

[BlackDog Foundry](#)

- [RSS](#)



Enter search term

[BlackDog Foundry](#)

- [LogDiver](#)
- [About](#)

[Bookmark This page](#)

Common Xcode4 Plugin Techniques »

craig
23 Oct
[Tutorials](#)
[objective-c](#), [plugin](#), [xcode4](#)
[2 Comments](#)

Common Xcode4 Plugin Techniques

This tutorial explores the next steps for writing your own Xcode4 plugin. An [earlier article](#) describes how to create a project that successfully builds a plugin; this article describes a few techniques that you can use.

This article builds upon the original project that can be found on [GitHub](#).

Word of Warning

As you would know, Apple does not formally support plugins and the underlying Xcode classes may change at any time. It is very important to write your plugin code to be a) resilient to the presence (or not) of various classes, and b) robust so that your plugin doesn't crash Xcode.

Often, you will be listening for `NSNotification` events and inspecting view hierarchies to find the objects you need. Your code needs to gracefully degrade to handle the cases where, say, notifications don't arrive or the view hierarchy changes.

You'll often need to [swizzle](#) methods so that you can hook into the existing Xcode behaviour.

You will also likely have to refer to private classes, but because there are no public headers, you will have to use the `id` type. Study up on how to use reflection effectively.

Creating an Instance

Many callback methods require an instance of a class to call back to, so a very common technique is to create a singleton instance of your plugin using code in your plugin like the following:

```
static BDXcodePluginDemo *mySharedPlugin = nil;

+ (void) pluginDidLoad:(NSBundle *)plugin {
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        mySharedPlugin = [[self alloc] init];
    });
}

+ (BDXcodePluginDemo *)sharedPlugin {
    return mySharedPlugin;
}
```

Checking Available Events

So, now that you have a newly created instance of your plugin, what can we do with it?

Let's see what sort of notifications are getting posted by Xcode. Add the following code:

```
-(id)init {
    if (self = [super init]) {
        [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(notificationListener:) name:nil object:nil];
    }
    return self;
}

-(void)notificationListener:(NSNotification *)notification {
    // let's filter all the "normal" NSxxx events so that we only
    // really see the Xcode specific events.
    if ([[notification name] length] >= 2 && [[[notification name] substringWithRange:NSMakeRange(0, 2)] isEqualToString:@"NS"])
        return;
    else
        NSLog(@" Notification: %@", [notification name]);
}

-(void)dealloc {
    [[NSNotificationCenter defaultCenter] removeObserver:self];
    [super dealloc];
}
```

Build the project, close Xcode and tail the system log using:

```
tail -f /var/log/system.log
```

And then just start doing some stuff like opening files, projects, compiling, etc. You'll see all sorts of interesting events being output:

```
Oct 22 18:31:54 edwardaux Xcode[34888]: Notification: PBXUnarchiverDidFinishUnarchivingNotification
Oct 22 18:31:54 edwardaux Xcode[34888]: Notification: PBXProjectDidOpenNotification
Oct 22 18:31:54 edwardaux Xcode[34888]: Notification: XCPPropertyInfoContext_DictionaryWasAdded
Oct 22 18:31:54 edwardaux Xcode[34888]: Notification: IDEIntegrityLogDataSourceDidChangeNotification
Oct 22 18:31:54 edwardaux Xcode[34888]: Notification: IDEContainerDidOpenContainerNotification
Oct 22 18:31:54 edwardaux Xcode[34888]: Notification: IDEIntegrityLogDataSourceDidChangeNotification
Oct 22 18:31:54 edwardaux Xcode[34888]: Notification: IDEContainerDidOpenContainerNotification
Oct 22 18:31:54 edwardaux Xcode[34888]: Notification: IDEControlGroupDidChangeNotificationName
Oct 22 18:31:55 edwardaux Xcode[34888]: Notification: transition from one file to another
Oct 22 18:31:55 edwardaux Xcode[34888]: Notification: IDEEditorAreaLastActiveEditorContextDidChangeNotification
```

Obviously, in your real plugin, you wouldn't be listening for all notifications, but this technique gives you a pretty good view of the types of notifications that you can get your plugin to react to.

Adding a Menu Item

Depending on what you are doing, you may need to add a new menu into the main menu bar (or a new item into an existing menu item). Some example code is shown below to:

- Add a new menu item to the Edit menu called *Click me 1*
- Create a new menu called *Demo* with a single menu item called *Click me 2*

Note the code below is a little fragile for localised installations of Xcode. For example, in a non-English locale, the menu titles may not be what you expect. If you know the existing menu index number, you may be better to use `[mainMenu itemAtIndex:2]`.

```
-(void)addMenuItems {
    NSMenu *mainMenu = [NSApp mainMenu];

    // find the Edit menu and add a new item
    NSMenuItem *editMenu = [mainMenu itemWithTitle:@"Edit"];
    NSMenuItem *item1 = [[NSMenuItem alloc] initWithTitle:@"Click me 1" action:@selector(click1:) keyEquivalent:@""];
    [item1 setTarget:self];
    [[editMenu submenu] addItem:item1];

    // create a new menu and add a new item
    NSMenu *demoMenu = [[NSMenu alloc] initWithTitle:@"Demo"];
    NSMenuItem *item2 = [[NSMenuItem alloc] initWithTitle:@"Click me 2" action:@selector(click2:) keyEquivalent:@""];
    [item2 setTarget:self];
    [demoMenu addItem:item2];
    // add the newly created menu to the main menu bar
    NSMenuItem *newMenuItem = [[NSMenuItem alloc] initWithTitle:@"Demo" action:NULL keyEquivalent:@""];
    [newMenuItem setSubmenu:demoMenu];
    [mainMenu addItem:newMenuItem];
}

-(void)click1:(id)sender {
    NSLog(@"Menu item 1 clicked");
}

-(void)click2:(id)sender {
    NSLog(@"Menu item 2 clicked");
}
```

Don't forget to add a call to `addMenuItems` in your `init` function:

```
-(id)init {
    if (self = [super init]) {
        ...
        [self addMenuItems];
        ...
    }
    return self;
}
```

Finding a Control

So, let's say that you want to write a plugin that manipulates some control within your Xcode environment. How do you get a reference to that object? Well firstly, you need to see what is available.

One way is to get a reference to the root window and walk the window hierarchy dumping controls as you go. A very simple implementation is to create a category on `NSView` like so:

```
@implementation NSView (Dumping)

-(void)dumpWithIndent:(NSString *)indent {
    NSString *clazz = NSStringFromClass([self class]);
    NSString *info = @"";
    if ([self respondsToSelector:@selector(title)]) {
        NSString *title = [self performSelector:@selector(title)];
        if (title != nil && [title length] > 0)
            info = [info stringByAppendingFormat:@"% title=%@", title];
    }
    if ([self respondsToSelector:@selector(stringValue)]) {
        NSString *string = [self performSelector:@selector(stringValue)];
        if (string != nil && [string length] > 0)
            info = [info stringByAppendingFormat:@"% stringValue=%@", string];
    }
    NSString *tooltip = [self tooltip];
    if (tooltip != nil && [tooltip length] > 0)
        info = [info stringByAppendingFormat:@"% tooltip=%@", tooltip];

    NSLog(@"%s", indent, clazz, info);
}
```

```

    if ([[self subviews] count] > 0) {
        NSString *subIndent = [NSString stringWithFormat:@"%%%%", indent, ([indent length]/2)%2==0 ? @"| " : @" : "];
        for (NSView *subview in [self subviews])
            [subview dumpWithIndent:subIndent];
    }
}

@end

@implementation BDXXcodePluginDemo
...
-(void)dumpWindow {
    [[NSApp mainWindow] contentView] dumpWithIndent:@" ";
}
...
@end

```

It outputs a nicely formatted tree like:

```

Oct 23 14:17:12 edwardaux Xcode[40551]: NSView
Oct 23 14:17:12 edwardaux Xcode[40551]: | DVTTabSwitcher
Oct 23 14:17:12 edwardaux Xcode[40551]: : NSTabView
Oct 23 14:17:12 edwardaux Xcode[40551]: : | DVTControllerContentView
Oct 23 14:17:12 edwardaux Xcode[40551]: : : DVTSplitView
Oct 23 14:17:12 edwardaux Xcode[40551]: : : | DVTReplacementView
Oct 23 14:17:12 edwardaux Xcode[40551]: : : : DVTControllerContentView
Oct 23 14:17:12 edwardaux Xcode[40551]: : : : | NSView
Oct 23 14:17:12 edwardaux Xcode[40551]: : : : : DVTBorderedView
Oct 23 14:17:12 edwardaux Xcode[40551]: : : : : | DVTReplacementView
Oct 23 14:17:12 edwardaux Xcode[40551]: : : : : : DVTControllerContentView
Oct 23 14:17:12 edwardaux Xcode[40551]: : : : : : | NSScrollView
Oct 23 14:17:12 edwardaux Xcode[40551]: : : : : : : NSClipView
Oct 23 14:17:12 edwardaux Xcode[40551]: : : : : : | IDENavigatorOutlineView
Oct 23 14:17:12 edwardaux Xcode[40551]: : : : : : : | NSScroller
Oct 23 14:17:12 edwardaux Xcode[40551]: : : : : : : DVTChooserView
Oct 23 14:17:12 edwardaux Xcode[40551]: : : : : : : | NSMatrix stringValue=1
Oct 23 14:17:12 edwardaux Xcode[40551]: : : : : : : DVTBorderedView
Oct 23 14:17:12 edwardaux Xcode[40551]: : : : : : : | IDENavigatorFilterControlBar
Oct 23 14:17:12 edwardaux Xcode[40551]: : : : : : : : DVTSearchField tooltip=Show files with matching name
Oct 23 14:17:12 edwardaux Xcode[40551]: : : : : : : : DVTRollerImageButton title=Button stringValue=0 tooltip=Show only recent files
...

```

Note that I found the neat tree drawing in an [article](#) on StackOverflow – thanks to user [proqmr](#).

What if Something Goes Wrong?

If your plugin starts crashing Xcode, or prevents it from starting up, the fix is pretty simple. Just delete your plugin from the plugin directory using the following command:

```
rm -rf /Users/edwardsc/Library/Application\ Support/Developer/Shared/Xcode/Plug-ins/BDXXcodeDemoPlugin
```

And then restart Xcode. *Take note of the embedded space in `Application Support` in that directory path though.*

Publicly Available Plugin Source

There are a number of plugins that have the source available and may be useful as a source of inspiration (in no particular order):

- [Dash-Plugin-for-Xcode](#) – Uses Dash as the documentation viewer when option-clicking.
- [ColorSense-for-Xcode](#) – Displays a colour picker to easily view and select colours when working with `NSColor` and `UIColor`.
- [MiniXcode](#) – Shows and hides the Xcode main toolbar.
- [XVim](#) – A vim plugin for Xcode.
- [Xcode4 Fixes](#) – A collection of improvements to the base Xcode behaviour.


Closing

If you spot any inconsistencies or errors in the above, please [let me know](#). Additionally, if there is any specific topics that you would like covered, feel free to drop me an email.

All articles in this series

- [Creating an Xcode4 plugin](#)
- [Common Plugin Techniques](#)
- [Introducing XcodeExplorer](#)
- [Checking out the Xcode controls](#)

2 Comments »

1.  OpenFibers says:
[April 16, 2013 at 5:32 am](#)

Great tutorial!

2.  alextd says:
[May 7, 2013 at 6:46 pm](#)

Thanks for xCode plugins, tutorial.

Leave a Comment »

Name (required)

Mail (will not be published)(required)

Website

Submit

Categories

- [Tips](#) (7)
- [Tutorials](#) (9)