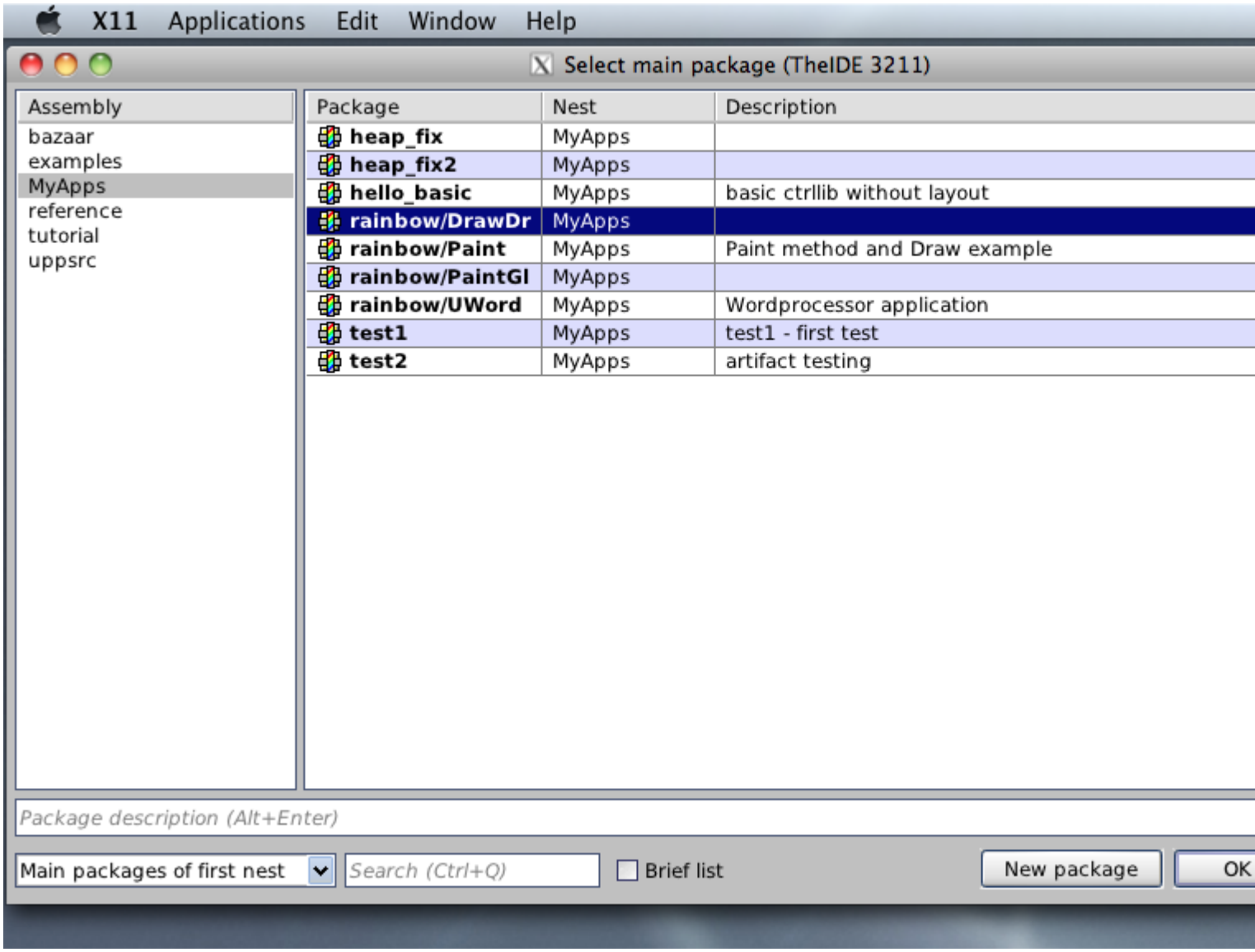
Dave

Subject: Re: Mac OS X port architecture
Posted by daveremba on Tue, 19 Jul 2011 00:18:00 GMT
View Forum Message <> Reply to Message

screenshot of Rainbow development started
using theide on MacOSX

File Attachments

1) theide_rainbow.png, downloaded 2258 times



Subject: best way to draw text/fonts on MacOS from C/C++
Posted by daveremba on Sun, 24 Jul 2011 06:37:09 GMT
View Forum Message <> Reply to Message

Apparently not all graphics calls need to be
done using Objective-C/C++ on Apple.

Here are some relevant calls for fonts, in C/C++:

CTFontCreateWithName
CTFontGetGlyphsForCharacters
CTFontGetAdvancesForGlyphs

then, to draw, one uses the Core Graphics library,

also callable in C/C++ (drawing the line writes the
text characters into the framebuffer):

CGContextSetTextMatrix
CGContextSetTextPosition
CGContextSetShouldAntialias
CTLineDraw

The CT... functions are part of CoreText.

The CG... functions are part of Quartz 2D API.

According to Apple, one is better off drawing text
not using Quartz, but instead the Core Text
functions:
"Quartz 2D provides a limited, low-level interface for
drawing text encoded in the MacRoman text encoding and for
drawing glyphs"
The code technique above is used in FLTK
(see below) and mixes Quartz (CG)/CT.

In a C or C++ file one can access CG and CT by including
the ApplicationServices header and library (and these
functions are directly callable from C/C++).

Regarding the Core Graphics calls, Apple says:
"You can obtain a graphics context by using Quartz graphics
context creation functions or by using higher-level functions
provided in the Carbon, Cocoa, or Printing frameworks. Quartz
provides creation functions for various flavors of Quartz
graphics contexts including bitmap images and PDF. The Cocoa
framework provides functions for obtaining window graphics
contexts. "

Modern x86_64 apps apparently need to use Cocoa
only for non-drawing functions, and to create the
graphics context and associate it with a window
(and this must be done in Objective-C/C++).
The .mm file and -framework Cocoa
needs to be passed into gcc, maybe as a post build
step in UPP. I will try this out.

n.b. Quartz 2D does not seem to have the MacOS functions for
kybd, mouse, etc. (I will check; maybe that was in Carbon).

So for UPP on MacOS: text/font drawing into a
non-X11 (native) window could be
implemented now with the existing C/C++ tools

(after the window is created).

--

This info I learned from examining the Fast Light
Toolkit (FLTK), which I've worked with and
enjoyed using. They've ported it to MacOSX.

I think FLTK gets the Mac OS port done in a simpler and smaller amount
of code than Firefox, wx, or Tk/tcl:

fl_cocoa.mm window, event, kybd, mouse, dnd code
Objective-C++ (3500 lines)
mac.H header mapping FLTK to mac functions
fl_font_mac.cxx font/text info & drawing code in C++

FLTK does not use any high-level widgets on any platform;
it works a lot more like UPP, and draws itself whatever
is needed from basic 2D elements: polylines/polygons,
images/pixmaps, and text/fonts. So, for these
reasons I recommend taking a look at it.

Dave

Subject: Re: best way to draw text/fonts on MacOS from C/C++
Posted by mirek on Sun, 24 Jul 2011 08:06:02 GMT
View Forum Message <> Reply to Message

Quote:
Modern x86_64 apps apparently need to use Cocoa
only for non-drawing functions, and to create the
graphics context and associate it with a window
(and this must be done in Objective-C/C++).
The .mm file and -framework Cocoa
needs to be passed into gcc, maybe as a post build
step in UPP. I will try this out.

Actually, I believe we should rather teach theide to recognize .m/.mm files (should be simple) and put -framework options to linker step (can be done in Package oraganizer now).

We should bind adding -framework and the whole conditional compilation for MACOSX as "MACOSX" flag. Later, it will be "host flag" added automatically (just like WIN32 or LINUX are now).

So I see this as next step:

- investigate how minimal Cocoa application looks like - it is enough to display "Hello world" in otherwise emtpy window (view?), preferably without the presence of .nib files

- this can be developed and tested in XCode

- then make theide compile this app

Quote:

fl_cocoa.mm window, event, kybd, mouse, dnd code
Objective-C++ (3500 lines)
mac.H header mapping FLTK to mac functions
fl_font_mac.cxx font/text info & drawing code in C++

FLTK does not use any high-level widgets on any platform;
it works a lot more like UPP, and draws itself whatever
is needed from basic 2D elements: polylines/polygons,
images/pixmaps, and text/fonts. So, for these
reasons I recommend taking a look at it.

Dave

This is a good find. Looking at FLTK might be a good kickstart...

Mirek

Subject: Re: best way to draw text/fonts on MacOS from C/C++
Posted by daveremba on Sun, 24 Jul 2011 23:34:02 GMT
View Forum Message <> Reply to Message

Ok, I will post a sample hello-world
using C/C++ main & cocoa here.
I'll include a cmd-line build script
for gcc also.

Dave

Subject: Re: best way to draw text/fonts on MacOS from C/C++
Posted by daveremba on Tue, 26 Jul 2011 04:11:45 GMT
View Forum Message <> Reply to Message

Regarding:
Quote:- investigate how minimal Cocoa application looks like -
it is enough to display "Hello world" in otherwise emtpy window
(view?), preferably without the presence of .nib files
1. build the minimal app is easy, gcc on Mac knows
how to handle and link the file types without
any unusual flags (although one can get fancy and
specify architectures for "universal binaries" etc.)

The following command works, to mix a C main file
have it start an objective-C Cocoa application:

gcc main.c main.m cocoa_test2AppDelegate.m -framework Cocoa -o
cocoa_test2_bundle.app/Contents/MacOS/cocoa_test2
theide code editor will load the .m files after I set the
filter to All Files.

2. The template code generated in Xcode sets up Cocoa
so it does expect to have a .nib file (if one deletes it
the app will not display a GUI).

One can however create a toplevel window manually and
handle the events for it. (Apple provides sample code
to do this for legacy Carbon apps, but we won't use it).

FLTK (and other modern GUI frameworks) create the
toplevel and handle events in Objective-C calling Cocoa.
One subclasses from NSView to create the window,
and one subclasses from NSObject to create
a delegate class to handle the events.
I have not finished looking the Fl_Cocoa.mm file yet.

One can access the FLTK/Cocoa bridge online. That is what I am looking at.

Dave

Subject: Re: best way to draw text/fonts on MacOS from C/C++
Posted by mirek on Tue, 26 Jul 2011 05:11:31 GMT

daveremba wrote on Tue, 26 July 2011 00:11Regarding:
Quote:- investigate how minimal Cocoa application looks like -
it is enough to display "Hello world" in otherwise emtpy window
(view?), preferably without the presence of .nib files
1. build the minimal app is easy, gcc on Mac knows
how to handle and link the file types without
any unusual flags (although one can get fancy and
specify architectures for "universal binaries" etc.)

The following command works, to mix a C main file
have it start an objective-C Cocoa application:

gcc main.c main.m cocoa_test2AppDelegate.m -framework Cocoa -o
cocoa_test2_bundle.app/Contents/MacOS/cocoa_test2
theide code editor will load the .m files after I set the
filter to All Files.

Easy or not, I still see as a good next step producing source code for that app, then adjusting theide to build it...

Mirek

Subject: Re: best way to draw text/fonts on MacOS from C/C++
Posted by daveremba on Wed, 27 Jul 2011 06:12:28 GMT

Here is the minimal app from the Xcode template
for a Cocoa MacOS application:

```
// this code creates a top level window, and draws whatever is in the nib file
// if there is no nib file, the app will draw nothing and no top window appears
// Xcode template puts this code into three files, but they can be merged into one .m file:

#import <Cocoa/Cocoa.h>
int main(int argc, char *argv[])
{
    return NSApplicationMain(argc,  (const char **) argv);
}

@interface cocoa_test2AppDelegate : NSObject <NSApplicationDelegate> {
    NSWindow *window;
}
@property (assign) IBOutlet NSWindow *window;
```

```
@end

@implementation cocoa_test2AppDelegate
@synthesize window;
// The @synthesize directive automatically generates the setters and getters
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
 // Insert code here to initialize your application
}
@end
```

and this file must exist:
  cocoa_test2_bundle.app/Contents/Resources/English.lproj/MainMenu.nib

---

Subject: Re: best way to draw text/fonts on MacOS from C/C++
Posted by mirek on Wed, 27 Jul 2011 08:15:38 GMT
View Forum Message <> Reply to Message

daveremba wrote on Wed, 27 July 2011 02:12Here is the minimal app from the Xcode template for a Cocoa MacOS application:

```
// this code creates a top level window, and draws whatever is in the nib file
// if there is no nib file, the app will draw nothing and no top window appears
// Xcode template puts this code into three files, but they can be merged into one .m file:

#import <Cocoa/Cocoa.h>
int main(int argc, char *argv[])
{
    return NSApplicationMain(argc,  (const char **) argv);
}

@interface cocoa_test2AppDelegate : NSObject <NSApplicationDelegate> {
    NSWindow *window;
}
@property (assign) IBOutlet NSWindow *window;
@end

@implementation cocoa_test2AppDelegate
@synthesize window;
// The @synthesize directive automatically generates the setters and getters
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
 // Insert code here to initialize your application
}
@end
```

and this file must exist:
cocoa_test2_bundle.app/Contents/Resources/English.lproj/MainMenu.nib

Could we do without nib eventually?

Subject: Re: best way to draw text/fonts on MacOS from C/C++
Posted by daveremba on Wed, 27 Jul 2011 19:31:27 GMT
View Forum Message <> Reply to Message

Quote:Could we do without nib eventually?
Yes, but maybe 2000 lines of code is needed:
(comparing to FLTK)

If there is no nib file then it seems one
must create their own top level window and
process event handling (using NS... obj-C API),
and draw text and/or geometry (using CG... and CT...
C++ APIs) at minimum.

Rather than end up writing a stand alone app
(even using theide), I think it makes sense to either
a) fit new MacOSX code into the Rainbow framework,
or b) connect the UPP low level drawing code
using CtrlCore mods.

probably option a is the recommended path.

Do the test and example applications work
for Rainbow with UPP on Windows?

I'd have to guess at what UPP should do with
some of these window & app events, maybe using
other UI kits as a guide.

MacOSX question (for me): what is difference
between a "key" window, "main" window,
and an "active" window?
what's the correct way to send the events into UPP?

from fl_cocoa.mm
window & application events:

- (void)windowDidMove:(NSNotification *)notif;
- (void)windowDidResize:(NSNotification *)notif;
- (void)windowDidResignKey:(NSNotification *)notif;
- (void)windowDidBecomeKey:(NSNotification *)notif;
- (void)windowDidBecomeMain:(NSNotification *)notif;
- (void)windowDidDeminiaturize:(NSNotification *)notif;
- (void)windowDidMiniaturize:(NSNotification *)notif;
- (void)windowWillClose:(NSNotification *)notif;
- (void)anywindowwillclosenotif:(NSNotification *)notif;
- (NSApplicationTerminateReply)applicationShouldTerminate:(NSApplication*)sender;
- (void)applicationDidBecomeActive:(NSNotification *)notify;
- (void)applicationWillResignActive:(NSNotification *)notify;
- (void)applicationWillHide:(NSNotification *)notify;
- (void)applicationWillUnhide:(NSNotification *)notify;
- (id)windowWillReturnFieldEditor:(NSWindow *)sender toObject:(id)client;

device events:
+ (void)prepareEtext:(NSString*)aString;
- (id)init;
- (void)drawRect:(NSRect)rect;
- (BOOL)acceptsFirstResponder;
- (BOOL)acceptsFirstMouse:(NSEvent*)theEvent;
- (BOOL)performKeyEquivalent:(NSEvent*)theEvent;
- (void)mouseUp:(NSEvent *)theEvent;
- (void)rightMouseUp:(NSEvent *)theEvent;
- (void)otherMouseUp:(NSEvent *)theEvent;
- (void)mouseDown:(NSEvent *)theEvent;
- (void)rightMouseDown:(NSEvent *)theEvent;
- (void)otherMouseDown:(NSEvent *)theEvent;
- (void)mouseMoved:(NSEvent *)theEvent;
- (void)mouseDragged:(NSEvent *)theEvent;
- (void)rightMouseDragged:(NSEvent *)theEvent;
- (void)otherMouseDragged:(NSEvent *)theEvent;
- (void)scrollWheel:(NSEvent *)theEvent;
- (BOOL)handleKeyDown:(NSEvent *)theEvent;
- (void)keyDown:(NSEvent *)theEvent;
- (void)keyUp:(NSEvent *)theEvent;
- (void)flagsChanged:(NSEvent *)theEvent;
- (NSDragOperation)draggingEntered:(id < NSDraggingInfo >)sender;
- (NSDragOperation)draggingUpdated:(id < NSDraggingInfo >)sender;
- (BOOL)performDragOperation:(id <NSDraggingInfo>)sender;
- (void)draggingExited:(id < NSDraggingInfo >)sender;
- (NSDragOperation)draggingSourceOperationMaskForLocal:(BOOL)isLocal;

Dave

---

daveremba wrote on Wed, 27 July 2011 15:31Quote:Could we do without nib eventually?
Yes, but maybe 2000 lines of code is needed:
(comparing to FLTK)

If there is no nib file then it seems one
must create their own top level window and
process event handling (using NS... obj-C API),
and draw text and/or geometry (using CG... and CT...
C++ APIs) at minimum.

Well, we will have to do those anyway...

Quote:
Rather than end up writing a stand alone app
(even using theide), I think it makes sense to either
a) fit new MacOSX code into the Rainbow framework,

Sure, obviously. But I guess you might now be a little bit cofused about Rainbow by "Framebuffer backend".

Rainbow is a compile time method how to replace GUI backend in U++ without tampering with CtrlCore. Framebuffer is an example of such relacement backend.

MacOS X will be another such replacement backend (and perhaps, after fully developed, we will move it to CtrlCore anyway).

All in all, right now, creating "nib-less" cocoa "Hello world" and then making it to compile in theide still makes the most sense as the very next step...

Quote:
what's the correct way to send the events into UPP?

from fl_cocoa.mm
window & application events:
- (void)windowDidMove:(NSNotification *)notif;
- (void)windowDidResize:(NSNotification *)notif;
- (void)windowDidResignKey:(NSNotification *)notif;
- (void)windowDidBecomeKey:(NSNotification *)notif;
- (void)windowDidBecomeMain:(NSNotification *)notif;
- (void)windowDidDeminiaturize:(NSNotification *)notif;
- (void)windowDidMiniaturize:(NSNotification *)notif;
- (void)windowWillClose:(NSNotification *)notif;
- (void)anywindowwillclosenotif:(NSNotification *)notif;

- (NSApplicationTerminateReply)applicationShouldTerminate:(NSApplication*)sender;
- (void)applicationDidBecomeActive:(NSNotification *)notify;
- (void)applicationWillResignActive:(NSNotification *)notify;
- (void)applicationWillHide:(NSNotification *)notify;
- (void)applicationWillUnhide:(NSNotification *)notify;
- (id)windowWillReturnFieldEditor:(NSWindow *)sender toObject:(id)client;

device events:
+ (void)prepareEtext:(NSString*)aString;
- (id)init;
- (void)drawRect:(NSRect)rect;
- (BOOL)acceptsFirstResponder;
- (BOOL)acceptsFirstMouse:(NSEvent*)theEvent;
- (BOOL)performKeyEquivalent:(NSEvent*)theEvent;
- (void)mouseUp:(NSEvent *)theEvent;
- (void)rightMouseUp:(NSEvent *)theEvent;
- (void)otherMouseUp:(NSEvent *)theEvent;
- (void)mouseDown:(NSEvent *)theEvent;
- (void)rightMouseDown:(NSEvent *)theEvent;
- (void)otherMouseDown:(NSEvent *)theEvent;
- (void)mouseMoved:(NSEvent *)theEvent;
- (void)mouseDragged:(NSEvent *)theEvent;
- (void)rightMouseDragged:(NSEvent *)theEvent;
- (void)otherMouseDragged:(NSEvent *)theEvent;
- (void)scrollWheel:(NSEvent *)theEvent;
- (BOOL)handleKeyDown:(NSEvent *)theEvent;
- (void)keyDown:(NSEvent *)theEvent;
- (void)keyUp:(NSEvent *)theEvent;
- (void)flagsChanged:(NSEvent *)theEvent;
- (NSDragOperation)draggingEntered:(id < NSDraggingInfo >)sender;
- (NSDragOperation)draggingUpdated:(id < NSDraggingInfo >)sender;
- (BOOL)performDragOperation:(id <NSDraggingInfo>)sender;
- (void)draggingExited:(id < NSDraggingInfo >)sender;
- (NSDragOperation)draggingSourceOperationMaskForLocal:(BOOL)isLocal;

Sounds pretty much similar to Win32/X11. Coresponding files in CtrlCore are Win32Proc.cpp and X11Proc.cpp...

Subject: Re: best way to draw text/fonts on MacOS from C/C++
Posted by fudadmin on Mon, 05 Sep 2011 12:35:53 GMT
View Forum Message <> Reply to Message

mirek wrote on Wed, 27 July 2011 09:15daveremba wrote on Wed, 27 July 2011 02:12Here is the minimal app from the Xcode template
for a Cocoa MacOS application:

```
// this code creates a top level window, and draws whatever is in the nib file
// if there is no nib file, the app will draw nothing and no top window appears
// Xcode template puts this code into three files, but they can be merged into one .m file:

#import <Cocoa/Cocoa.h>
int main(int argc, char *argv[])
{
    return NSApplicationMain(argc,  (const char **) argv);
}

@interface cocoa_test2AppDelegate : NSObject <NSApplicationDelegate> {
    NSWindow *window;
}
@property (assign) IBOutlet NSWindow *window;
@end

@implementation cocoa_test2AppDelegate
@synthesize window;
// The @synthesize directive automatically generates the setters and getters
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
 // Insert code here to initialize your application
}
@end
```

and this file must exist:
cocoa_test2_bundle.app/Contents/Resources/English.lproj/MainMenu.nib

Could we do without nib eventually?

Yes. As I said in this post http://www.ultimatepp.org/forum/index.php?t=msg&goto=30421&&srch=nib#msg_30421
Quote: upp mac apps without nib files will be better than Qt's

Have you had a look inside http://code.google.com/p/upp-mac/
?Quote:Yes, but maybe 2000 lines of code is needed:
(comparing to FLTK)
1 line actually made the difference:

```
 [application setDelegate:appDelegate]; //create menus in appDelegate
//[application setDelegate:nil]; //to test or if can find another way to apple mac style menus
//!!! return NSApplicationMain(argc,  (const char **) argv); //don't use this if you want nibless
```

P.S I am back from my kind of holiday again.