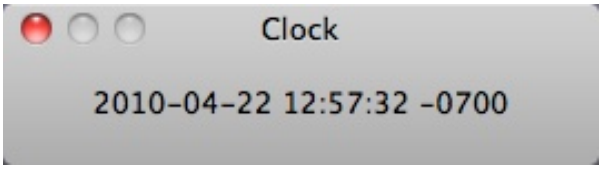
CLOCK TUTORIAL

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:

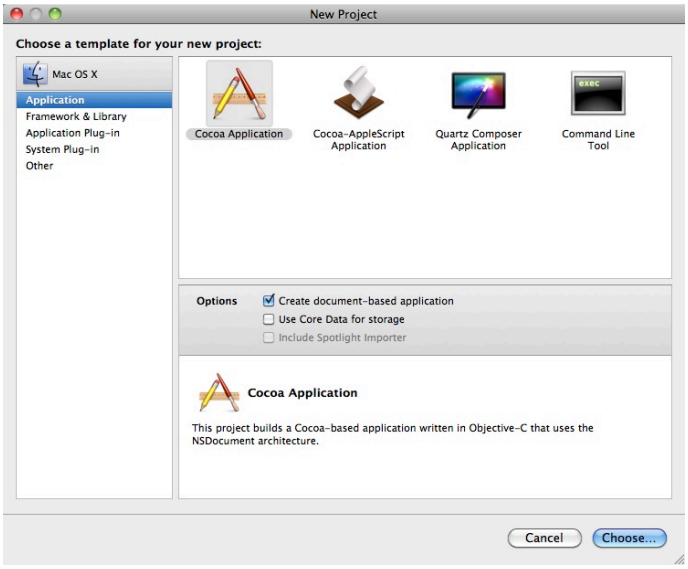
1) Creating a very simple application

2) Using NSAppDelegate and NSWindowDelegate

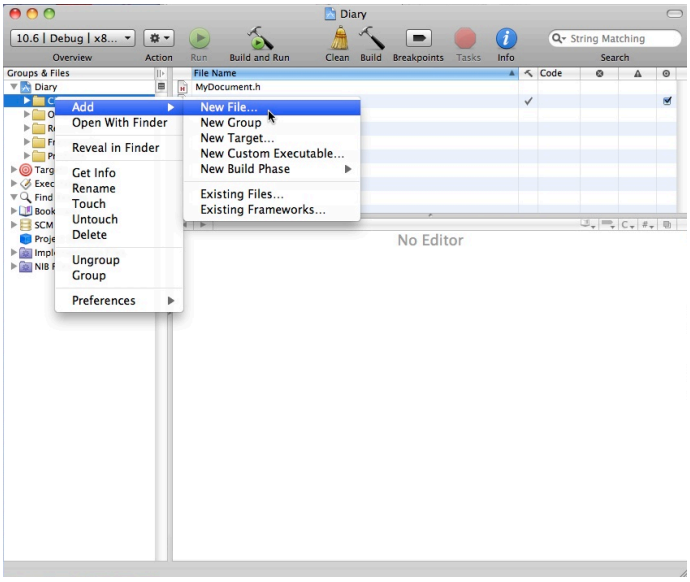
3) A little discussion about messages and selectors.

START IN XCODE

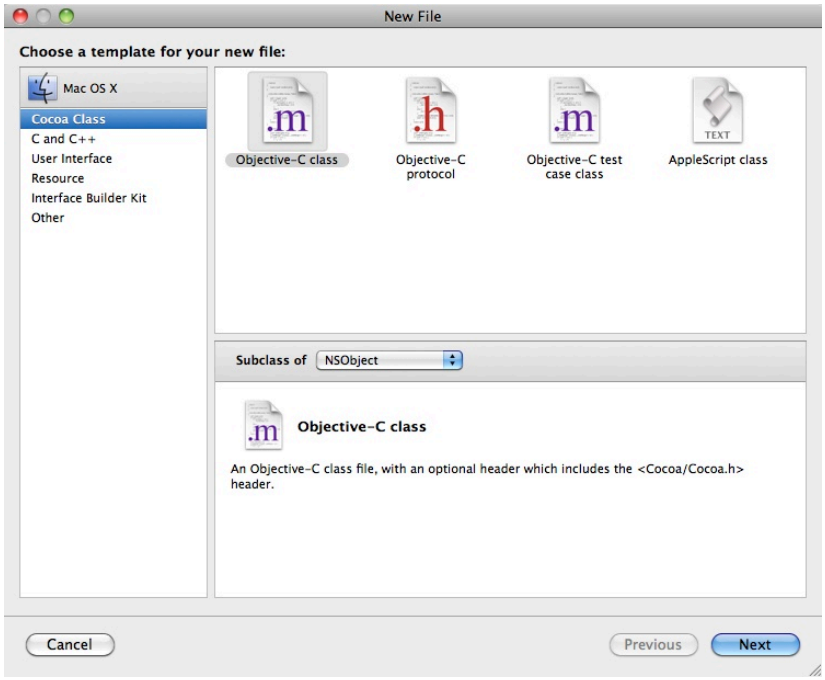
1) Start with an empty application. No need for a document:



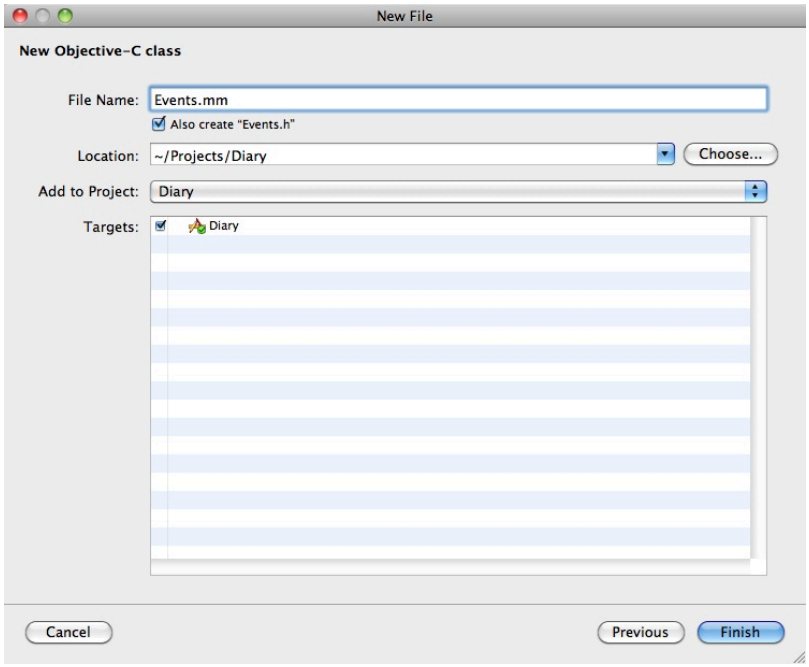
2) Select "Classes", right-click, Add > New File ...



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



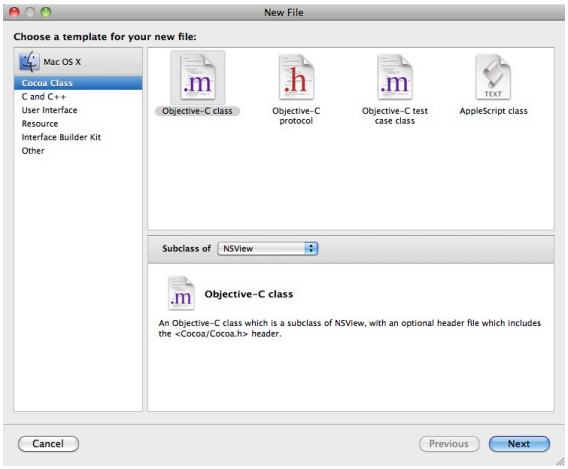
4) Call it Event.mm and check Also create Event.h



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

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

6) Create a view, we're going to use this for printing.



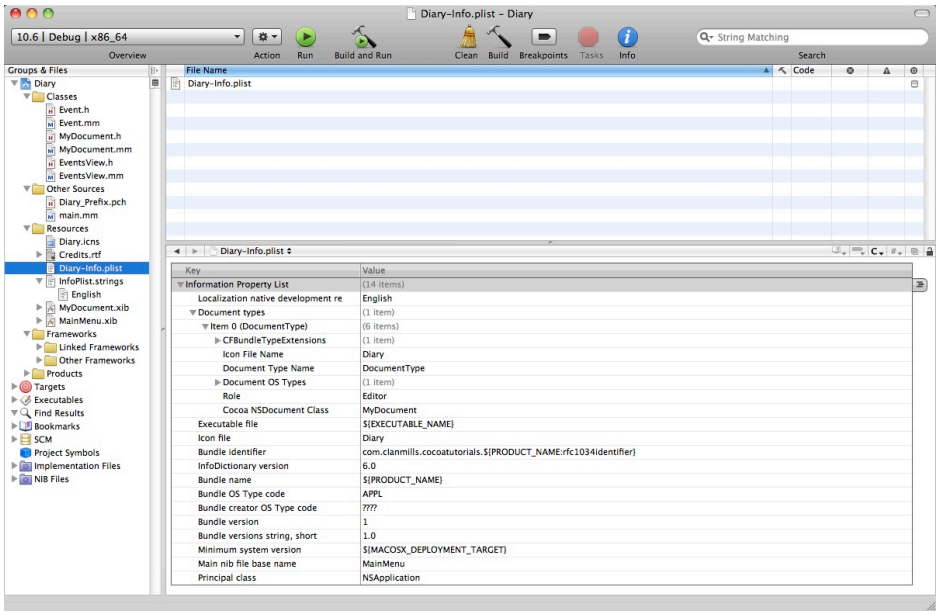
This is almost identical to creating Event.mm and Events.h **HOWEVER,** it's to be a subclass of NSView and it's to be called EventsView.mm (and EventsView.h).

7) Copy the following files from clanmills.com/files/Diary.zip into your project.

main.mm
Event.mm and Event.h
MyDocument.mm and MyDocument.h
EventView.mm and EventView.h
Diary.icns

Right-click on the Resources and Add File .... Diary.icons

8) Select Diary-Info.plist and edit it as shown



Update Icon File. And update the Document types - this will associate your icon with the extension .diary. And it will associate the extension .diary with your application. Later, we'll save documents and this will take care of the association.

Build and run your application. It is of course very boring and brings up a dialog box saying "Your document contents here". However it should show up with it's own icon - a cup of Cocoa.

IN INTERFACE BUILDER

1) Double click "MyDocument.xib" in XCode

This will launch Interface Builder of course. Rename "File's Owner" to be "MyDocument". Calling it "File's Owner" is (in my opinion) a bug. "File's Owner" confuses the user. To be fair to Apple, their thinking is that NIB has an owner object which has been created by the run-time system before the NIB is loaded. If you don't understand this, don't worry, just rename it.

2) Drag'n'Drop Event.h from XCode to Interface Builder "MyDocument.xib"

This will tell Interface Builder about our Event class (which we'll discuss later).

3) Go into Library/Objects/Objects and Controllers/Bindings.

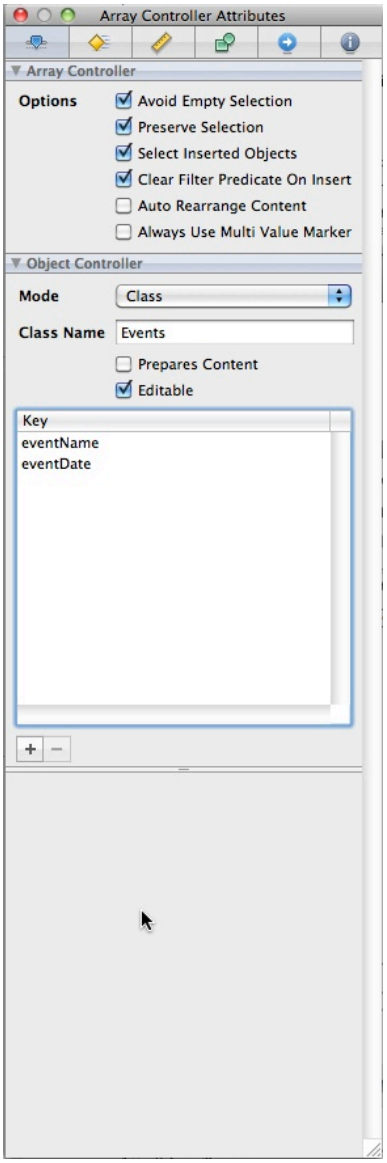
Pick up an NSArrayController (3 blue bricks) and drop it on MyDocument.xib. Rename him from "Array Controller" to "Events".

4) Use the Inspector (shift-Cmd-I)

Select "Events" in MyDocument.xib and use the Inspector to specify that the Array Controller is controlling an array of Event.
You'll realize that MyDocument has an NSMutableArray* events; The class of objects in the array are Event.

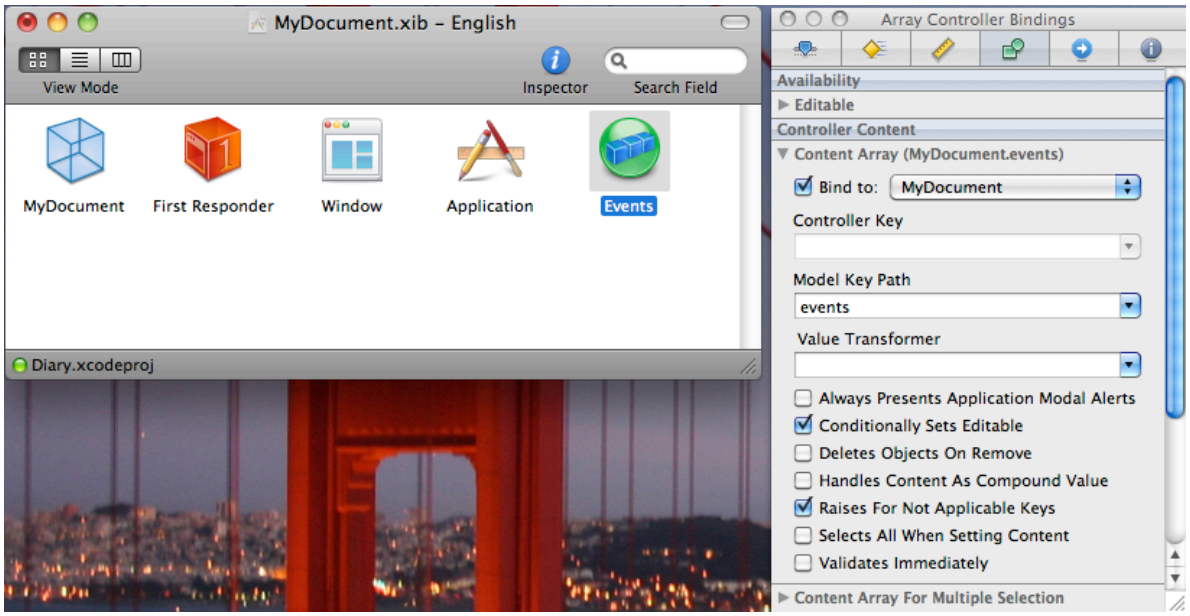
5) Use the + button and specify keys eventName and eventData

I'm rather surprised that Interface Builder doesn't give us any assistance or visual feedback when we use the + button. I'm sure one day Interface Builder will be more friendly in this area. For the moment, relax. Trust me!



6) Bind the MyDocument.events (an NSMutableArray) to Events (an Array Controller)

WARNING:This is very easy to get wrong. Be very careful to select Controller Content : Content Array (MyDocument.events)

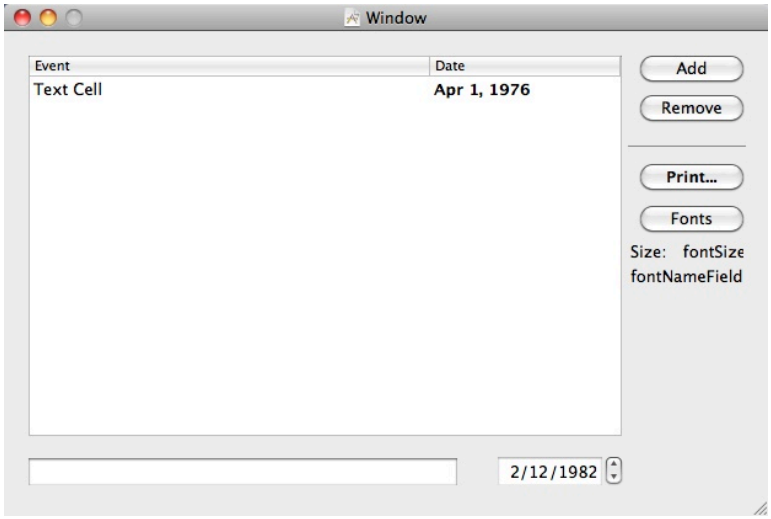


Your Array Controller (which we renamed Events) is to be bound to MyDocument.events. The array controller is managing your events array. This is a binding - any change in the MyDocuments.events object will be visible to the your ArrayController. And any changes to your array controllers objects will cause MyDocuments.events to change in memory. Binding does this for you.

HOWEVER: At this time, we have created invisible plumbing to connect MyDocument and Events. We haven't bound any visible elements of the UI. So we'll layout the UI and have a break. Then we'll connect all the visual stuff together.

8) Layout the controls in the dialog box.

Remove the "Your document contents here" label. Drag a TableView from Library/Objects/Data Views/Table View to occupy most of the space. Then add buttons, labels, text fields and a date picker.



Save in Interface Builder. Build and Run from XCode. The UI should appear. However nothing will work until we hook the UI to our code.

9) Save Everything. Make a back up and have a break.

The next part of the tutorial is rather difficult. There are lots of different ways to select different elements from the table view and when you get things wrong, you'll probably find it impossible to fix.

So, I recommend you:

Terminate XCode and Interface Builder
Make a back up of your project
ditto ~/Projects/Diary ~/Projects/Diary.good

When things go wrong, you can ditto the good guy back to this restore point.

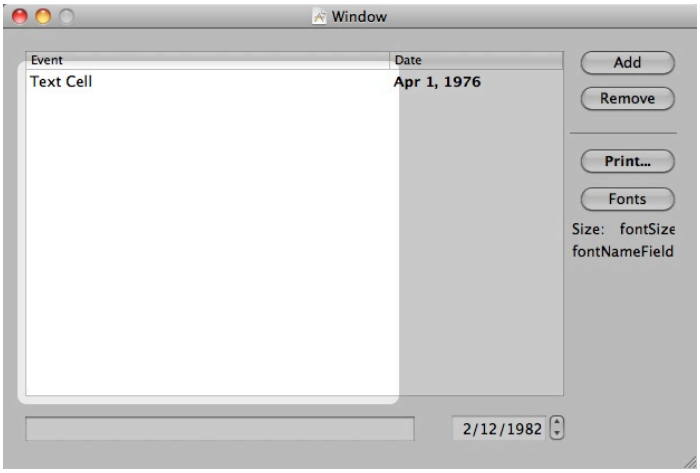
Incidentally, XCode does have something for make a 'restore point' in your project. I've never used it. It's probably pretty good. I prefer to use ditto because I know for sure exactly what I have.

HAVE A BREAK

IN INTERFACE BUILDER

WARNING: This stuff is tricky. Selecting the correct part of the text view control is mission critical. You can waste a lot of time if you haven't selected things correctly.

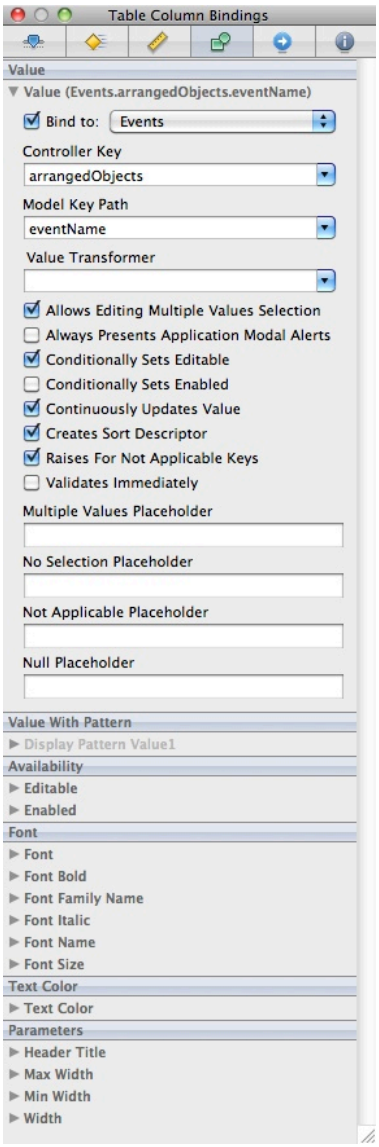
1) Select the Text Cell below Event



2) Use the Inspector to set the Table Column Bindings

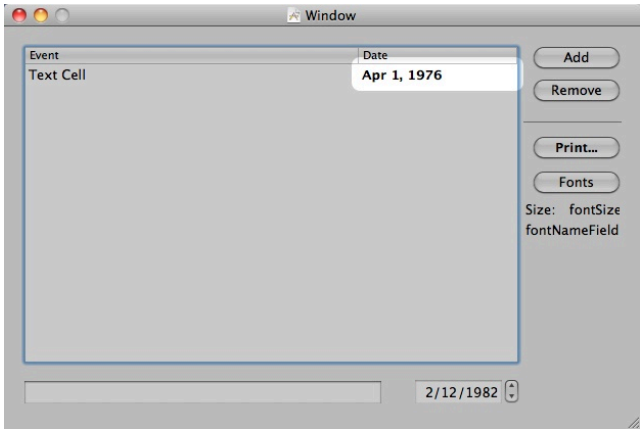
WARNING:
You want to use Table Column Bindings: **Value** (Events.arrangedObjects.eventName).

The Table Column Bindings can be applied to Editable, Enabled and other categories You want **Value**.

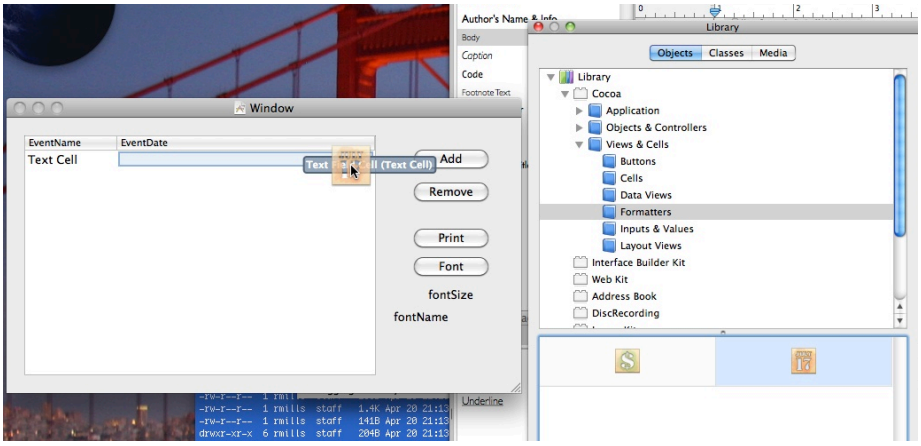


3) Do a similar operation on the Date Column

4) Add a date formatter to the event date cell (not the column, the cell)

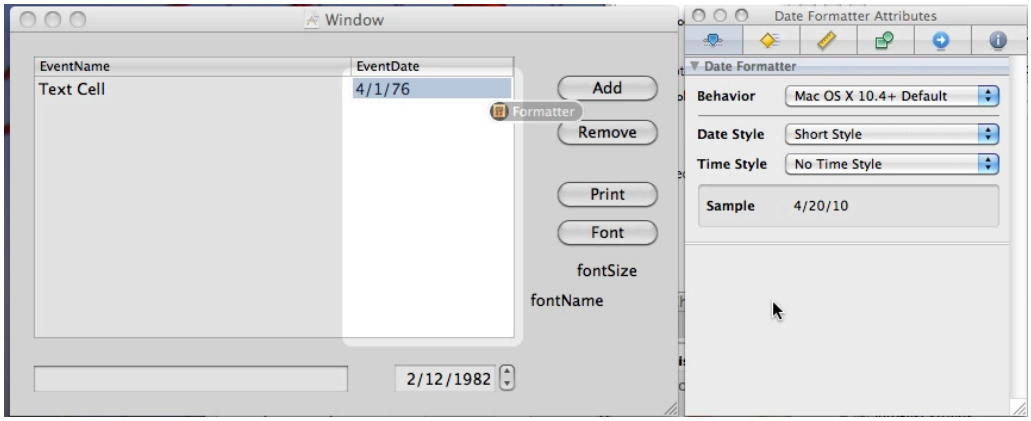


5) Drag'n'Drop a Date Formatter onto the Text Cell



6) The next step is incredibly difficult

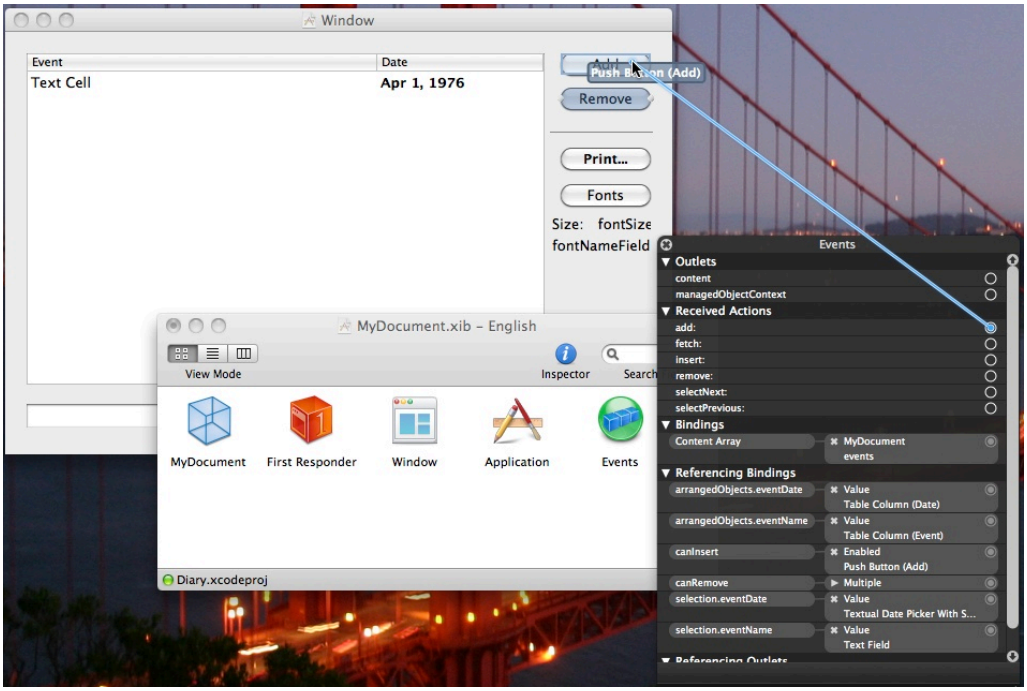
Interface Builder now proceeds to make the date invisible. Click and clunk until you get the properties of the formatter on screen:



Select the properties of the formatter I've used Date: Short Style, Time: No Style). This will show the date in local format.

7) Select the "Events" in MyDocument.xib (this is the array controller).

Right-click to get its Actions and connect add: to your Add button, and "remove:" to your Remove button.



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

Press the "Add" button and "Remove Buttons". You should be adding and removing diary items. You should be able to save and open documents. Amazing.

Time for another break. Then we'll discuss how the code works. And then we'll deal with the printing business. You're getting there.

HAVE A BREAK

HOW THE CODE WORKS

The whole application is less than 500 lines of code. And let me list all the things that are provided:

1) You can read and write .diary files.
   NSDocument takes care of almost everything.
   You have to add code to write/read an Event to/from disk.
   NSKeyArchiver, NSKeyUnarchiver and NSCoder look after the formatting in XML
   NSDocument looks after the FileOpen, FileSave and Print Panels

2) When you edit the document, you can choose to edit in the Table view, or the textFields below the table view. Changes are reflected instantly into the other without us providing any code at all. All we did was to bind correctly in Interface Builder.

3) The application has a custom icon. The file extension .diary is associated with your application.
   So you can double-click a .diary file and the application will launch.

4) You can edit more than one document at a time.
   NSDocumentController manages the multi-document environment (and is almost invisible to our code).
   Our MyDocument code does not know anything about multiple documents.

5) You can print with any text font.
   The pagination is automatic and respects both fontSize and the size of the currently selected printer page.
   We can select any printer and use different page sizes, color and layout such as n-Up printing.
   Our icon is watermarked below every page.
   The Font Manager's Font Chooser is provided.

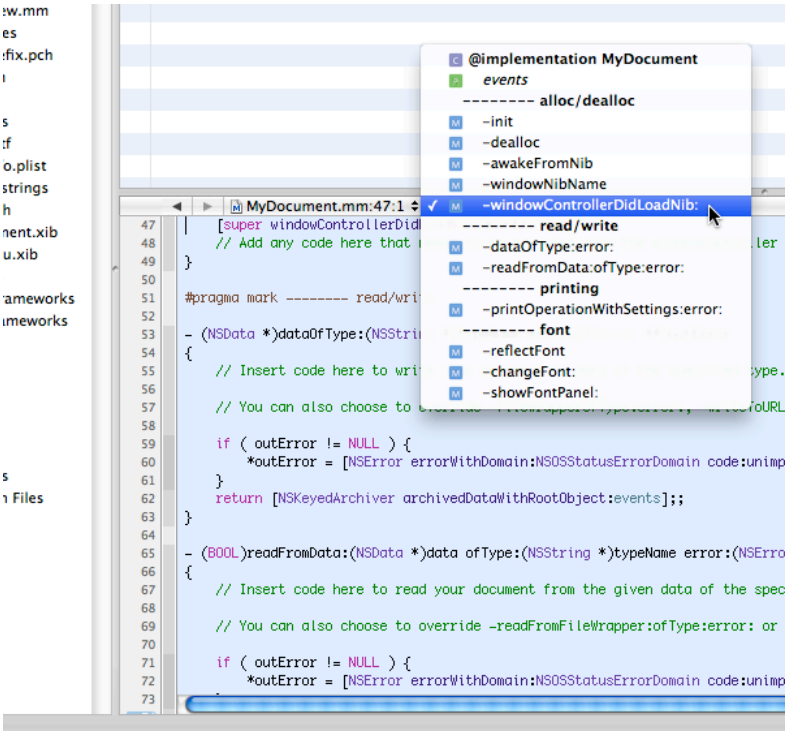
We could easily add undo. We could use CoreData and store the documents in SQLite.
The Cocoa framework is our friend - he is doing a lot on our behalf.

1) MyDocument.h and MyDocument.mm

MyDocument.h is a typical UI header file. We have several Outlets and Actions. And of course we need the events (an NSMutableArray). There's a longer discussion of Outlets and Actions in the Favorites Tutorial.

In this tutorial, I decided to use the @property keyword in MyDocument.h to define the events property. In MyDocument.mm, the @synthesize keyword is used to request the compiler to generate the setters and getters code. For sure this hides painful details of Obj/C at the expense of restricting the code to Obj/C 2.0 (XCode 3.0 and above).

You will see that I have also introduced the #pragma mark. This enables a menu to be provided for code navigation in XCode:



You can see that I've divided the code into 4 sections:

alloc/dealloc		This is very simple and obvious code
read/write		The two functions here are provided by the Wizard. I've added a line of code to each function:
	read:	`return [NSKeyedArchiver archivedDataWithRootObject:events];`
	write:	`[self setEvents:[NSKeyedUnarchiver unarchiveObjectWithData:data]];`
printing		One very simple function which I'll discuss below when I discuss EventsView.h and EventsView.mm
font		3 small functions for interacting with the FontManager. Discussed with EventsView.{h|mm}

The function isDocumentEdited is used by the NSDocumentController class to handle dirty documents and discussed later.

2) Event.h and Event.mm

These define the Event and how to read and write Event to memory (with NSCoder).

Event.h is very straightforward. It defines the properties of the event (a string for an eventName, and a data for an eventDate).

Event.mm provides some code for initializing events. That's more complicated that it needs to be. I added some C code to initialize events from the calendar. I did that for my debugging convenience. In a real application, you could:

1) init the eventDate to be today and give it a name such as "no name yet".
or
2) display a Panel and ask the user to enter details about the event.

The code to write an event to/from the disk is very simple. You have to implement initWithCoder (to convert data to an object)

```
- (id) initWithCoder : (NSCoder*) coder
{
      [ super init ] ;
      [ self setEventName: [ coder decodeObjectForKey : EVENT_NAME]] ;
      [ self setEventDate: [ coder decodeObjectForKey : EVENT_DATE]] ;
      NSLog(@"initWithCode %@ %@",eventName,eventDate);
      return self ;
}
```

You have to implement encodeWithCode to write the data for an object:

```
- (void) encodeWithCoder : (NSCoder*) encoder
{
      [ encoder encodeObject : eventName  forKey : EVENT_NAME ] ;
      [ encoder encodeObject : eventDate   forKey : EVENT_DATE ] ;

      NSLog(@"encoding eventName, eventDate = %@,%@",eventName,eventDate);
}
```

The method isEqualTo: is used by MyDocument isDocumentEdited: and is discussed under "dealing with dirty documents" below.

3) EventView.h and EventView.mm

These objects are used to print your diary.

As you know in software, there's more than one way to do anything. In fact there's usually an infinite number of solutions. However what I chose to do was:

> When you press the 'print' button, I create an EventView object.
> EventView has a copy of the events (it isn't a pointer to the objects, it's a copy)
> EventView is initialized with the current font (fontName and fontSize)
> We could make this more complex by including more font attributes such as color and underlining.

The framework (NSDocument) takes care of calling for the Print Panel, creating previews, paper sizes, orientation (landscape, portrait). As far as our code is concerned, we are simply painting the data on a rectangular area.

The user interface (in MyDocument.h) has a couple of textFields which are used to display the text font. When you click the "Fonts" button, I display the system's NSFontManager's shared text font chooser. We use the "reflect" method to provide a callback and the system calls us when the user select fonts. The UI in MyDocument is updated. I explain below how to do this in Interface Builder.

The longest stream of code in EventsView.mm is the search for the imageRep to be used. Tutorial ix has a longer discussion. The code to locate the path to the icon is interesting. It looks in the application bundle's plist file and figures it out from there.

DEALING WITH DIRTY (NS) DOCUMENTS

The NSDocumentController is doing rather a lot of work for you behind the scenes. Our code is unaware that the application can operate on multiple documents. You can open "Robin.diary" and "Prince.diary" at the same time. Edit one or both and save. You can even have multiple "untitled" documents which the document controller will conveniently name as "Untitled" and "Untitled 2".

Because we have subclassed NSDocument to create the MyDocument class, the NSDocumentController (which is managing the application windows) will know there are changes to be saved. By default (if we don't define this), it will return NO and you will loose changes to document if you don't save them.

If you define this function to return YES, You should be alerted when documents have been changed. However you'll also be alerted when changes have NOT been made.

So we need to have a way of knowing that changes have been made. If we used NSUndoManager, we could simply ask him if he has anything to undo and return that result. However I haven't used NSUndoManager, so we need to do something for ourselves.

I decided to add an array of events (called ovents = old events). The read/write and init code has been modified to update ovents when necessary. So MyDocument isDocumentEdited: method has to compare ovents and events. He does this by comparing firstly the number of events in each. If they are the same, I loop over both and call their isEqualTo method for each event. If both documents have the same events in the same order, they are considered to be equal. I rather like this definition as you can delete and reinsert events manually and the documents can be equal. An undo manager would be unable to know that this type of change is benign and should be forgiven.

MORE INTERFACE BUILDER FUN'N'GAMES

Although we've come a long way, we've still got a few details to implement. Specifically, we'd like to:

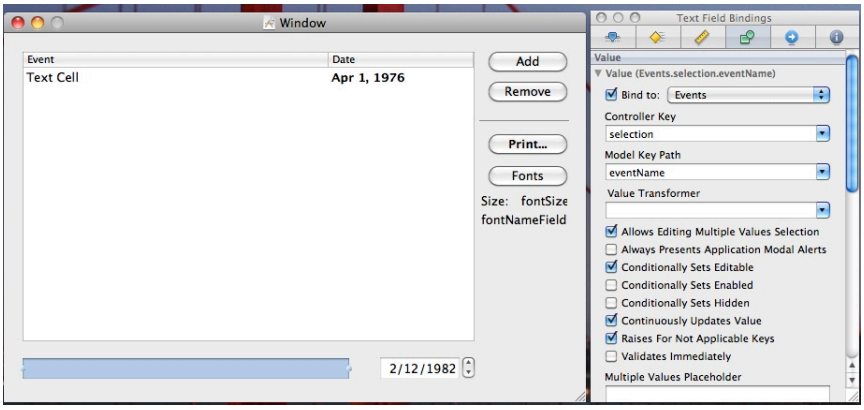
1) Be able to edit the name of the event in either the table view, or the text field below.
2) Update the date with the Date Picker
3) Enable/disable the Add/Print/Remove buttons

No items in the table view:	Add: Enable	Remove: Disable	Print: Disable
Items in the table view:	Add: Enable	Remove: Enable	Print: Enable

1) Bind the text view to the eventName

Here's what to do. Select the field. Display the Text Field Bindings (cmd-shift-I) and check "Bind to Events", Controller Key = "selection" and Model Key Path = "eventName". This says display the currently selected event.eventName in this field.

Be sure the check "Continuously Updates Value", then when you type, the TableView will be updated at the same time.

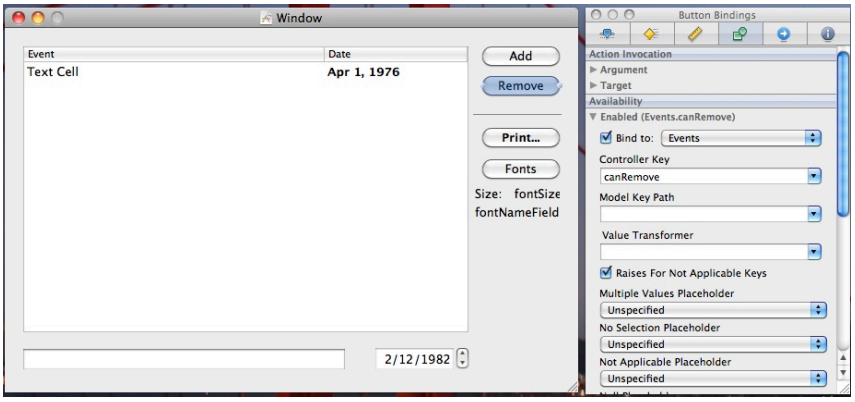


2) Bind the date picker to the eventDate

This is almost identical to the procedure the bind the eventName (step 1, above). You'll probably also want to inspect the bindings of your table view cells to ensure they also have "Continuously Updates Value" checked.

3) Enabling/Disabling Add/Remove/Print Buttons

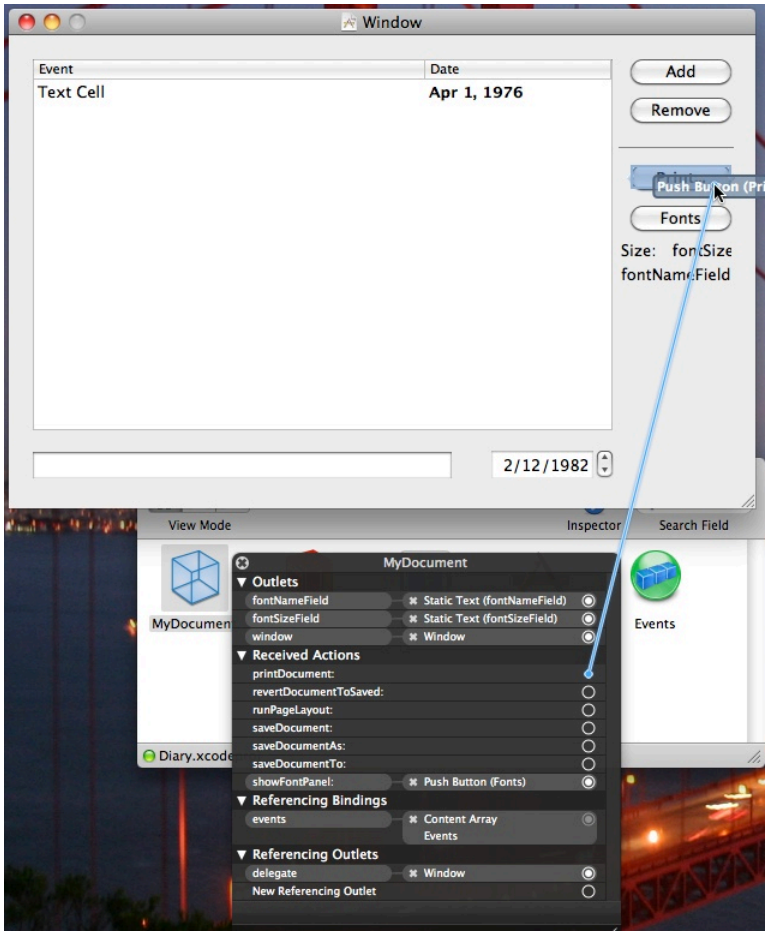
Take Care: This is similar to setting the binding, however we're not binding Value. We're binding **Enabled** and we use controllerKey "canRemove" which has been provided automatically by the ArrayController.



You may recall the warning earlier in the tutorial about being careful to bind Value. Now you're binding Enabled. Other binding categories are available and we won't touch them in this tutorial.

4) Connect the print button action to the document

The purpose of the "Print" button is to print the document. So select "MyDocument" and right-click. Connect the printDocument: action to the "Print" button.



Alternatively, you can ctrl-drag from the "Print" button to "My Document" and when you release the mouse, you'll be offered a menu of actions.

5) Connect the fonts button to the document

This is almost identical to connecting the print button. In this case, we added the showFontPanel: method. In the case of the "Print" button, the printDocument: method was provided by NSDocument. We have to provide the code for showFontPanel and this was discussed above.

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

If you're in luck, you'll be saving, printing, updating. Everything should be working.

SOME FINAL THOUGHTS

I don't think this is an easy tutorial. In fact, it's downright difficult. I didn't find it easy to document. You may find it hard to follow.

However I'm sure you'll agree that the amount of code required to make all of this work is remarkably small. If you're going to be working in this type of application (and many office applications are very much like this), then you probably want to practice this tutorial until you can do it in about 5 minutes. (you'll have to simply cut'n'paste the code of course to do it so quickly).

The real challenge of this tutorial is:

1) To understand how Interface Builder is creating the marriage between the UI elements and the your data.
   In particular to understand how the Array Controller works and provides glue between your arrays and the table view.

2) To understand the concept of a binding.
   This enables data in memory and multiple UI controls to remain in-sync without writing any code (the framework does it).

3) To understand how much work is being done on your behalf by the following classes:

   NSDocument
   NSKeyArchiver and NSKeyUnarchiver
   NSCoder
   NSFontPanel and NSFont
   NSPrintInfo and NSPrintOperation

4) This is a little introduction to the world of Quartz, the MacOS-X imaging model.
   In this simple application we've been able to take advantage of compositing to write a watermark logo on every page.

5) And just for fun, I've thrown in the subject of locating the icon in the application bundle and using it as a watermark.

I hope you enjoyed this tutorial. I know you'll tell me if you find errors or can't follow it.

Most of all, I hope you've learned something from this tutorial.

LICENSE


```
//
//  Diary.pdf
//  This file is part of Diary
//
//  Diary is free software: you can redistribute it and/or modify
//  it under the terms of the GNU General Public License as published by
//  the Free Software Foundation, either version 3 of the License, or
//  (at your option) any later version.
//
//  Diary is distributed in the hope that it will be useful,
//  but WITHOUT ANY WARRANTY; without even the implied warranty of
//  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//  GNU General Public License for more details.
//
//  You should have received a copy of the GNU General Public License
//  along with Diary.  If not, see <http://www.gnu.org/licenses/>.
//
//  This file is original work by Robin Mills, San Jose, CA 2010 http://clanmills.com
//
```


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

20100428	Updated to handle "dirty" documents by adding MyDocument isDocumentEdited: method and explanation.
20100421	Completed.
20100420	Initial incomplete version for Prince Kumar.