# IX TUTORIAL

*IX = ICON EXTRACTOR*

*THIS IS A PROGRAM FOR CREATING PDF FILES FROM IMAGES*

During my work on the open source project Phatch, we started to discuss the design of the icon to be used on MacOSX. And I thought "you know, it would be very useful to be able to see all the icons on my mac". So the purpose of this program is to generate a PDF file which contains one page per icon, imaged at the highest resolution available.

Some of those icons are incredibly beautiful. They are examples of 21st century Industrial Art. I can imagine an art exhibition in 100 years from now in which these things are on the walls of an art gallery (of course most are horrible and deserve to die!).

The icons above are "System Preferences" (Apple) and "Firefox" (Mozilla). I hope Apple and Mozilla will forgive my breach of copyright, this article is seeking to extoll the virtues of your work and products.

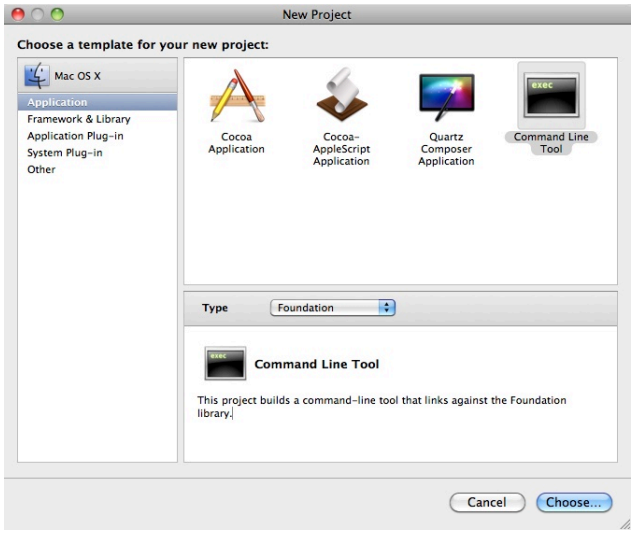
You can download the code from: **http://clanmills.com/files/ix.zip**

The program has a command-line interface. Of course it would be quite easy to have a GUI version of this code,, however I leave that to you the reader as a challenge. The code (and this document) has been published under the GPL license, so you are free to modify and use the code for your own purposes.

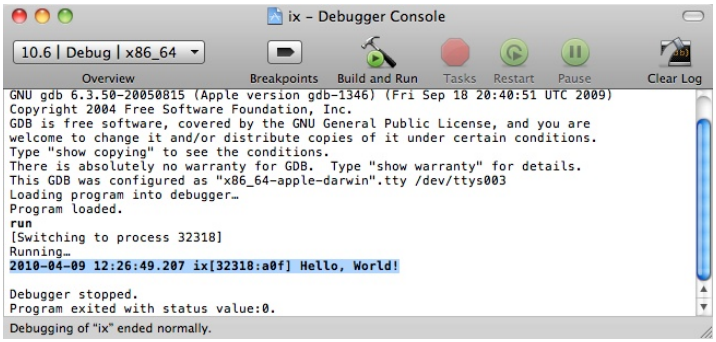
## THE RECIPE

### 1) Get the Wizard to generate a project for ix

- Start with a new Application Command Line Tool using type Foundation



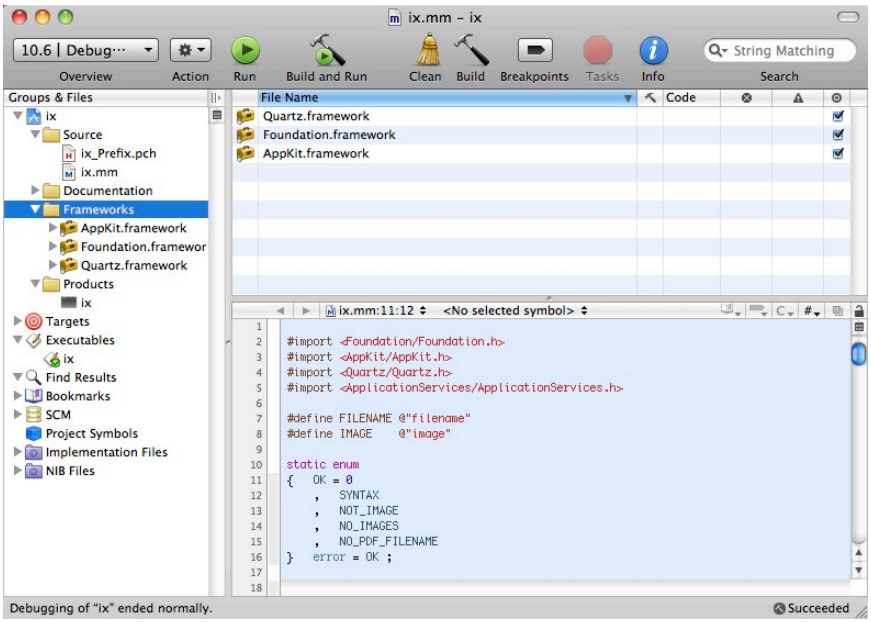
- Rename main.m to main.mm (this makes it Obj/C++ instead of Obj/C)
- Build and run the application. It's boring of course. Well that's a good start.



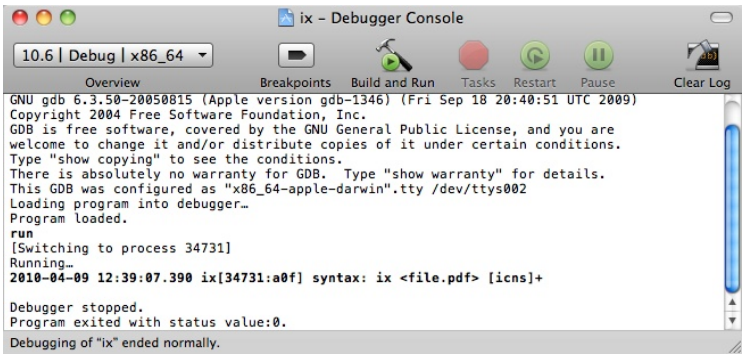

## 2) Paste the code from ix.zip into main.mm and link the frameworks

Just go ahead and get the code in ix.zip/main.mm and paste it into the application. We'll discuss how it works in a moment.

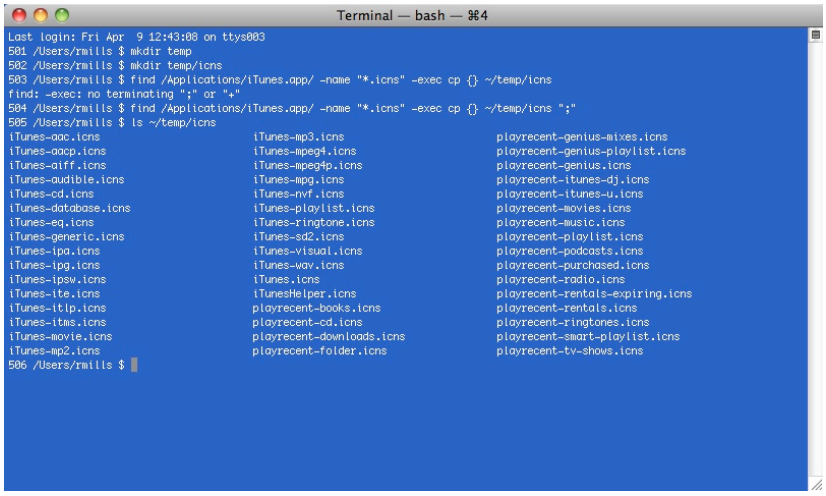
- Link Quartz and AppKit into the Frameworks.
- I renamed "External Libraries" as "Frameworks" and added them (Right-click/Add existing frameworks...)



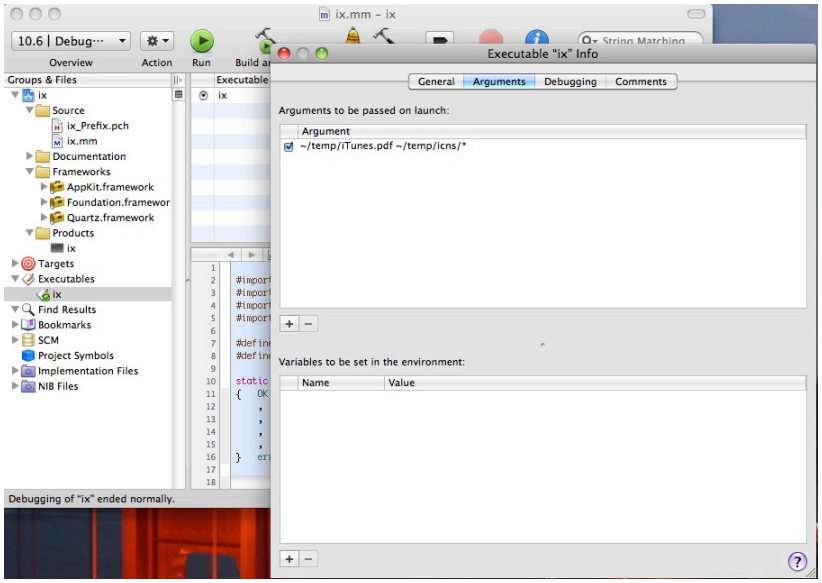
3) Run the application again, and you should get (almost as boring)



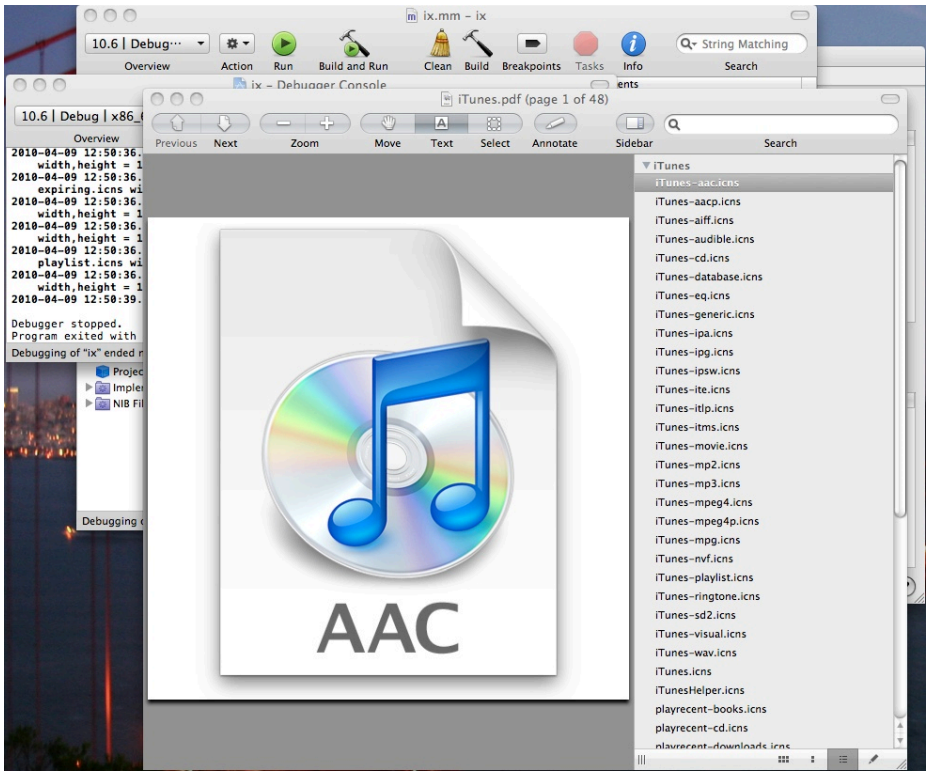

3) Use the terminal to get some Icons:




4) Update the projects "Executable" Info/Arguments to read your icons



5) Run the application again - not nearly so boring, is it?



# CODE DISCUSSION

## 1) command-line arguments and error management

The command line arguments for this program are very simple. The first argument (after the program name) is the PDF file to be created. After that follows the names of the icon files (well, they can be any image file in fact - PDF, GIF, ICNS etc).

I use a global enum type to track the error status of the program. The default OK = 0, so you can check status with if ( !error ). The functions which report errors also set the error variable and this keeps the code nice and tidy (in my opinion).. In fact, I decided to reduce NOT_IMAGE to being a warning, and that only required the assignment to error to commented in notImage()

## 2) converting the command-line arguments to Cocoa data objects

I represent the command line in two ways:

```
NSString* pdfFileName       the path to the PDF file to be created
NSArray* images             an array of (NSDictionary) objects to be displayed in the PDF file
                images[i]  << /FileName string /Image nsImageRep >>
```

When I open an image file, I look for the highest (width or height) image representation and store it in the dictionary.
I think the Apple's Icons are the only image file type which offer multiple resolutions within a single image. JPEGs often contain low resolution previews which are used by the digitial camera to render on the device display (and by Finder, I believe).

## 3) sorting the graphics file by name

A little later in the program, we're going to label every page in the PDF file with the name of the file from which it came and I want the pages to be sorted by name. It's entirely like that two application will have an icon file called myApp.icns (or something equally generic). All the myApp.icns files will be consecutive pages in the output file.

## 3) creating and displaying PDFs

Writing a PDF file is very simple:

```
PDFdocument* document = [[PDFdocument alloc]init];
.... populate the document ...

[document writeToFile:pdfFileName];
```

To display it, I used the system() api and the command open pdfFileName. I'm sure there's a better (and safer unicode) way to do this, however this is sufficient for my purposes.

Adding the content with the imageReps (from the dictionary in the images array) is straightforward:

```
NSImage image = [[NSImage alloc]init];
[image addImageRepresentation imageRep];
PDFpage = [[PDFpage alloc]initWithImage image];
[document insertPage:page atIndex:pageNumber];
```

## 4) adding the Table Of Contents information (PDF Outlines)

This part of the program took a while to figure out. I encountered several issues with the Apple Documentation and sample source code and I have documented those in the source code. I'll leave you to read the code.

## 5) other uses for this program

This program was designed to work on .icns file - however it works just as well on any image file. ix GrandCanyonSunday.pdf ~/Dropbox/Photos/2010/GrandCanyon/Sunday/* produces a beautiful PDF 250mb file in 10 seconds of photos I took on Vacation.

```
find /Applications/iTunes.app -name "*" -exec echo \"{}\" ";" | xargs ix iTunes.pdf
```

This creates a PDF with every graphic resource from iTunes.

# CHALLENGES

## 1) Add a GUI

This is a rather interesting challenge. The finder recognizes an application because it's stored in an "App Bundle". This is a directory with the extension .app and contains sub-directories such as Contents/Resources. Very interestingly, it usually contains a command-line program MyApp.app/Contents/MacOSX/MyApp

The challenge is the create the ix.app. When it's run by the Finder (double click in Finder, or use terminal command open ix.app) it should present a GUI. When run from the terminal $ ix.app/Contents/MacOSX/ix - it should behave as the command line version above. Two programs for the price of one!

## 2) More command line options

The field is open on this one. You could add arguments to tell it to put borders and titles around PDF files and all sorts of stuff.

## 3) Respect PDF input images as PDF

Don't render them as images, copy their content.

## 4) Treat every page in an input PDF file as an image

# 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

| | |
|---|---|
| 20100409 | Initial version. |