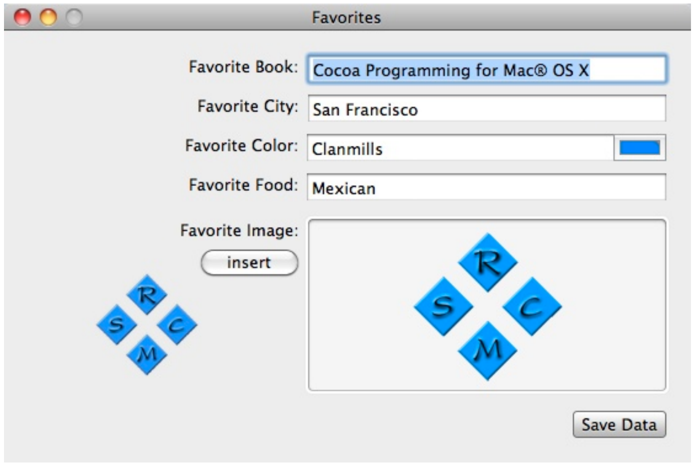
FAVORITES TUTORIAL

CREATING, STORING AND UPDATING PREFERENCE FILES

Mr Prince Kumar, a reader on http://www.cocoaforum.com/ asked about writing data to file from a simple application to file. I asked him to send me his code, and after some negotiation and bug fixing, here's the result.

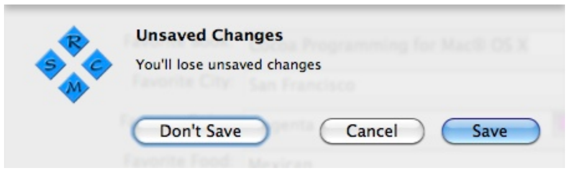

Favorites

Favorite Book: Cocoa Programming for Mac® OS X
Favorite City: San Francisco
Favorite Color: Clanmills
Favorite Food: Mexican
Favorite Image:

You can download the code from: **http://clanmills.com/files/Favorites.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 program has a single window/dialog box and the user can update his 'Favorites' (book, city, color, food and image). When you click "Save Data", the current settings are stored (in your preferences). If you wish to quit the program with unsaved changes, you'll be presented with the following "sheet":


Unsaved Changes

You'll lose unsaved changes

And the action after that is rather obvious. "Don't Save" simply exits, "Save" saves and exits, "Cancel" dismisses the sheet.

Now before I got into the details of how to create this program, I'd like to mention a couple of points about this:

1) The "Favorite Color" and the Color Picker are coordinated. When you open the Color Picker, the name of the color can be "White", "Red" .... and some other values. As you select different colors, the code to decide the name is part of this program.
2) The "Favorite Image" is displayed in an Image Well.
3) The program has its own Icon (which is also used in the About Box) and displayed on the Dialog box.
4) The default "Favorite Image" is the logo (which is stored in the application bundle).
5) You can use cmd-S (and the File/Save) menu to save the data.

ACKNOWLEDGEMENTS

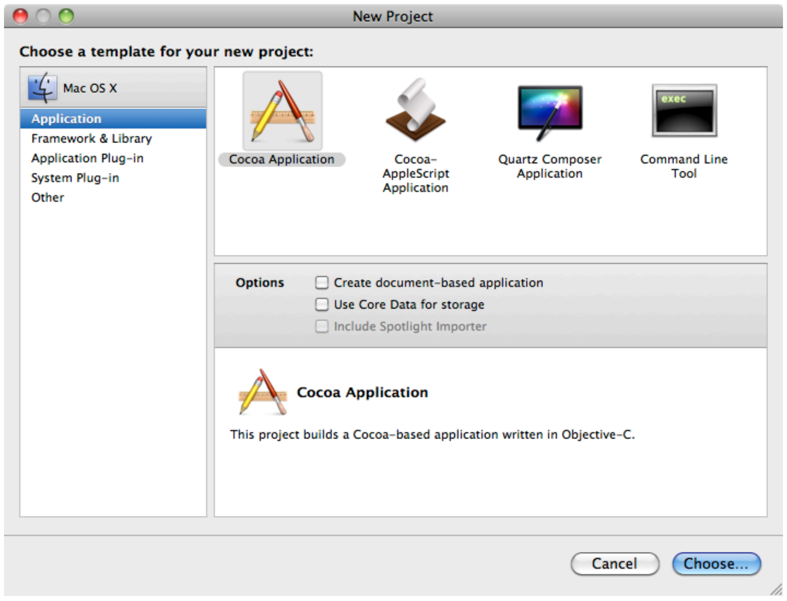
These notes are published under the GPL License. Mr Kumar (the original author) has kindly agreed to his code being published under GPL. If you are concerned about copyright, please drop me an email.

I'd also like to thank Prince for his assistance in getting this tutorial written. He provided most of the code of course, and told me about bugs I hadn't noticed. Thanks, Prince - it was fun working with you and talking on Skype.

THE RECIPE

1) Get the Wizard to generate a Favorites starter app

- Start with a new Cocoa App Favorites (no Document, no Core Data)
- just a vanilla application that brings up a window.
- Build and run the application. It's boring of course and simply displays a window. Well that's a good start.



Choose a template for your new project:

Options
- [ ] Create document-based application
- [ ] Use Core Data for storage
- [ ] Include Spotlight Importer

Cocoa Application

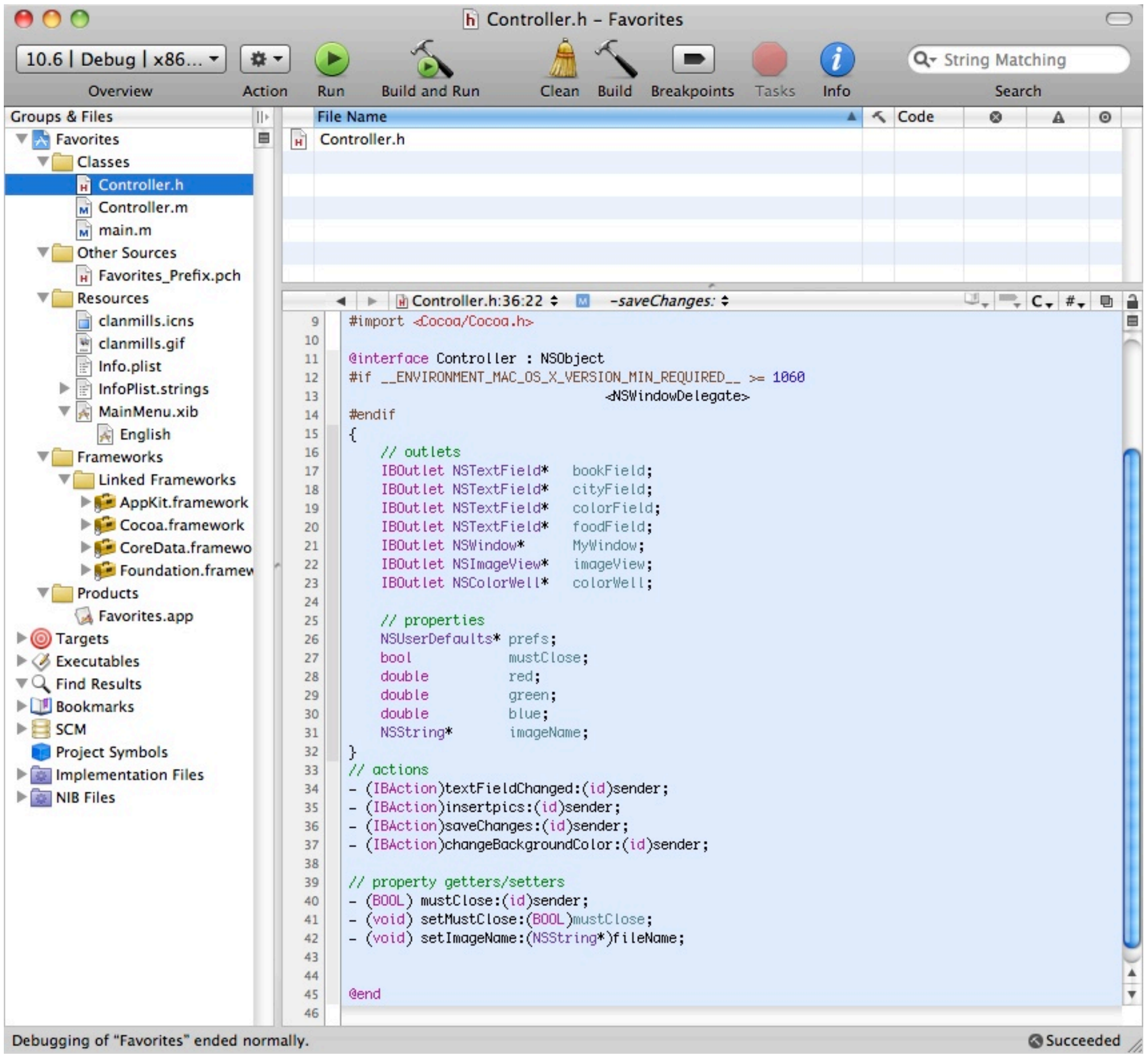
This project builds a Cocoa-based application written in Objective-C.

2) Define the class Controller to manage the Favorites Window

For the UI Outlets we'll need textFields for the book, city, color and food as well an imageView and and a colorWell.
We're also going to have properties for the window and favorites.
And data for the current rgb settings, the imageName and a magic property (mustClose) which I'll discuss in later.

The action performed by the UI require textFieldChanged, insertPics, saveChanges and changeBackgroundColor

- Add a file Controller.m (and .h) based on NSObject (the code is available on-line)
- in Controller.h, add the IBOutlets, data and Action (see the image below)
- In XCode, Link the Cocoa, CoreData, AppKit and Foundation framework



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

@interface Controller : NSObject
#if __ENVIRONMENT_MAC_OS_X_VERSION_MIN_REQUIRED__ >= 1060
                                            <NSWindowDelegate>
#endif
{
    // outlets
    IBOutlet NSTextField*   bookField;
    IBOutlet NSTextField*   cityField;
    IBOutlet NSTextField*   colorField;
    IBOutlet NSTextField*   foodField;
    IBOutlet NSWindow*      MyWindow;
    IBOutlet NSImageView*   imageView;
    IBOutlet NSColorWell*   colorWell;

    // properties
    NSUserDefaults* prefs;
    bool            mustClose;
    double          red;
    double          green;
    double          blue;
    NSString*       imageName;
}
// actions
- (IBAction)textFieldChanged:(id)sender;
- (IBAction)insertpics:(id)sender;
- (IBAction)saveChanges:(id)sender;
- (IBAction)changeBackgroundColor:(id)sender;

// property getters/setters
- (BOOL) mustClose:(id)sender;
- (void) setMustClose:(BOOL)mustClose;
- (void) setImageName:(NSString*)fileName;


@end
```

Debugging of "Favorites" ended normally.

3) Add the graphic resource clanmills.icns and clanmills.gif. Update Info.plist

- You'll have to set Bundle identifier in Info.plist to com.clanmills.${PRODUCT_NAME:identifier} - that's used to determine the name of your Preferences file.
- Update Icon file to clanmills (yes, it's not clanmills.icns) to use the icon from the resources.

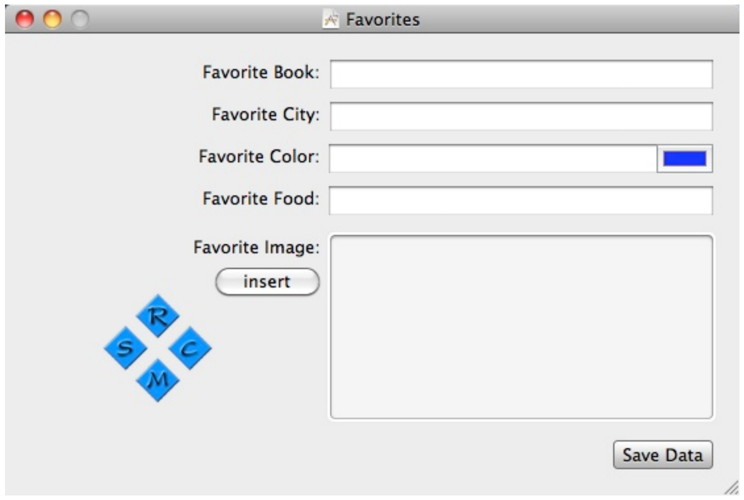
4) Add an instance of Controller to the NIB in Interface Builder

Open the MainMenu.xib in InterfaceBuilder (you'll find him in Resources - double click him to launch IB).

- Drag Controller.h (from XCode) onto the InterfaceBuilder MainMenu.xib pane.
- Drag an NSObject (a Blue Cube) from Interface Builder/Library onto the MainMenu.xib pane
- Select the new NSObject and set its class to Controller

5) Layout the UI in Interface Builder

- Drop the NSTextField,s and other controls from the library onto the Favorites Window in InterfaceBuilder.



6) Connect the UI widgets to your code

- Right click on the Controller object in the MainMenu.xib pane
- attach outlets: foodField to the TextField and so on.
- attach actions: insertpics Action to the 'insert' Button, SaveData to saveChanges and so on.
- Save MainMenu.xib in interface builder.

7) In XCode update your code in Controller.m (the code is available on-line)

- There's a much longer discussion of the code a little later in the report. For the moment cut'n'paste it and get it to run and we'll discuss how it works in a moment.

8) Compile and run

I hope it works for you. When it works, Congratulations. I know you'll tell me where you are stuck.

CODE DISCUSSION

1) init, awakeFromNib, saveChanges and hasChanged

init has to create a Dictionary to be used by the preferences. It populates the dictionary with defaults. Controller has a NSUserDefaults object called 'prefs' which holds the system together. The prefs object is a bi-directional channel to the preferences file. If you modify values in the prefs object, the file will be updated. However be aware that the system does cache this information and the changes may not be instantly available from the disk. However the XML and I/O to the disk is totally invisible to our code.

awakeFromNib is called when the UI is first build in memory (and before being display). This function reads the current Favorites (from the preferences files) and set the values appropriately on the display.

saveChanges sets the current values into the prefs object and the system takes care of writing the file for us.

hasChanged simply compares the current values settings (in memory) to the values stored in the preferences (on disk). It returns YES when there are outstanding changes to be saved. We'll need this during window and application close to decide to show the sheet and prompt the user to save/cancel/exit.

In fact the original idea was to use testFieldChanged and setDocumentEdited to provide the knowledge that the UI is out of sync with the preferences on file. I abandoned this as unreliable. It's rather irritation to change a value, then change it back to your original thoughts and be asked "Save Changes?" when nothing has been changed!

There's another UI/HIG consideration here. The "Mac Way" of dealing with preferences is to change them instantly without asking permission to save and without providing an undo capability. I'm a little unhappy with this as my Windows experience is usually to make changes and hit "Apply" to enforce them (while "Cancel" allows you to escape without change). The code here reflects my preference.

2) sheets and panels

The program uses an alert sheet and an a file open panel. A 'sheet' is a UI element that appears attached to its parent and usually partially obscures the parent. The "Don't Save / Cancel / Save" UI element is a sheet.

An NSOpenPanel is used to pick an image file. A Panel is a child dialog box.

The NSOpenPanel is a modal dialog box - the application thread waits for the Panel to be used or dismissed by the user. The programming is simple - you only have to call this in-line.

The Alert is slightly more complicated. When you display an Alert, your program immediately continues and you have to define a delegate to receive the user response to the sheet.

3) applicationShouldTerminateAfterLastWindowClosed, windowShouldClose and mustClose

The system calls windowShouldClose to see if it's time to close the window. If there are unsaved changes, we have to say "No" and ask the user. Now the panel delegate alertDidEnd knows what the user decided to do and if necessary does a close. Sadly we're now going round in circles if the user said "Don't save". The methods setMustClose and mustClose are used to avoid confrontation.

4) the color code

This is rather fun. The UI has an NSColorWell object and we connected that to backgroundHasChanged action. To that function is called as different colors are selected using the standard color pickers in the system. Now to add to the complication, different color pickers operated in different color spaces. However as the the NSColor object has the methods redComponent (green and blue), we can find the RGB values without being concerned about the color space. Color purist's will of course start to argue about the nature of the RGB value generated and the color profile in use. I'll leave them to argue about that without me. I also added a C function "colorName()" which will determine the nearest color name about which it's confident. So when you're close to red, the name will be "Red" and so on. If you're not close to my predefined colors, you'll get the name "Custom". I believe there is an object with the names and specifications of colors (similar to the HTML/CSS names) and although I intended to use that, I'll leave that to you the reader to add.

5) the image code

My original intention was to store in the 'favorite image' in the preferences. However this turned out to be more difficult than expected. The difficulty is in serializing (archiving) the binary image into a string for the preferences object (which converts it to XML). In the end I decided to make life easier and only store the path to the favorite image.

AND SOME OTHER MATTERS

1) The logo file is clanmills.gif. You only have to drop that onto the user interface in IB.

2) I used the Utility: /Developer/Applications/Utilities/Icon Composer.app to create clanmills.icns
In fact, I had to convert clanmills.gif in PhotoShop Elements to the RGB color space and then copy/paste the image into the 128x128 icon. You'll appreciate that Icons in MacOS-X can be defined at multiple resolutions and the system will select the source 'sub image' from the .icns file for its various purposes. In this case, I have only provided a single resolution.

3) You define the IconFile in Info.plist under the key "Icon file" (don't forget to include clanmills.icns in the project)

4) The default About Box displays the Icon - and of course it is also used in the Dock and by the Finder.

5) I also modified the "Bundle version" in Info.plist and this is reflected in the About Box and the Finder (cmd-I) on Favorites.app

6) I also modified the text in the menu bar in Interface Builder to say things like "About Favorites" instead of "About New Application". And I attached the File/Save menu item to the controller's saveChanges method. So cmd-S does a save.

7) Build settings in XCode are sometimes a nightmare as you're not sure how the system converts the UI into commands. XCode 3.2.1 appears to have hidden the command-line transcript somewhere so deep I can't find it. However that doesn't defeat me. Here's what I did in the terminal:

```
xcodebuild -showsdks
xcodebuild -project Favorites.xcodeproj -activetarget -sdk macosx10.6
```

and now I can see the build console. You'll see the sdk becomes -mmacosx-version-min=10.6

To find the compiler's predefined marcos:

```
touch empty.cpp ; touch empty.cpp ; /Developer/usr/bin/gcc-4.2 -mmacosx-version-min=10.6  -E -dM empty.cpp
```

... you'll see all the macros. And I discovered that macosx-version-min becomes:

```
#define __ENVIRONMENT_MAC_OS_X_VERSION_MIN_REQUIRED__ 1060
```

So this explains the mysterious 10.6 and created <NSWindowDelegate> declaration in Controller.h @interface statement.

CHALLENGES

1) Use a document model.
The current version of the code saves the data in the user preference folder, in ~/Library/Preferences/com.clanmills.Favorites.plist
You can read the preferences current settings with the command defaults read com.clanmills.Favorites It would be nice to save and store this information in a file - probably with a custom extension such as .favorites

2) Add Undo, Revert and enable/disable menu/buttons appropriately.
At the moment, the "Save Changes" button is always enabled - even when nothing has changed. And there's no way to undo the changes except to exit the program and start over. And we can't simply "revert" to get rid of all unsaved changes.

3) There's an extra-ordinary bug/feature/issue. Refactor MyWindow as myWindow and the exit/close code doesn't work correctly.

4) The color names are updated by the color picker. However if you type in the name of a color, the image well is not updated.

5) Store the image (not the image path) in the preferences file.

Revision History

20100409	Revised following comment by Price. Added many details. Fixed lots of typos. Added GPL to end
20100407	Initial version - send to Prince Kumar for comment.

LICENSE