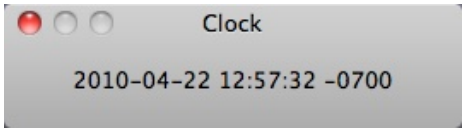
CLOCK TUTORIAL

VERY SIMPLE "HELLO WORLD" COCOA APPLICATION

Life in a new programming environment has to start somewhere. Everybody knows the "hello world" application written in C. This little clock is about the simplest Mac application which I can imagine.



I'd like to acknowledge the contribution of my buddy maddogandnoriko on http://www.cocoaforum.com/ Thank you Todd for bringing the method NSObject performSelector:withObject:afterDelay: to my attention. It's perfect for this application.

I actually have in mind creating a series of clock applications. Quartz Clock will of course use Quartz to render a beautiful clock on the display - complete with dock icon which will of course also be rendered. And probably a WebKit Clock which would use JavaScript, and maybe even PDF Clock. We'll see.

The application is very simple. It's a dialog box with a string. When the application starts, it calls tick: and his job is to update the string and send a message to himself after a delay of 1 second.

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

The subjects which we deal with in this tutorial are:

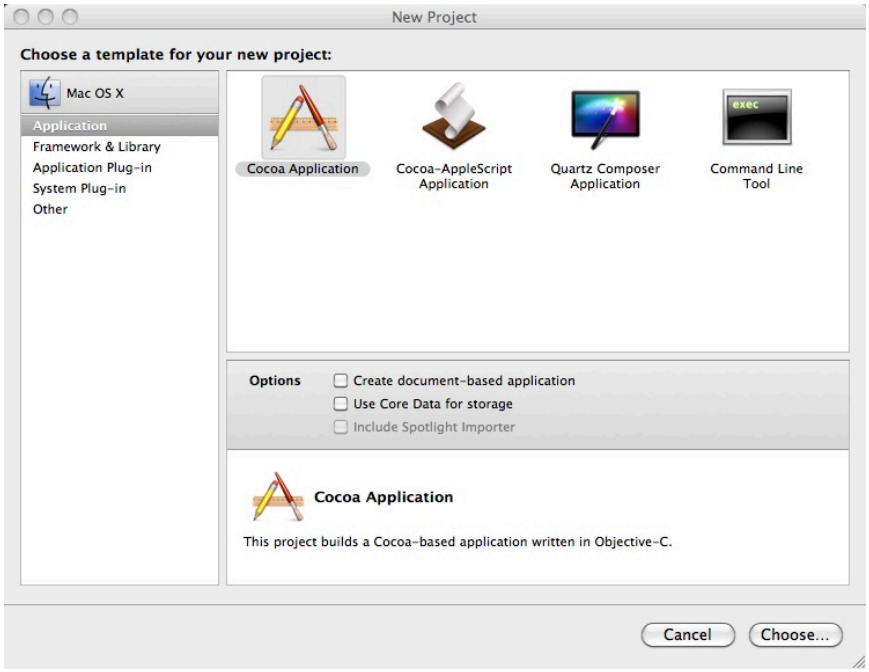
1) Creating a very simple application

2) Using NSAppDelegate and NSWindowDelegate

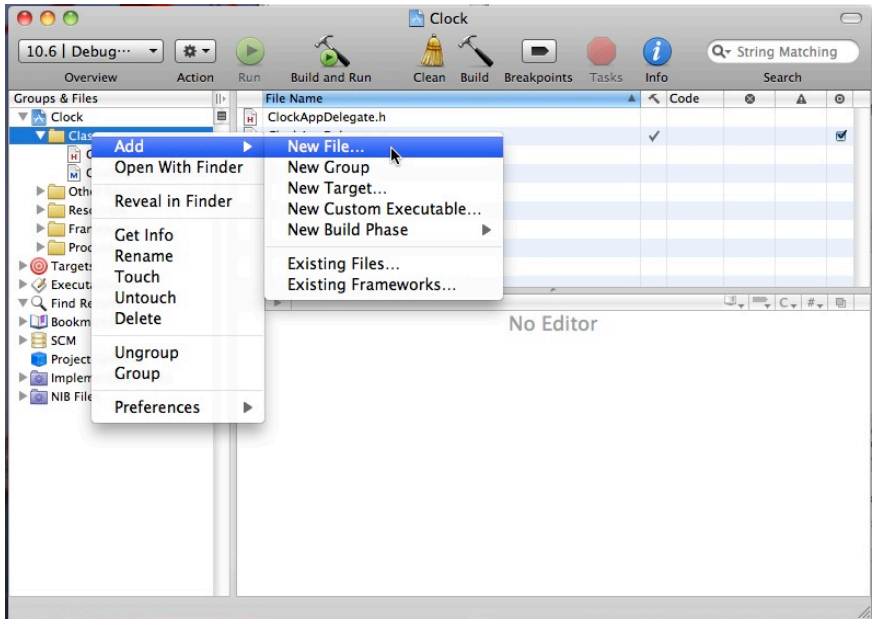
3) A little discussion about messages and selectors.

START IN XCODE

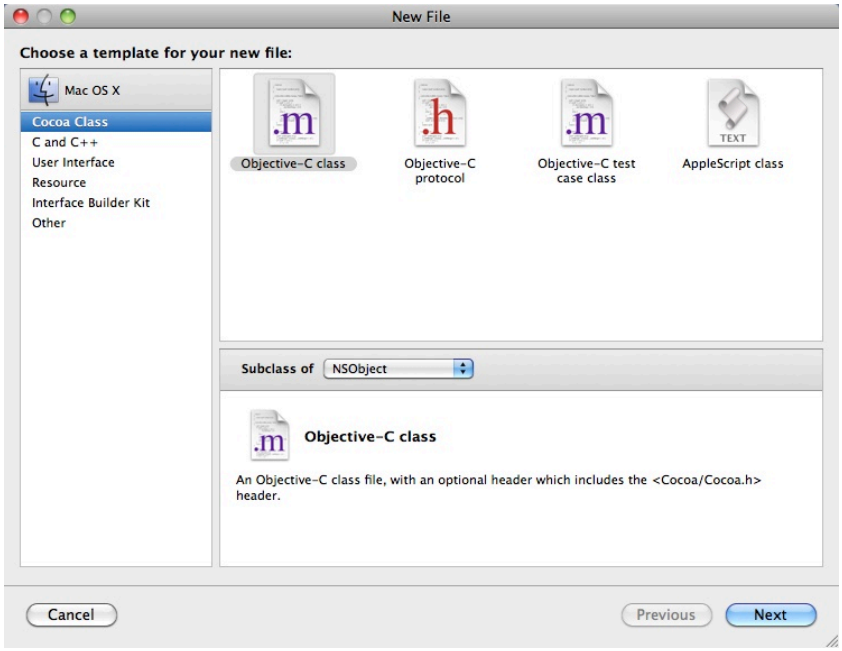
1) Start with an empty application. No need for a document:



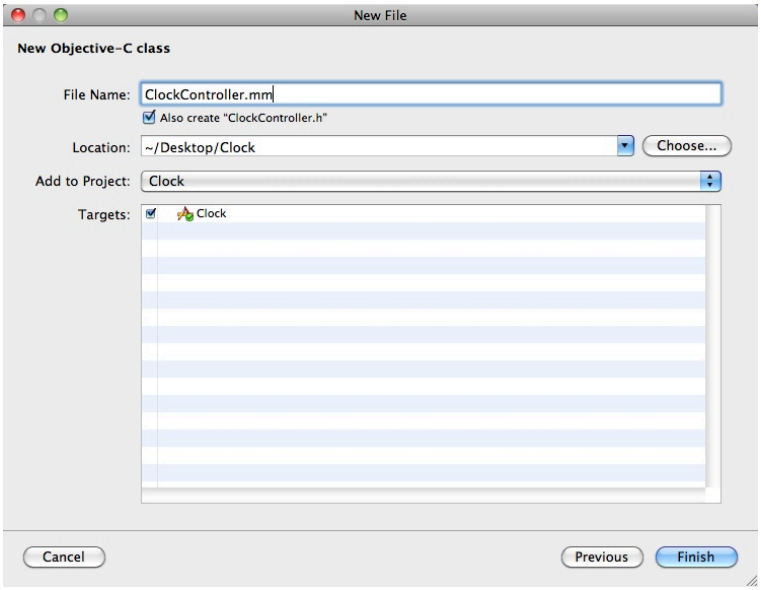
2) Select "Classes", right-click, Add > New File ...



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



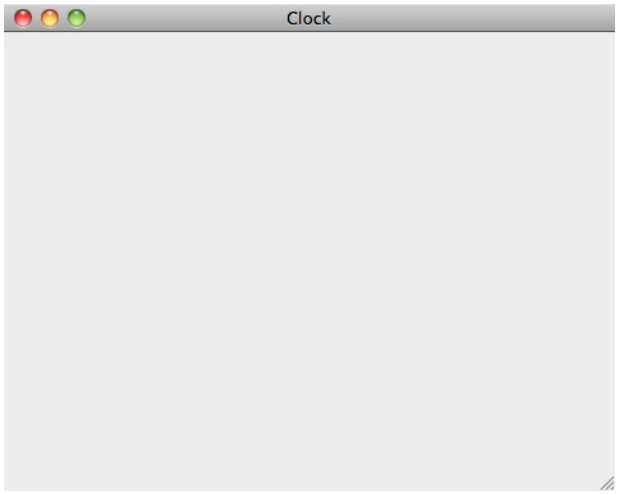
4) Call it ClockController.mm and check to also create ClockController.h



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

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

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



It's incredibly boring. We'll fix that in a couple of minutes.

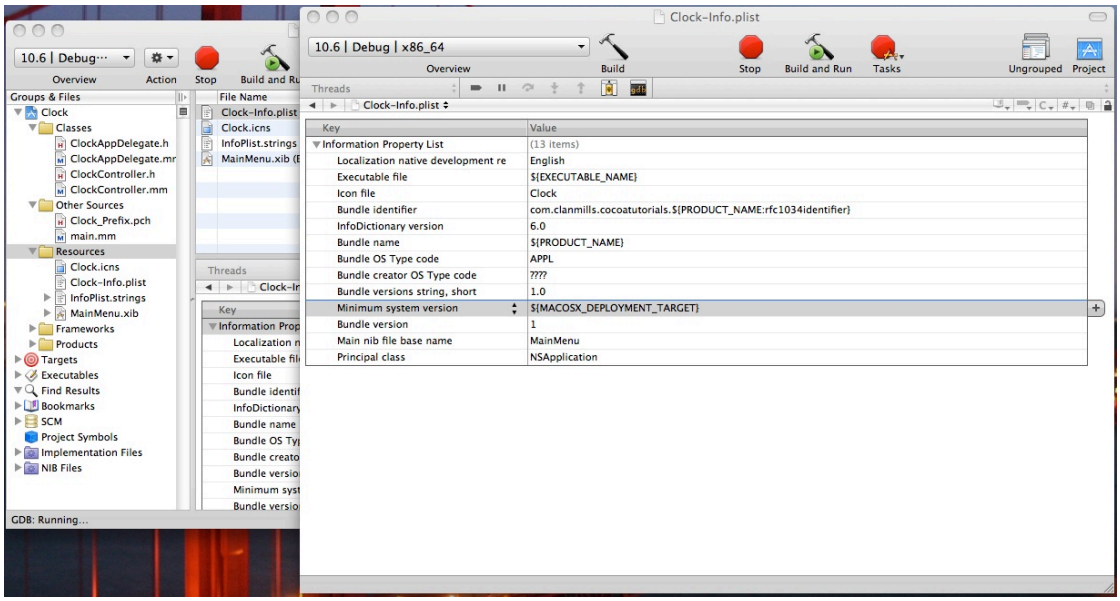
6) Create an icon and add it the Resources in the project

You can of course simply copy Clock.icns from the zip on clanmills.com. Alternatively, google up an image. Use the Icon Composer application (in /Developer/Applications/Utilties). Get your image onto the clip-board and paste it into Icon Composer. Save your new icon as Clock.icns In XCode, Select Resource, right-click Add / Existing Files ... and select Clock.icns.

If you've downloaded a clock image that you like (eg by dragging it off Safari onto the Desktop). Open the image in Preview and copy it cmd-A, cmd-C.

7) Select Clock-Info.plist and edit it.

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

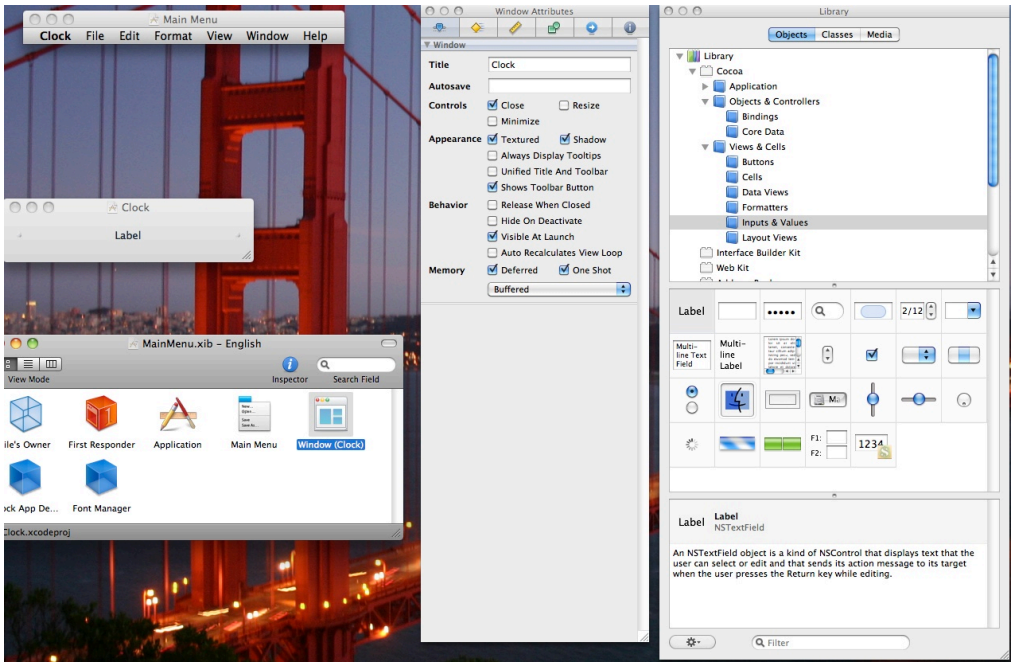


IN INTERFACE BUILDER

2) Double click "MainMenu.xib" in XCode

This will launch Interface Builder of course.

Resize the "Clock" dialog box. Bring up the Inspector (cmd-shift-I) and the Library (cmd-shift-L). Drop a label on the dialog box and set the Window Attributes as shown (No minimize button and textured). Save in Interface Builder.



EDIT OUR CODE IN X-CODE

1) Define our ClockController Interface in ClockController.h

```
@interface ClockController : NSWindowController
{
```

```
      // outlets
      IBOutlet NSTextField* timeString ;

      // properties
}
// actions

// property getters/setters

// methods
- (void) tick:(id)sender;
@end
```

There is a single TextField (which will be paired up with Interface Builder shortly). And a single method tick: (which we'll write).

2) Add code to ClockController.mm

Here's the code for ClockController.mm. The method awakeFromNib: is called by the system when our application starts. So we call tick. The tick method updates the timeString in the UI and asks the system to call him again in 1 second. How hard can it be? Simple isn't it? I'll explain a little more about all of this in a moment.

```
#import "ClockController.h"

@implementation ClockController

- (void) awakeFromNib
{
      [self tick:nil];
}

- (void) tick:(id)sender
{
      [ timeString setStringValue:[[NSDate date]description]];
      [ self performSelector: @selector(tick:)
                  withObject: nil
                  afterDelay: 1.0
      ];
}

@end
```

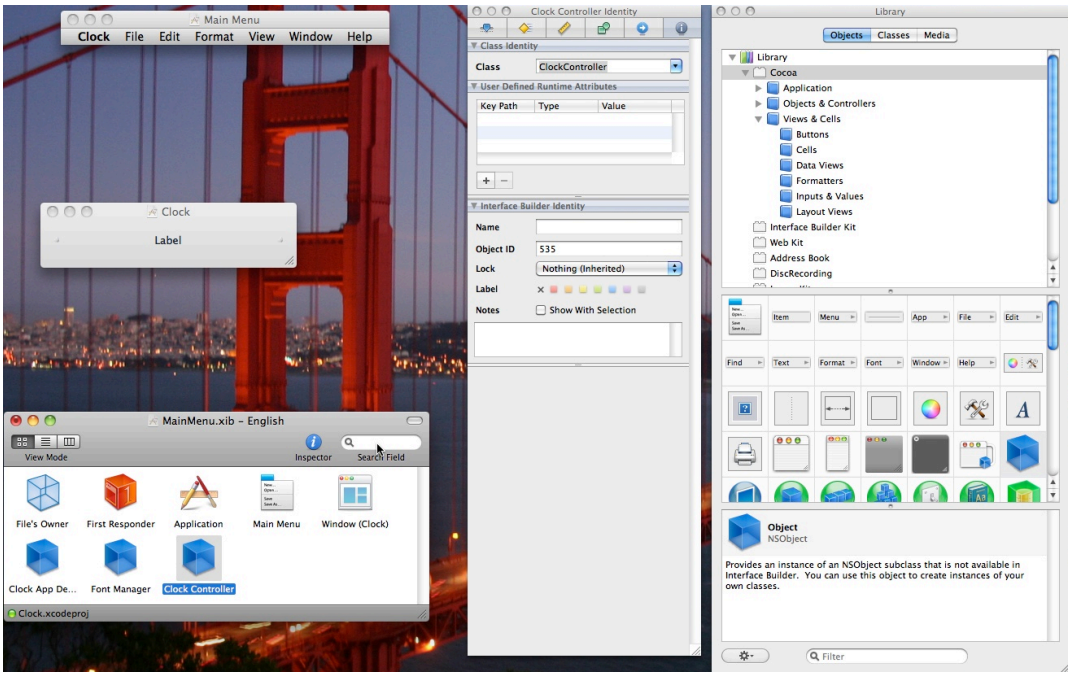
GO BACK TO INTERFACE BUILDER

2) Drag'n'Drop ClockController.h from XCode to MainMenu.xib in Interface Builder

This enables Interface Builder to know about the ClockController interface.

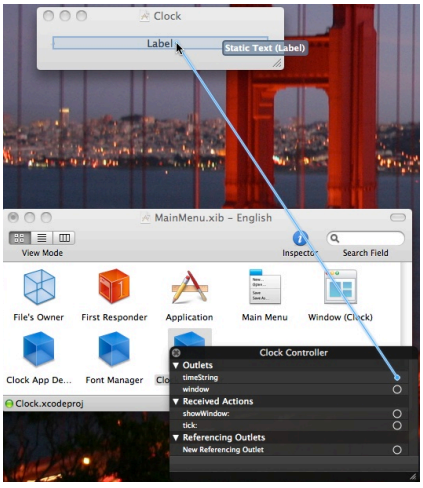
2) Drag'n'Drop an NSObject (a cube) from the Library to MainMenu.xib

Set the cube's Class to ClockController with the Inspector. The 'cube' will change it's name by magic.



3) Right click on the Clock Controller 'cube'

Drag the blue band to connect our timeString in Obj/C to the Label in the User Interface.



3) Save in Interface Builder. Build in XCode and Run

Yes, it is that easy. You've got a clock.

ADDING BELLS AND WHISTLES IN XCODE

The application's working. However there are a couple of things I'd like to change:

1) I want it to sit on top of all other windows.
2) When we close the window, I'd like the application to terminate.

2) Change ClockAppDelegate.h

Modify the ClockAppDelegate to tell him that he implements the NSWindowDelegate protocol. (I'll explain this shortly)

```
#import <Cocoa/Cocoa.h>

@interface ClockAppDelegate : NSObject
#if __ENVIRONMENT_MAC_OS_X_VERSION_MIN_REQUIRED__ >= 1060
                  <NSWindowDelegate,NSApplicationDelegate>
#endif
{
    NSWindow* window;
}

@property (assign) IBOutlet NSWindow *window;

@end
```

2) Change ClockAppDelegate.mm

We're going to modify applicationDidFinishLaunching to make the Windows sit on top (the setLevel code). And we're going to tell the window the treat us as a delegate. The window will then call windowShouldClose when the user presses the window close button. We die gracefully.

I'm going to explain more about Delegates in a moment.

```
#import "ClockAppDelegate.h"

@implementation ClockAppDelegate

@synthesize window;

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
      // Insert code here to initialize your application

      NSLog(@"applicationDidFinishLaunching");
      [window setDelegate:self];
      [window setLevel:NSFloatingWindowLevel];
}

- (BOOL) windowShouldClose:(NSNotification *)notification
{
      NSLog(@"windowShouldClose");
      [NSApp stop:nil];
      return YES;
}

@end
```

3) Build in XCode and Run

Happiness, I hope.

ABOUT THE CODE

It's not easy to get started with Cocoa. And this very small application requires knowledge of some concepts that are not obvious. If you're arriving as a refugee from Windows (MFC, Win32, or VB), you'll probably feel very lost in the land of Cocoa. Knowledge of C++ doesn't help much either with Obj/C. The object model in Obj/C is very different from C++ and reminds me a lot of JavaScript. In C++ the compiler knows everything about your code, and the run-time system incur almost no overhead. Obj/C is much more like JavaScript and the run-time system manages your objects.

Some of the subjects I'm going to discuss below are quite difficult to understand - especially if you are 100% new to Cocoa. If you don't understand them, don't worry. If you've understood the tutorial and have Clock working, you're in good shape. However you might want to return to these pages when you've learned more Cocoa and you'll probably appreciate more of what I have to say here.

Without doubt, Apple's culture is both "No invented here" and "We never do things the way anybody else does things". I don't know if this is good or bad - it's how it is. Everything in Cocoa has a different name from what it's called in other domains.

1) What's self ?

C++ and JavaScript call it 'this'. Its a keyword in Obj/C to enable an object to refer to itself.

2) What's a delegate ?

A 'delegate' is simply a callback. So when we say we're going to ourself to be the window delegate what we're saying is "mr window, please call me about various events". So:

```
[window setDelegate:self];
```

is simply telling the window to call us appropriately. When you consult the documentation for NSWindowDelegate, you'll discover there are 20 to 30 methods. All are optional. In this case we've implemented:

```
- (BOOL) windowShouldClose:(NSNotification *)notification
{
      NSLog(@"windowShouldClose");
      [NSApp stop:nil];
      return YES;
}
```

We return YES (which is 1). We also send a message to the application itself (our main.mm if you like) to ask him to stop.

The function NSLog(....) simply writes in the system log. When debugging in XCode, log messages are written to the console.

3) What's a selector ?

We use a selector in the tick method. It's a cousin of the delegate. When we call performSelect, we want to give it a callback. So when it's time to call, perform select knows what's to be called. So @selector(tick:) is effectively the address of the tick: function.

```
- (void) tick: (id) sender
{
    [ self performSelector: @selector(tick:)
        withObject: nil
        afterDelay: 1.0
    ];
}
```

4) How can a delegate have optional methods?

There is however a little more to be said about selectors. When Obj/C is compiling code, he assigns a unique identifier to every entry point name in the code. So performSelect:withObject:afterDelay: has a number and it's different from tick:. So the selector isn't a memory address, it's an integer. You can find the name with the function:

```
NSString* aString = NSStringFromSelector(aSelector);
```

Obj/C uses late binding. In other words, the system figures out what to do at run-time. The compiler defers information to the run-time system. This is very different from C++ where the compiler determines everything and there is almost no run-time overhead.

Any object can an entry point for any selector. So the way in which a delegate works if for the caller to first ask the object if it has an entry point:

```
(BOOL) [ object respondsToSelector:@selector(windowShouldClose:)] ;
```

So the caller (in our case window) will test the delegate and if it's provided, he'll call.

This is a very flexible system. An object can be a delegate of any number of objects. And a delegate of any number of classes of object. Of course trouble will occur if two delegate protocols have the same name for different types of delegation.

4) What's is a NIB ?

The NIB is the Next Interface Builder binary file. With XCode 3, Apple introduced the .xib file which is an XML representation of the nib. When XCode builds your application, he creates the NIB for each of your .xib's and stores the in the Resources directory of your application bundle. Here are the files in the application bundle of Clock.app:

`Clock.app`	
`Clock.app/Contents`	
`Clock.app/Contents/Info.plist`	XML file with application information
`Clock.app/Contents/MacOS`	
`Clock.app/Contents/MacOS/Clock`	Terminal application Clock !
`Clock.app/Contents/PkgInfo`	
`Clock.app/Contents/Resources`	
`Clock.app/Contents/Resources/Clock.icns`	The icon
`Clock.app/Contents/Resources/English.lproj`	
`Clock.app/Contents/Resources/English.lproj/InfoPlist.strings`	Localization file
`Clock.app/Contents/Resources/English.lproj/MainMenu.nib`	The NIB

At run time, NIBs are loaded into memory. The system creates an object which is represented by the NIB (using the object init method), then it loads the nib and calls the objects - (void) awakeFromNib: method and the object can carry out additional initialization at that time. The business of creating the GUI itself is handled for your application and IBOutlets (such as timeString) are set up on your behalf.

I've personally found it difficult to understand the NIB because it's binary. You can't look inside it. And it contains lots of magic such as bindings. For the Clock application however, the nib is very simple.

5) What is - (and +) in function declarations and definitions ?

- is an instance method (applies to the object). + is a Class method (applies to the Class) + = static member function in C++.

6) What are all those square brackets [in the code] ?

Obj/C source code is C and has the extension .m. Obj/C++ source code is C++ and has the extension .mm

Obj/C is an effort to endow object technology (modelled after small-talk) to be used from the C programming language. There really is only very prominent extension to 'C' in Obj/C and that is the message syntax:

```
[ object        name : arg name2: arg2 ..... ];
```

This is the syntax to "send a message" to another object. The call is synchronous. The message isn't put onto the Q for later response, it is handled immediately. For the life of me, I don't know why then call it a message as it effectively the same as a method invocation. Microsoft Windows on the other hand has two ways to send a message. PostMessage says put it on the message Q and it will be eventually processed. SendMessage says "send immediately without delay".

Obj/C provides only SendMessage capability. However every object is derived from NSObject which as the following method:

```
(void)performSelector:(SEL)aSelector withObject:(id)anArgument afterDelay:(NSTimeInterval)delay
```

So we can use this method (with delay = 0) to "drop" a message on the message queue for subsequent processing.

7) What are those `@"...."` strings ?

They are NSString* strings.
C++ provides const char* "abc" ascii and const wchar_t* L"abc" unicode strings. Obj/C has @"abc" NSString* strings.

8) What's the difference between a delegate and a notification?

An object can only have a single delegate. There is no chain or list of delegates. An object has a delegate or it does not.

However an object can have properties and any number of other objects can Observe the property. When a property is changed, all Observers are notified.

We're not using Notifications in this application, so I'm not going to discuss this further in this tutorial.

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

20100423	Updated note about using performSelector:withObject:afterDelay to do PostMessage.
20100422	Initial version.