XcodeCapp 3.0

What is XcodeCapp?

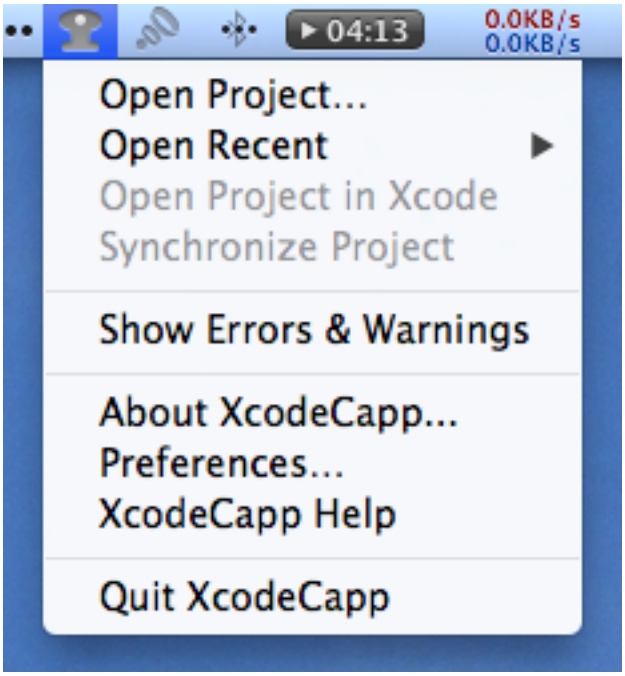
One of Cappuccino's greatest features is its ability to use Xcode 4 to create the user interface for your web applications. Xcode creates .xib files which then must be converted by the command line utility `nib2cib` to .cib files for use with Cappuccino. But beginning with Xcode 4, there is no way to directly create outlets and actions without editing Objective-C header files.

XcodeCapp acts as a bridge between Xcode and Cappuccino. It performs several main functions:

- Creates an Xcode project where you can edit your Cappuccino project's xibs.
- Reads your source files and automatically creates outlets and actions in the xib file.
- Automatically converts xib files to cib files when the .xib file is modified.
- Listens for changes to source files and xibs and updates outlets and actions in the xib file accordingly.

Using XcodeCapp

XcodeCapp is very easy to use and requires very little user interaction. When you build Cappuccino with jake, it will create a symlink to the XcodeCapp application in your Applications folder. Launch XcodeCapp from there and the XcodeCapp icon will appear in your menu bar. Clicking on the icon will display the XcodeCapp menu:



When the status icon is light gray, XcodeCapp does not have a project open.

Opening a Project

First you have to tell XcodeCapp what Cappuccino project to open.

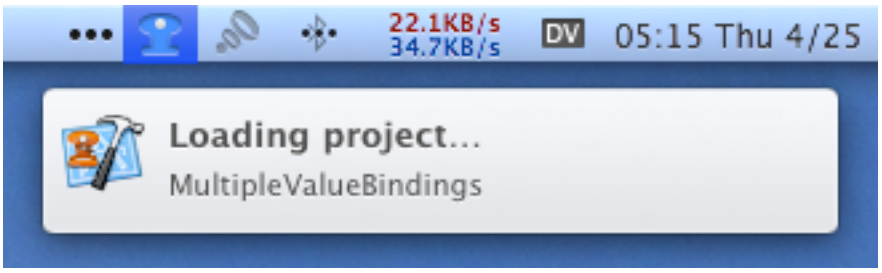
1. Select "Open Project…" from the XcodeCapp menu. A folder chooser dialog will appear.
2. Navigate to the root folder of your project — the one that contains index-debug.html and Jakefile.
3. Click Open.

Once you open a project, XcodeCapp does the following:

1. Creates an Xcode project with the name of the project directory, if no such project already exists.
2. Parses source code in the project directory and all non-Cappuccino subdirectories (such as Build, Frameworks, etc.).
3. Parses source code in user (non-Cappuccino) frameworks.
4. If no cib file exists for a given xib file, converts the xib using `nib2cib`.

5. If the preference is set to auto-open Xcode projects, opens the Xcode project.

When XcodeCapp is processing files, the status icon changes to light blue.



During source parsing, XcodeCapp creates an invisible folder called ".XcodeSupport" in the root project directory. This folder contains files created by XcodeCapp that act as the bridge between Cappuccino and Xcode, so you should never directly modify those files. You will probably want to ignore this folder in your IDE and source code management system as well.

After XcodeCapp has opened a project, it will listen for changes to the following files anywhere in the project:

- `*.xib` – Xcode 4 interface builder files
- `*.j` - Objective-J source
- `.xcodecapp-ignore` - Specifies files XcodeCapp should ignore. See "Ignoring Files and Folders" below.

When XcodeCapp has a project open and is listening for changes, the status icon changes to dark gray.

The most recently opened project is inserted at the top of the Open Recent submenu in the XcodeCapp menu. By default, the recent project list is limited to 20 projects. You can change that number in the Preferences window. See "Preferences" below for more information.

If you quit XcodeCapp while a project is open, by default when you reopen XcodeCapp it will reopen the most recently opened project. You can change this

behavior in the Preferences window. See "Preferences" below for more information.

Declaring Outlets and Actions

The most important function XcodeCapp performs is to create outlets and actions for use with Xcode. You create outlets by prefixing them in your source files with `@outlet` or `IBOutlet`. For example, in the class below, there are four outlets defined:

```
15  @implementation AppController : CPObject
16  {
17      @outlet CPWindow            theWindow;
18      @outlet CPSlider            slider1 @accessors;
19      @outlet CPSlider            slider2 @accessors;
20      CPObject                    foo @accessors;
21      CPString                    currentDate @accessors;
22      CPString                    currentTime @accessors;
23      CPArray                     people @accessors;
24      @outlet CPArrayController   peopleController;
25      BOOL                        allowColorChange @accessors;
26      BOOL                        canEditPeople @accessors;
27  }
```

Similarly, you declare actions by declaring the return type of the method to be `@action` or `IBAction`. For example, the following method can serve as an action for a control:

```
56  - (@action)setNullArgument:(id)sender
57  {
58      [self setFoo:[sender state] === CPOnState ? nil : self];
59  }
```

Notifications

XcodeCapp shows notifications whenever it loads a project or processes files. If you are using Mac OS X 10.8+, notifications will appear in Notification Center. Otherwise they will appear according to Growl's preferences if Growl is installed, or in a simple notification window if Growl is not installed.

When a project is loaded, XcodeCapp only shows notifications for the beginning and end of the load. After a project is loaded, by default XcodeCapp shows a notification for each modified source file that is processed. You can turn file processing notifications off in the Preferences window. See "Preferences" below for more information.

To completely turn off all notifications for XcodeCapp, use the Notifications control panel (OS X 10.8+) or Growl preferences.

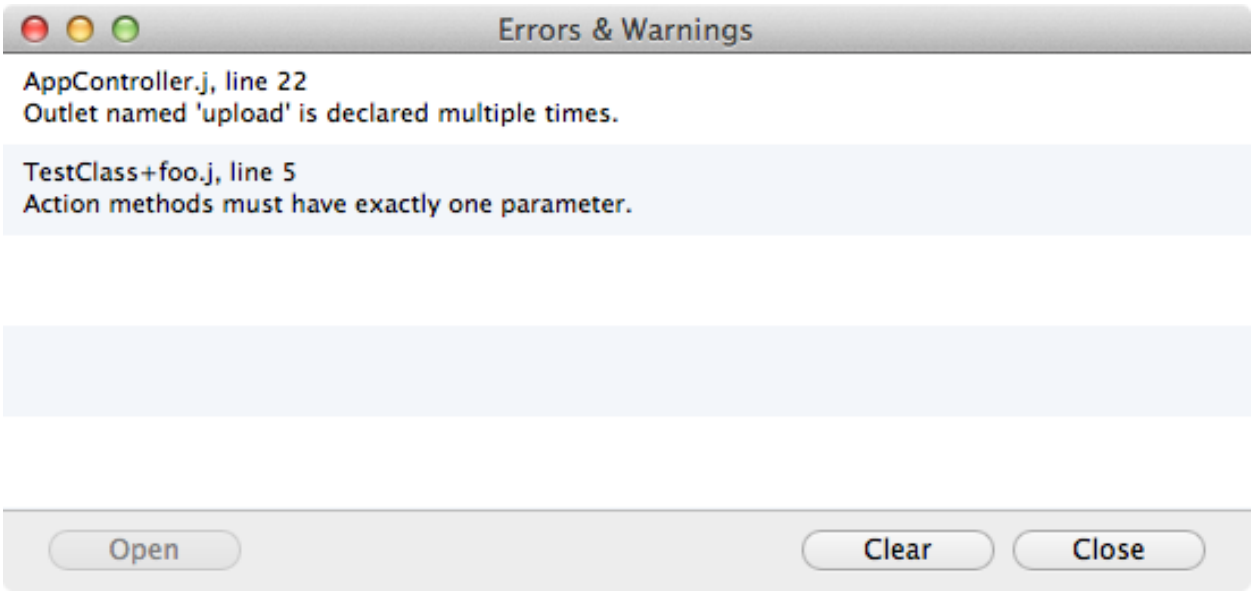
Note that notifications remain on screen for a minimum amount of time, and thus may lag behind the actual processing of the files.

Handling Errors

If an error or warning occurs during the parsing of an Objective-J file or the conversion of a xib, XcodeCapp notifies you in two ways:

- If an error occurs, the status icon becomes red.

- If an error or warning occurs, by default XcodeCapp will open the Errors & Warnings panel and display information about the error or warning.



Selecting an error in the list and clicking Open or double-clicking on an error opens the offending file in the preferred editor for that file. In the case of .j files,

XcodeCapp will attempt to open the file directly to the offending line in the following editors:

- Sublime Text
- TextMate
- TextWrangler
- BBEdit
- Chocolat
- MacVIM

The error list is cleared when a new project is opened. Also, whenever a file is modified or deleted, any existing errors for that file are removed. So the Errors & Warnings panel (should!) always represents the current state of your files.

Note: Errors are not tracked across file renames. Renaming a file will remove any errors originating from that file.

If you do not wish the Errors panel to open automatically when errors or warnings occur, you can change that behavior in the Preferences window. See "Preferences" below for more information.

Editing Xibs

When you open a project, XcodeCapp creates an Xcode 4 project with which you to edit your project's xibs. If the preference to auto-open Xcode projects is set, the Xcode project will be opened as soon as the Cappuccino project is finished loading.

To manually open the Xcode project, click on the XcodeCapp menu and select "Open Project in Xcode".

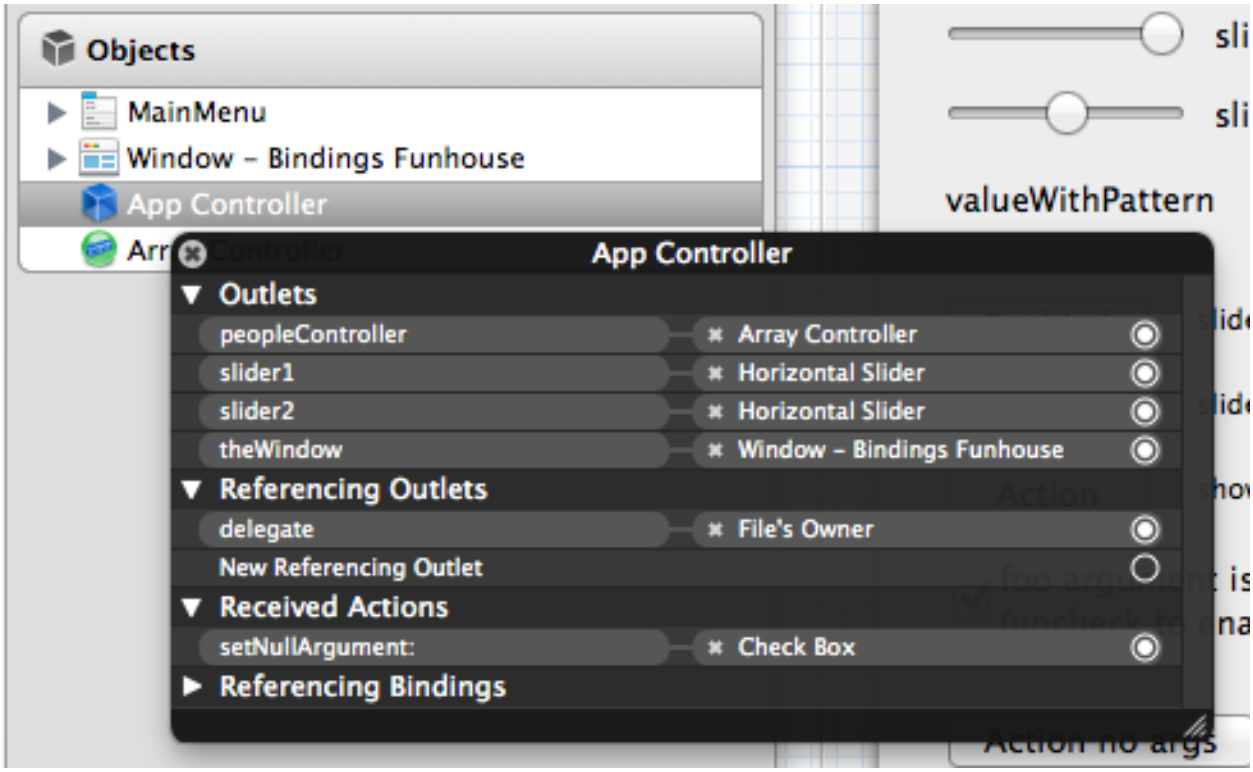
Once the Xcode project is open, you can edit your xibs with Xcode's interface builder. The Xcode project contains three folders by default:

- "Resources" – Contains the files in your project's Resources directory.
- "Cocoa Classes" – Contains the Objective-C classes generated by XcodeCapp. The filenames represent the project-relative path in which the Cappuccino source was found. You may open the .h files and connect

outlets and actions to controls in the interface builder. Do not edit or remove these files.

- "Cappuccino Source" – Contains the source files found by XcodeCapp. You may edit these files in Xcode if you wish.

All of the outlets and actions you declared in your source will be available in interface builder for connection to views. For example, the InfoPanelController shown above would have these outlets in interface builder:



To add more outlets or actions to the xib, you must edit the Cappuccino source and wait briefly while XcodeCapp processes the changes; do not attempt to add an outlet or action directly in Xcode.

Working with User Frameworks

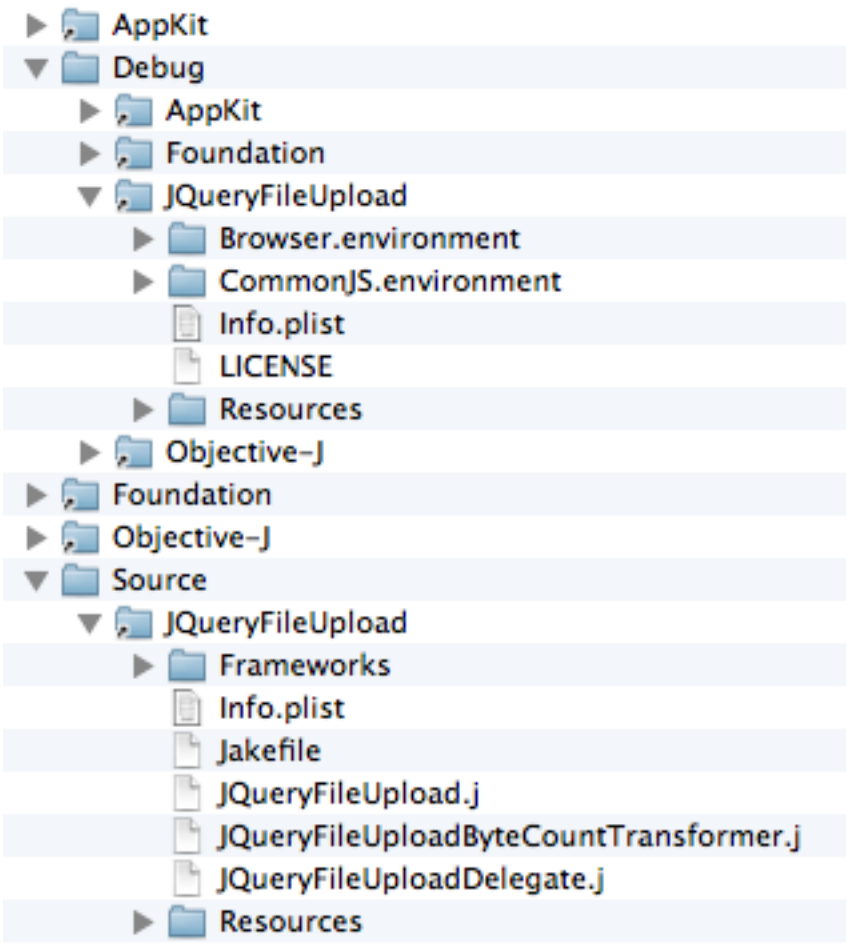
When a project is opened, XcodeCapp looks for user (non-Cappuccino) frameworks in your project's Frameworks/Debug directory. Any debug user frameworks in source form will be processed by XcodeCapp. This allows you to declare outlets and actions in frameworks and use them within Xcode. In addition, if a framework contains a Resources directory, a "<framework>

Resources" folder referencing that directory is created in the Xcode project. This allows you to reference framework resources within Xcode.

If you want your user frameworks to execute in compiled form, but you still want XcodeCapp to have access to the framework's source in order to parse outlets and actions, do the following:

1. Copy or symlink to the compiled framework as usual within your project's Frameworks directory.
2. Create a directory called "Source" within your project's Frameworks directory.
3. Within the Source directory, copy or symlink to the framework source directory.
4. XcodeCapp will recursively search the Frameworks/Source directory when a project is opened.

For example, here is the structure of a project's Frameworks directory:



Note the following:

- Debug/JQueryFileUpload is a symlink to the compiled framework. This is what will be executed when the app is run in debug mode.
- The Source directory contains a symlink to the JQueryFileUpload source. This will be parsed by XcodeCapp.

If you need to use image resources located in a framework — for example in an NSButton or NSImageView — you have two choices:

1. Use the base image name alone, for example "cancel", which you wish to represent the image "cancel.png" in the framework "JQueryFileUpload".
2. Use a full name@framework specification, for example "cancel@JQueryFileUpload" (without the quotes).

In the first form, during conversion, `nib2cib` will first search the app's Resources directory for an image with the given name, then it will search all of the Resources directories of the debug frameworks, followed by the release frameworks. The first match wins, and if the image was found in a framework, the framework's bundle identifier is recorded so the image can be loaded from the correct bundle at runtime.

Since there might be multiple images in the search path that have the same name, if you want to ensure you are using an image from a specific framework, use the second form. During conversion, `nib2cib` will check the named framework for the given image, and if it is found the framework's bundle identifier is recorded so the image can be loaded from the correct bundle at runtime.

Ignoring Files and Folders

There may be source which you do not want XcodeCapp to process. To specify directories or files which you want XcodeCapp to ignore, do the following:

1. Create a text file in the root of the project directory named ".xcodecapp-ignore".
2. Enter one line for each pattern you would like to ignore, where a pattern matches the *absolute* (full) path to the file or directory.

3. Save the file. Whenever .xcodecapp-ignore is modified, XcodeCapp will parse it to rebuild the ignore list.

Patterns may use "*" as a wildcard to match zero or more characters (it is *not* a shell glob) and are matched against the *absolute* path of the file or directory, so in most cases your patterns should begin with "*". To ignore a directory, add "/" at the end of the directory name. For example, "*/server/" will ignore that directory and any files within it.

For example, to ignore the "Modules" directory and the xib file "foo.xib" in your project, enter these lines in .xcodecapp-ignore:

```
*/Modules/
*/foo.xib
```

Sometimes you may want to ignore almost all of the files or directories within a given directory. In such cases you can use an include pattern. An include pattern has the same syntax as the normal exclude patterns, but begins with "!". Since ignore patterns are parsed in the order they appear in .xcodecapp-ignore, an include pattern should come after the exclude pattern it overrides. For example, if you wanted to exclude everything in the Modules directory, except for the Classes directory and any xib files, you would use the following patterns:

```
*/Modules/
!*/Modules/Classes/
!*/Modules/*.xib
```

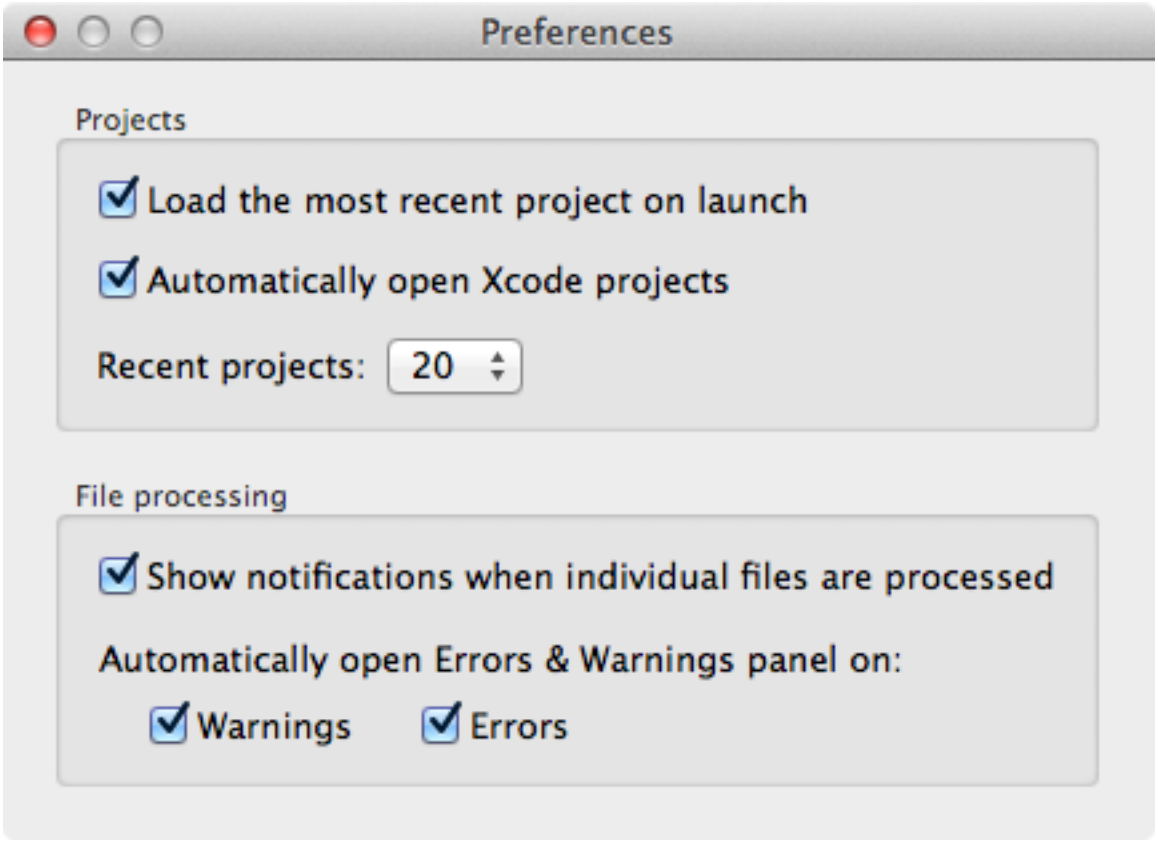
Note that in addition to whatever patterns you specify (if any) in .xcodecapp-ignore, XcodeCapp parses the following patterns *before* your patterns:

```
*/Frameworks/
!*/Frameworks/Debug/
*/AppKit/
*/Foundation/
*/Objective-J/
*/*.environment/
*/Build/
*/.*/
*/NS_*.j
*/main.j
```

```
*/.*
!*/.xcodecapp-ignore
```

Preferences

XcodeCapp offers the following preferences:



Currently no warnings are generated by XcodeCapp. But in the future if there are warnings which do not prevent the rest of the file from being parsed, you may choose not to display the Errors panel when such warnings are generated.

Troubleshooting

XcodeCapp tries its best to keep your project source and the Xcode project synchronized, and handles most file changes with no problem. However, it is possible for the Xcode project get out of sync with your source files. In such cases, you can synchronize the Xcode project with your source by selecting "Synchronize Project" from the XcodeCapp menu. This will remove and then regenerate the Xcode project and .XcodeSupport folder. If the Xcode project is open in Xcode it will be closed before being removed.

Credits

XcodeCapp was originally written by Francisco Tolmasky.

The Cocoa port was originally written by Antoine Mercadal for the Archipel Project (archipelproject.org), and was then made part of Cappuccino.

XcodeCapp 3 is a substantial rewrite by Aparajita Fishman.