QUARTZ CLOCK TUTORIAL

DRAWING A CLOCK IN QUARTZ

This tutorial describes a clock application. This application is fun. Of course it's more complex that the "Clock" Tutorial. However the result is pure eye candy which I hope you'll enjoy. I draw the clock face complete with shadows. The dock icon is "alive".

We're going to define a Custom Control called QuartzClock which will draw a Clock of course using the Core Graphics Library Quartz. We're going to have a couple of instances of the Quartz clock. One that's visible in a window, and another one in the Dock. The two could be identical, however I've chosen to make them different because it's more fun. I've also introduced the use of an Obj/C 'category' which we'll discuss later.

I found some code on the internet which saved me some work. I recognize and respect the contribution made by their authors:

Gradient fill category:	http://www.cocoadev.com/index.pl?GradientFill
Basic watch face:	http://iphone-dev-tips.alterplay.com/2010/03/analog-clock-using-quartz-core.html

You can download the code from: **http://clanmills.com/files/QuartzClock.zip**

All errors are mine and I hope you'll let me know if you are not able to follow the recipe. And of course if you don't understand how/why this all works, I'll be happy to explain a little more. You are also welcome to VNC to my computer for a live demo.

Subjects dealt with in this tutorial are:

1) The subjects covered in the Clock Tutorial. Please look at the Clock Tutorial. If you don't understand Clock, you'll be lost here.

2) Creating and using a Custom View.

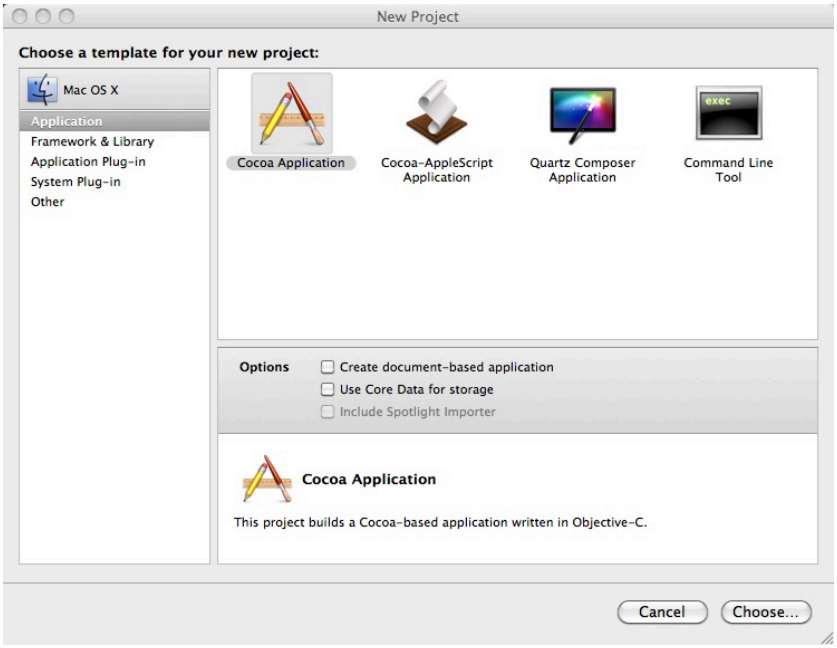
3) Quartz Graphics Model.

4) Drawing in the dock.

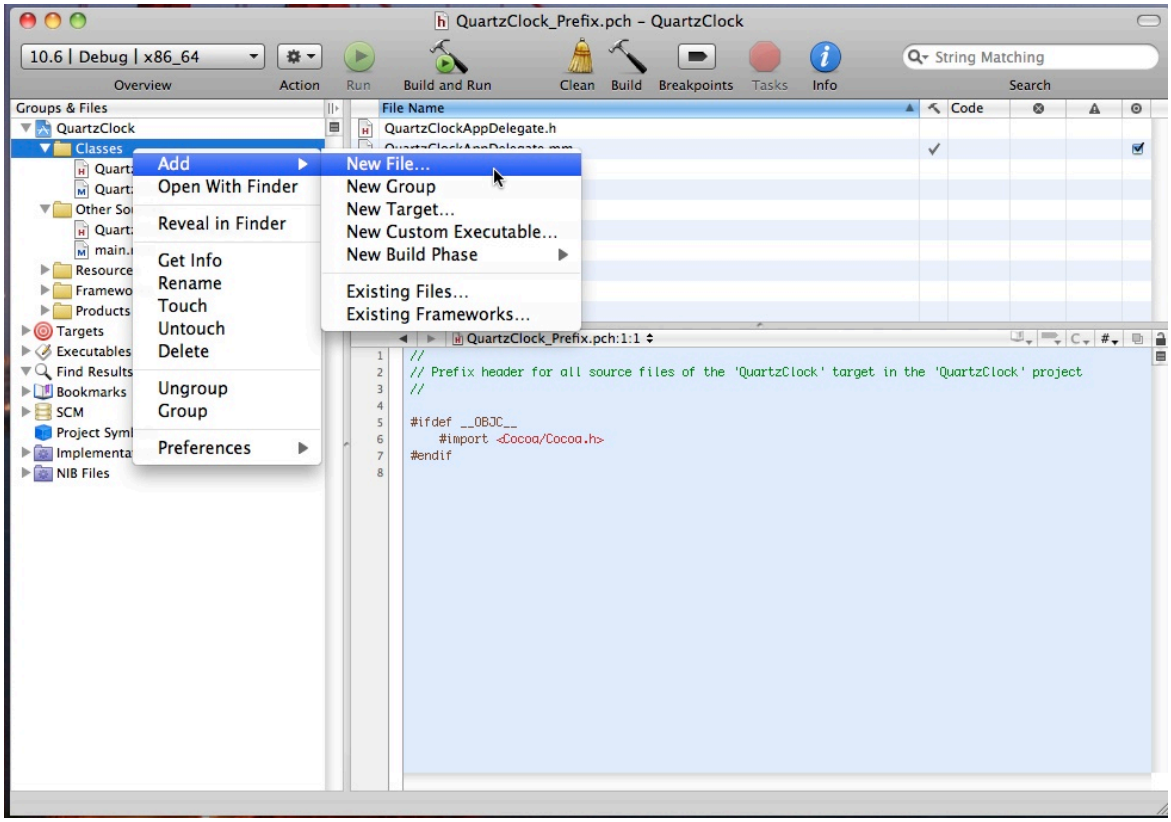
I recommend that you download, build and run QuartzClock.zip. QuartzClock is pure eye-candy. You'll be motivated to learn how this works. Incidentally, I found the code above and had the application implemented in less than 2 hours. Cocoa is your friend.

THE RECIPE

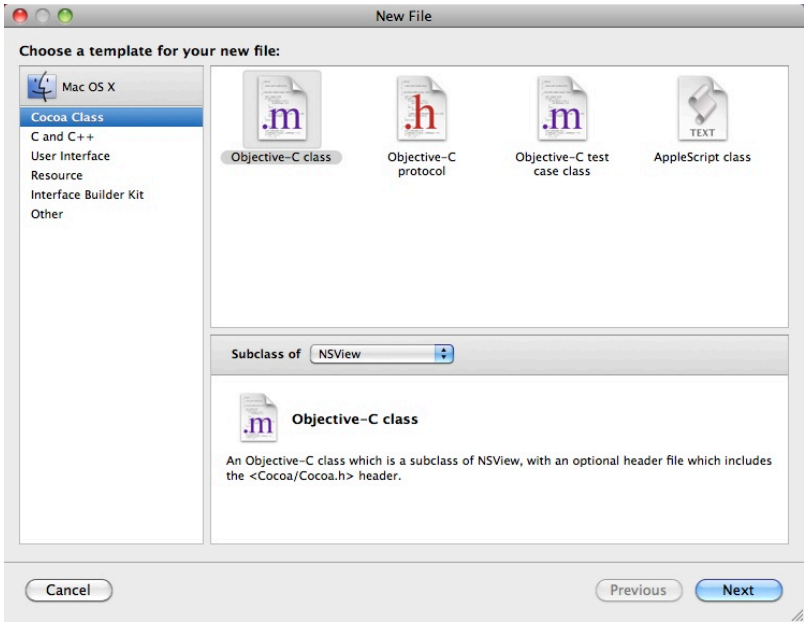
1) Start with an empty application. No need for a document:



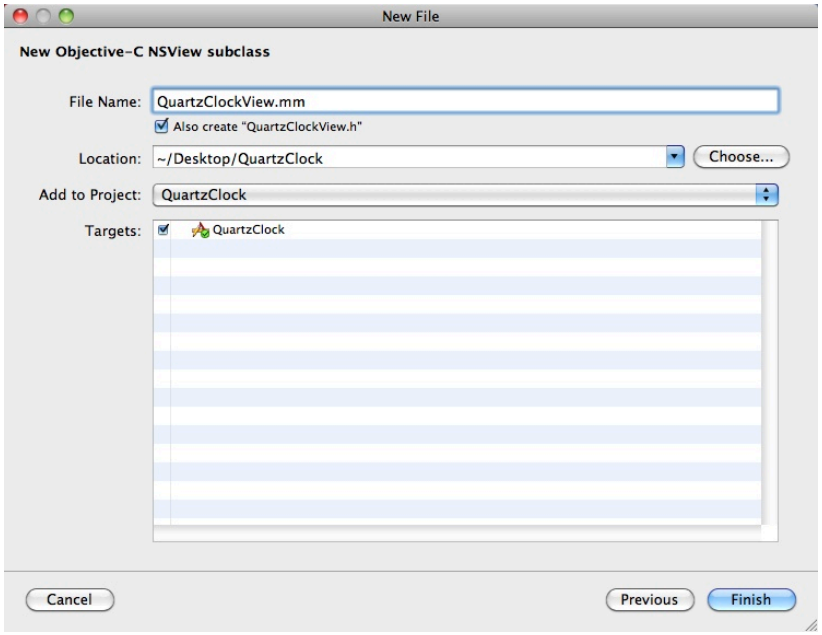
2) Select "Classes", right-click, Add > New File ...



3) Select a "Cocoa Class" .m Objective-C class Subclass of NSView



4) Call it QuartzClockView.mm and check to also create QuartzClockView.h



5) Convert your existing files to C++ (by renaming the .m files as .mm)

Select QuartzClock/Classes/QuartzClockAppDelegate.m, right-click and rename as QuartzClockAppDelegate.mm
Select Clock/Other Sources/main.m, right-click and rename as main.mm

6) Copy the code from clanmills.com/files/QuartzClock.zip

The files to copy over are 5 source files and one icon:

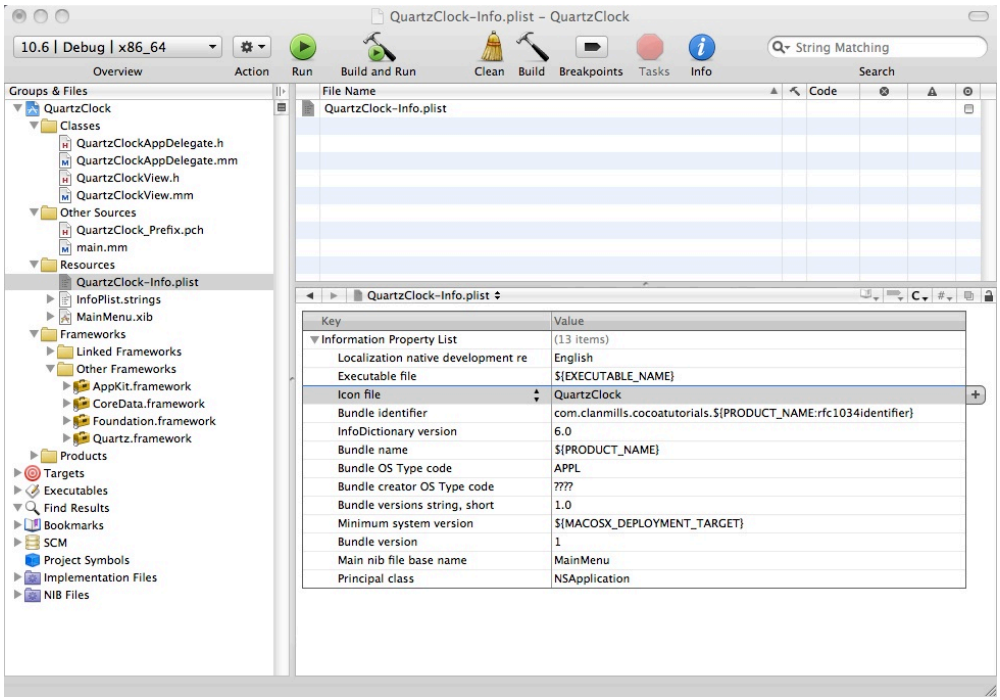
QuartzClockView.mm and QuartzClockView.h	This the main body of code.
QuartzAppDelegate.mm and QuartzAppDelegate.h	The changes here from the wizard's code are small.
main.mm	There are no code changes from the wizard's code.
QuartzClock.icns	I created this icon on your behalf.

7) Add QuartzClock.icns to your resources

Select QuartzClock/Resources/ right-click Add/Existing Files/ .... QuartzClock.icns

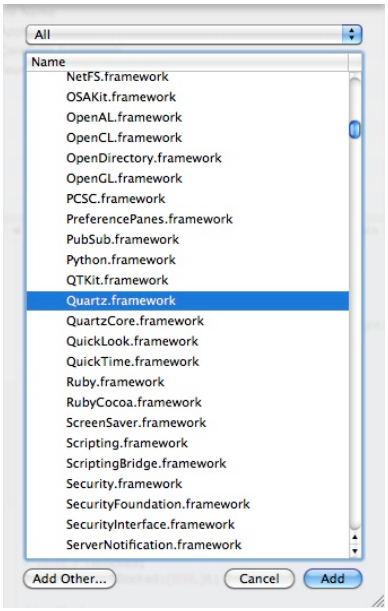
8) Select QuartzClock-Info.plist and edit it.

Set Icon file to QuartzClock (**not** QuartzClock.icns)



8) Add the Quartz to your (linked) Frameworks

Select QuartzClock/Frameworks/Other Frameworks/ Right-click Add/Existing Frameworks ...



9) Build and Run (cmd-B, cmd-R)

You get a big blank window. The icon should be in the dock. This is too boring to be discussed.

INTERFACE BUILDER

1) Double click "MainMenu.xib" in XCode

This will launch Interface Builder of course.

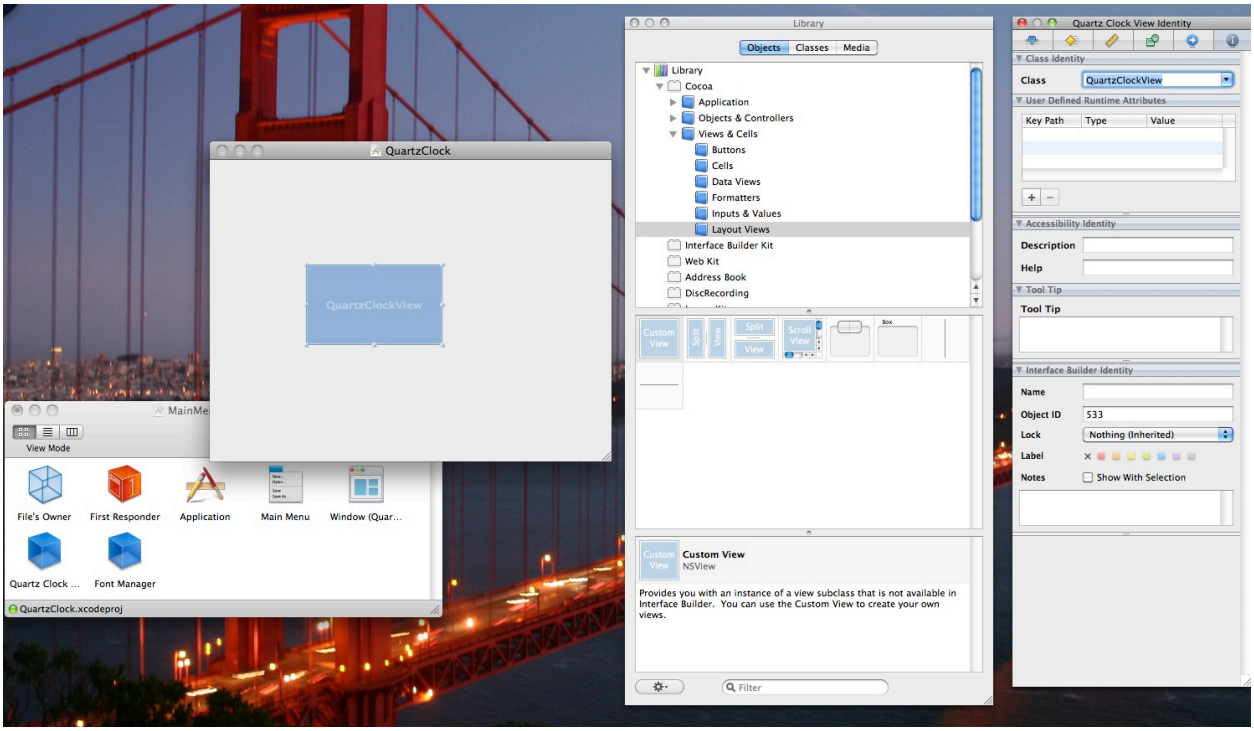
Resize the "QuartzClock" dialog box. Bring up the Inspector (cmd-shift-I) and the Library (cmd-shift-L).

2) Drop your header files from XCode into Interface Builder's "MainMenu.xib" window

Drop QuartzClockView.h. There is no visual feedback - Interface Builder is speechless (I think he's sulking and upset).

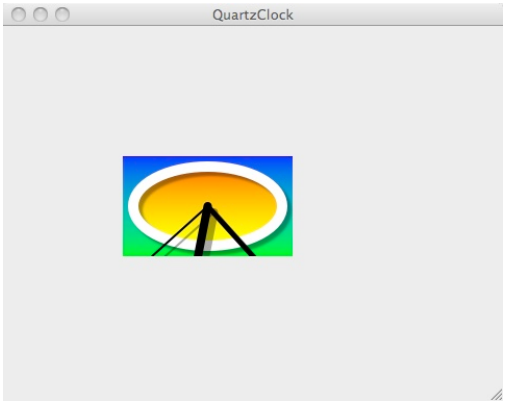
3) Drag a CustomView from the Library and drop him on your QuartzClock window.

Use the inspector to set the class of the CustomView to be "QuartzClockView". Interface Builder will respond by changing the label on the control in the QuartzClock dialog box. Later on we'll want to size this control correct and we'll want to define how he is to be resized. However for the moment, don't worry about that.



5) Save in Interface Builder. (cmd-S) Build and run in X-Code (cmd-B, cmd-R)

At this point you should have a clock. And if you look in the dock, you'll see the icon is "alive" - it's a little blue clock.



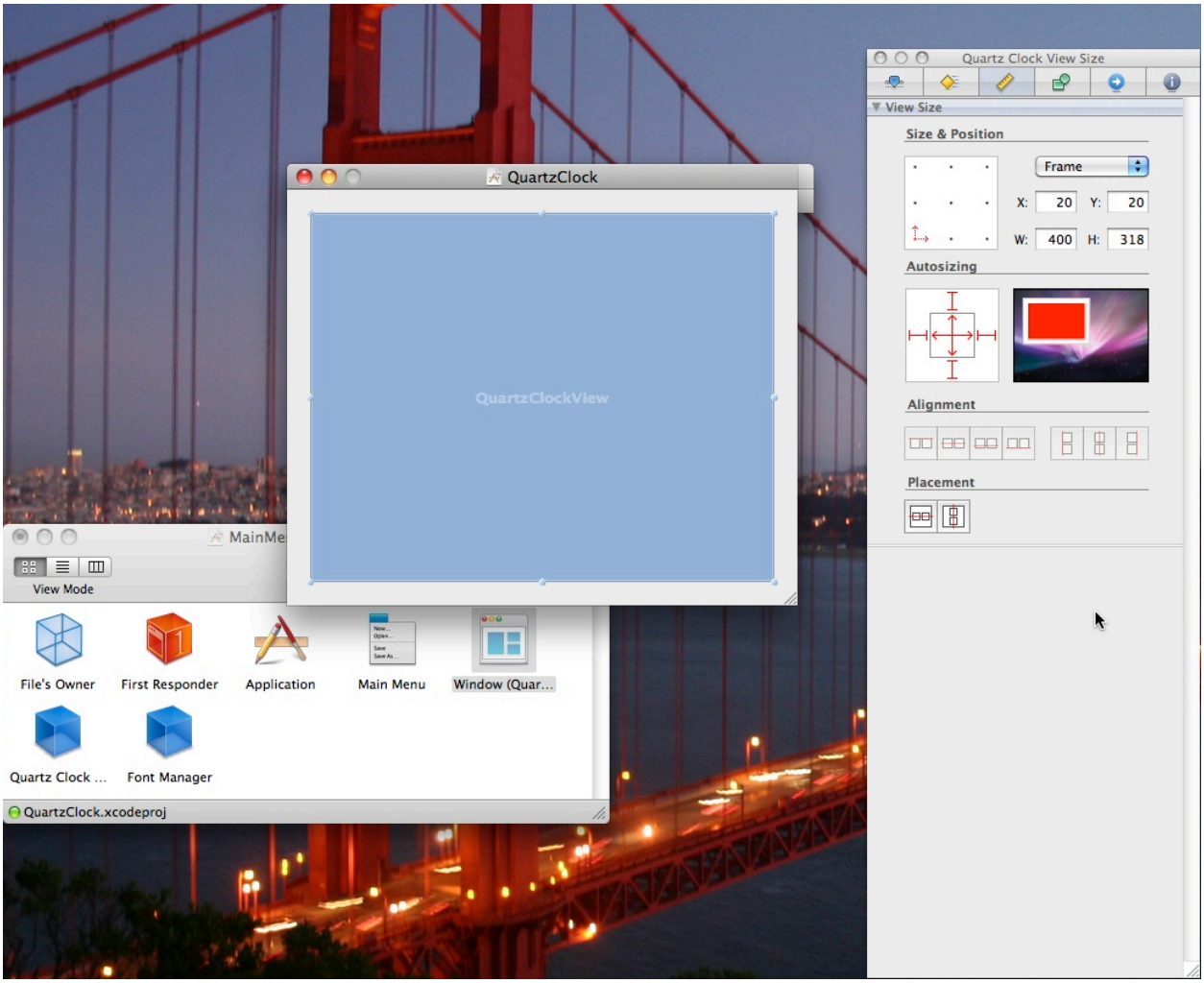
MORE POLISHING IN INTERFACE BUILDER

1) Resize the QuartzClockView to cover all of the our dialog box.

2) Use the QuartzClockView Size in the Inspector

Change the Autosizing controls (the little springs and arrows) to cause our clock to grow and shrink.

Mess around with the size of the Window and the size of the QuartzClock to ensure they start "square" at 300x300.



3) Save in Interface Builder. (cmd-S) Build and run in X-Code (cmd-B, cmd-R)

I hope everything's working. It's now time to discuss the Imaging Model and then to discuss the code.

THE QUARTZ IMAGING MODEL

Before discussing the Obj/C code, I'm going to discuss the imaging model used in MacOS X.

The imaging model being used dates back to PostScript which entered the world in 1985. The model is simple and elegant. When the Win32 API appeared around 1992, the GDI imaging model was a variant of the PostScript model.

Today, the model is used in the PDF standard which is managed and published by Adobe Systems Incorporated. Support for PDF is built into MacOS X and used extensively by the system. In addition to the Preview Application, printing in MacOS X is implemented by creating PDFs which are rendered by CUPS - the Common Unix Printing System.

The Imaging Model has the following features:

1) The PostScript model uses the painter's algorithm.
   The Quartz model has been extended to include transparency and 'compositing'. Compositing is the construction of graphics in layers which can be combined to create final images. Compositing enables transparency and drop shadows.

2) The model has a graphical state which includes attributes such as current color, line width, stroke and fill color. There is a graphical state stack. You can save the state, modify the state and restore to the stacked settings.

3) The current state includes the CTM - the current transformation matrix which defines the scale and rotation of graphics.

4) Device independent rendering. Objects are drawn into a context. The context can be the screen, a printer, a PDF file or an Image. Although your code can be sensitive to the context, in general it is not. So in this case the code to paint the Dock Icon is the same code used to paint on the display. I chose in this case to make the dock item different to show how that can be done.

5) Many 'advanced' graphics features are supported including:
   Text and fonts.
   Filling with patterns and gradients.
   Color adjustment using ICC profiles.
   Color space conversion.
   Graphics filters such as blur and sharpening.
   Animation.
   Shadows and Anti-aliasing.
   3-D support using OpenGL.
   Video.

6) Support for high performance rendering using graphics cards.
   Printing to vector and raster devices.
   Digital Half-toning.

7) Support is provided for standard image file formats such as JPG, PNG, GIF, TIFF and many others.

And I suspect that I haven't mentioned all the great things that are provided. It is sufficient to say that Quartz is a rich and full featured graphics library.

Here's some marketing information:	http://developer.apple.com/technologies/mac/graphics-and-animation.html
PostScript Specification	http://www.adobe.com/products/postscript/pdfs/PLRM.pdf
PostScript Cookbook	http://www-cdf.fnal.gov/offline/PostScript/BLUEBOOK.PDF
PostScript Language Program Design	http://www-cdf.fnal.gov/offline/PostScript/GREENBK.PDF
The PDF specification: pdf_reference.html	http://www.adobe.com/devnet/pdf/
More about PostScript:	http://en.wikipedia.org/wiki/PostScript

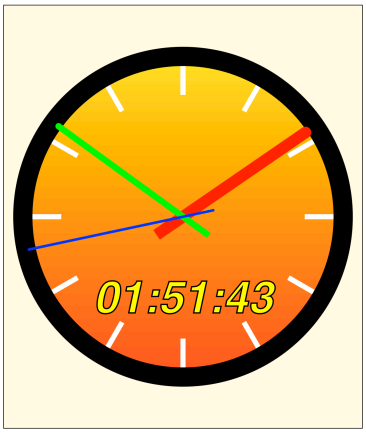
I've written some PostScript to draw a clock with a gradient filled background to demonstrate the kind of thing that can be done.

You'll notice the hands are "rounded" at one end and "butted" at the other. The wider hands are also shorter.

The time is display in both analog and digital format.

The digital time is painted in yellow with a black outline.

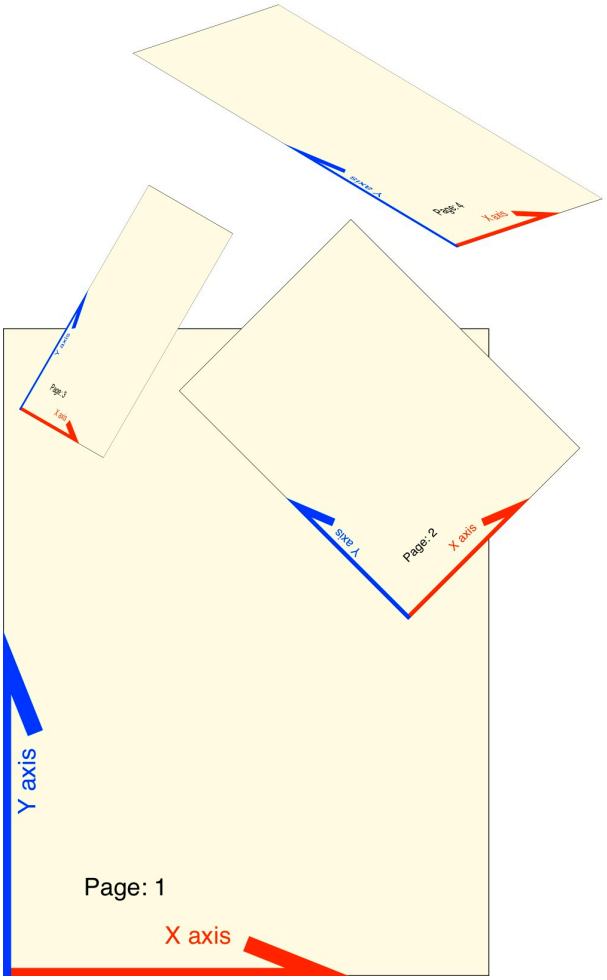
The PostScript code for this is more involved than I want to discuss in this tutorial. The code is in the file clock2.ps if you are interested.



Here is some PostScript and its output. You can run the PostScript interpreter using the command pstopdf from the Terminal.

The PostScript language uses postfix notation (reverse polish) so the operator is placed after the data. The PostScript: 4 setlinewidth would eventually become the "C" code CGSetLineWidth(context,4.0);

```
%!PS
%%File: pages.ps
/red   { 1 0 0 setrgbcolor   } def
/blue  { 0 0 1 setrgbcolor   } def
/black { 0 0 0 setrgbcolor   } def
/linen { 1 1 0.9 setrgbcolor } def

/page {
  /p exch 3 string cvs def
  /Helvetica 15 selectfont
  0 0 300 400 rectclip

  linen 0 0 300 400 rectfill
  black 0 0 300 400 rectstroke

  gsave
     red 10 setlinewidth
    -20 0 moveto 200 0 lineto -50 20 rlineto stroke
    100 20 moveto (X axis) show
  grestore

  gsave
    blue 90 rotate
    10 setlinewidth
   -20 0 moveto 200 0 lineto -50 -20 rlineto stroke
   100 -20 moveto (Y axis) show
  grestore

  50 50 moveto (Page: ) show p show
} def

1.3 dup scale
gsave                                          1 page grestore
gsave 250 220 translate 45 rotate .5 .5 scale  2 page grestore
gsave  10 350 translate -30 rotate .2 .4 scale 3 page grestore
gsave [ 0.3 .1 -.5 .3 280 450 ] concat         4 page grestore

showpage
%%EOF
```



The purpose of this code is to paint a page on the paper. The page is very simple. I draw the linen colored background and then draw the X axis in red (and label it) and the Y axis in blue (and label it). The code to paint the page is in the procedure page.

The main program executes the page painting code within the context of gsave .... grestore. So no matter what happens during page painting, the graphical state is the same at the beginning of every page.

You can see that the default coordinate system is Right-Handed with 0,0 in the lower left. A positive rotate is anti-clockwise (right-hand rule). I've drawn the page in 4 different locations and scales.

Page: 1 is normal.
Page: 2 rotated 45 degrees and scaled equally in X and Y by 50%
Page: 3 rotated -30 degrees and scaled unequally
Page: 4 I've used a matrix to apply the transform
The matrix [....] is a general 3x2 2-dimensional transformation matrix.

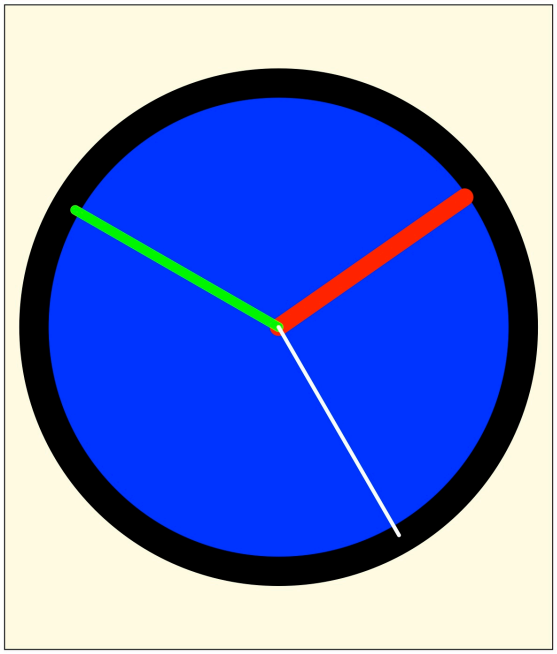
You'll appreciate that the PostScript language cannot be explained in a couple of pages. However I hope you can appreciate the concepts of gsave and grestore. Additionally, you can see operators such as rectclip setrgbcolor setlinewidth selectfont rectfill rectstroke moveto lineto rlineto setlinewidth stroke and show being used. The are about 400 operators in PostScript - so this is a simple example.

The Quartz imaging library provides everything offered by PostScript and much more.

DRAWING THE CLOCK

I've written the clock code in PostScript as I think it's easier to understand than the Obj/C version.

```
%!PS
%%File: clock.ps
/xc     280 def  % center of clock
/yc     330 def
/radius 250 def  % size
/hour     1 def  % time at the moment (1:50:25)
/minute  50 def
/second  25 def

/hour hour minute 60 div add def

/red   { 1 0  0 setrgbcolor } def
/green { 0 1  0 setrgbcolor } def
/blue  { 0 0  1 setrgbcolor } def
/white { 1 1  1 setrgbcolor } def
/black { 0 0  0 setrgbcolor } def
/linen { 1 1 .9 setrgbcolor } def

/mangle { 6 mul        } def % number mangle number
/hangle { 5 mul mangle } def % number hangle number

/hand { % width xc yx angle length hand -
  save
    /length exch def
    /angle  exch def
    moveto
    setlinewidth
    1 setlinecap
    length width sub angle sin mul
    length width sub angle cos mul
    rlineto stroke
  restore
} def

20 20 translate

% draw the page
gsave linen 0 0 xc 2 mul yc 2 mul rectfill   grestore
gsave black 0 0 xc 2 mul yc 2 mul rectstroke grestore

% fill clock background, stroke the perimeter
gsave blue                  xc yc radius 0 360 arc fill   grestore
gsave black 30 setlinewidth xc yc radius 0 360 arc stroke grestore

% draw the hands of the clock
red   18 xc yc hour   hangle radius hand
green 10 xc yc minute mangle radius hand
white  4 xc yc second mangle radius hand

showpage
%%EOF
```



Notice the use of setlinecap. This causes the line cap (terminator) to be circular. No effort for the clock's hards to have round ends. And the hand code make the wider hands shorter. So the hour hand is wider and shorter than the minute and second hands.

The code in the application takes advantage of shadows in Quartz. PostScript does not support shadows or transparency. Shadows are used on the clock perimeter and the clock hands. This gives QuartzClock a pseudo 3-d appearance.

The code in the application uses gradient fills to paint the exterior of the clock and to fill the face of the clock. These are available in PostScript, however I've chosen to only show the solid fill clock face which is displayed in the dock. I've also shown the clock perimeter here in white. I colored the hands because it's more interesting. In the Obj/C code, the perimeter and hands are white.

CODE DISCUSSION - QuartzClockView.mm

1) Drawing the clock's hands

I found the following code on the internet:. http://iphone-dev-tips.alterplay.com/2010/03/analog-clock-using-quartz-core.html

I've modified the original code a little. Mostly to make it run on the desktop. It was originally written for the IPhone and uses classes unique to that platform. The code is clear and simple. It obtains the time from the clock and calculates the angle of the hands of the clock. Then it draws three lines - the hour, minute and second hands of the watch. Good stuff.

2) The gradient fill category

A category is a very interesting concept in the Obj/C language. The code @interface NSBezierPath (Additions) ... @end tells the run-time system that you are going to define a class method. Every object of class NSBezierPath will have this method.

The code @implementation NSBezierPath (Additions) .... @end defines the code.

This construct is called a 'category'. Again Apple have found a unique and confusing name. In JavaScript this is called a prototype function. Mind you that's also a horrible name. I'd call it a class extension interface.

This category adds a GradientFill to the NSBezierPath class. NSBezierPath has methods like fillPath for filling areas with a single color. The new method fillGradientFrom:to:angle provides beautiful gradient fills. Quite awesome. When defined as a category, this code can be called from any part of our program whenever using NSBezierPath. Beautiful and very elegant.

I found the category code on the internet at: http://www.cocoadev.com/index.pl?GradientFill

This code works so well that I haven't bothered to read it. It just works.

3) init: initWithFrame: and awakeFromNib:

The startup code calls our initFromFrame: method in QuartzClockView when the UI is constructed. However I want a second instance of QuartzClockView to be used by the Dock. You might be afraid of an infinite recursion of QuartzClockView however that's not what happens. The startup code needs a QuartzView for the UI, so he creates it and calls initWithFrame:. When we create a QuartzClockView for the Dock, that's it - there are only two clocks.

When the NIB's QuartzClockView has been create, we create the Icon version and set it in the dockTile of the application. We also start the timer by calling startClockUpdates.

4) How does the timer work?

startClockUpdates create a timer which is called every second. The timer 'chimes' it call update: to paint our object and ask the Dock to update his icon.

5) How does the Dock Icon work?

initFromFrame instances a second QuartzClockView and sets his isDocked property to be true. In the painting code in drawRect respects that flag and paints a different style of clock in the Doc.

The QuartzClockView myIcon is established by the awakeFromNib. And of course when we update the view, we also update the iconic version in the Dock.

6) Anything else in QuartzClockView.mm?

Yes. Thanks for asking. There is some rather interesting code - (BOOL) isFlipped. Some devices do not have their initial CTM as a right-handed coordinate system in the bottom-left. When we subclass NSView, we implement this for QuartzClockView to return YES. Oddly, I'm not convinced this is working correctly on my PPC/Tiger laptop. A curious matter to be investigated one day.

CODE DISCUSSION - QuartzClockAppDelegate.mm

The methods here modify the user experience. Some this can be enabled in Interface Builder. However run-time code is also required to achieve these results. I wanted to change the default behavior in the following ways:

1 The window remains square when resized
2 You can close and open the window
3 The window is always 'topmost" and cannot be covered by other windows
4 The application starts without showing it window.
5 I don't want it to minimize into the dock - the application's already running nicely there.

1) windowWillResize:

This is part of the NSWindowDelegate protocol. We are called back by the window when the window wishes to resize. To avoid the problem of drawing an oval clock, I've taken advantage of the delegate to ensure that our control is always square.

2) applicationDidFinishLaunching:

This is very similar to the code in the Clock Tutorial. I unchecked the box "visible at launch" in the Window attributes in Interface Builder. And the first thing I do is toe hide the app. Then set it to the front. See Clock.pdf for additional information.

3) windowShouldClose:

This is called when you click the "Close" button (the red top-left window control). Say NO and Hide the application.

CHALLENGES

This is a fun application and many things could be added to this.

1) When you modify the system clock the clock isn't modified.

2) Use a non rectangular window to house the clock (use a circular window)

3) Enable the user to drag the clock over the screen by dragging on the clock face

4) Enable the user to resize a circular window

5) When the clock is asked to close, simply hide him. When asked to show, unhide him.

6) Add a preference panel to the application to allow the user to store preferences such as:

Current location and size of clock
Color Settings for the clock (color of the face, gradient colors and so on).
Store the preferences in ~/Library/Preferences

7) Add more clock "Themes".

At the moment I have two themes: The gradient fill used in the window
The rather dull them in the dock
Make the theme preview 'alive' in the preference panel.
Make the clock icon in the "About Box" to be 'alive'.
Get the clock to speak (see Tutorial "Listen")
Add a 'alarm clock' feature. Enable the alarm to be set by voice commands (see Tutorial "Listen")

8) Use an image as part of the Theme!

Store "Clock.icns" (from the Clock Tutorial) in the resources
Render "Clock.icns" as part of drawRect: Then draw the clock hands over the Clock image.

9) Define Themes in PostScript

Use the PostScript interpreter in MacOS X to render user defined themes.

10) Get it to work on Tiger, Leopard and Snow Leopard

Create a Universal binary for PPC and Intel, 32 and 64 bit. Test on all platforms.

11) Design and implement a plugin mechanism to allow other people to add themes

That lot will keep you busy for at least an afternoon. When you're finished, you'll have a shareware software product.

LICENSE

Robin Mills
Software Engineer

Windows/Mac/Linux
C++, Web, JavaScript, UI

400 N First St #311
San Jose, CA 95112

T (408) 288 7673
C (408) 394 1534
robin@clanmills.com

http://clanmills.com

Revision History

20100426	Added clock2.ps
20100425	Modified the user experience to start the application without showing the main clock. Starts in dock.
20100424	Add clock.ps. Fixed some typos.
20100423	Initial version.