



Published on [MacDevCenter](http://www.macdevcenter.com/) (<http://www.macdevcenter.com/>)

[See this](#) if you're having trouble printing code examples

# Programming with Spotlight

by [Matthew Russell](#)

07/12/2005

The API for Spotlight offers highly advanced search capabilities. In fact, you can develop some of the very features of Tiger we've already grown to love using Spotlight's API. In this piece, we'll ease into Spotlight programming from a Cocoa development perspective so that you can make your applications Spotlight enabled. Next time (this coming Friday), we'll finish our work with Spotlight by hacking up a plugin for Stickies.

## Pregame

As with any endeavor, an adequate background is vital to success. Apple has provided quite a bit of documentation on Spotlight, and it is very good, although not quite final. This writing assumes that you've done a bit of Cocoa programming before and understand how Spotlight works at a conceptual level. If you need a quick crash course, review:

- [Introduction to Cocoa Design Patterns Guide](#) (especially the Model View Controller paradigm)
- [Working with Spotlight](#) (including the "For More Information" links at the bottom)
- [Introduction to Carbon-Cocoa Integration Guide](#) (especially "Toll-Free Bridging," briefly)

We'll ease into the Cocoa programming initially, but will quickly increase the pace because there's a lot of turf to cover with Spotlight specifics. If you need more context on general purpose Cocoa programming than Apple's developer documentation, check out one of the many excellent tutorials here on MacDevCenter. Without further adieu, sit back, relax, and strap on your seat belt.

## Developing with Spotlight

As a developer, you can interact with Spotlight in a variety of ways. Here are a few of the most common ways:

- Use Carbon- or Cocoa-level function calls from within your compiled application
  - To have your application display the Spotlight search window
  - To directly examine the metadata of a specific file
  - To query the Spotlight server for specific metadata constraints on an operating system-wide level

### Related Reading

- Use a command line tool from within a script
  - Parse the output of an existing metadata tool such as `mdls` or `mdfind`
  - Create your own customized command line tool that performs a task of your own choosing
- Create a plugin
  - Allow Spotlight to use the metadata available from your own application's custom file types
  - Create a plugin for an existing file type for which there's not any available plug-in

We'll work through each of these possibilities, and you'll soon be able to interact with Spotlight on a variety of levels. Let's get our hands dirty with some code by building a sample application, looking at some of Apple's examples, and reviewing some of the command line tools.

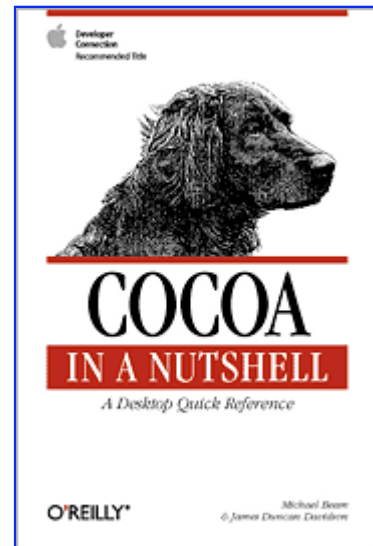
## Displaying the Spotlight Search Window

Since Cocoa is the drink of the day (as always), let's make an example project that interacts with Spotlight. If you haven't already, now is a good time to update to [Xcode 2.1](#). In Xcode:

- Create a new project
  - Open Xcode
  - Create a new "Cocoa Application"
  - Name it "SpotlightExamples" and save it somewhere

With a template in place, let's proceed to create a controller class since we're big fans of the Model View Controller paradigm

- Create a controller class
  - From Xcode's "File" menu, choose "New File"
  - Pick "Objective-C class"
  - Name it "Controller" and choose to also create the header
  - In Xcode's "Groups & Files" pane, drag the Controller files into the "Classes" folder



### [Cocoa in a Nutshell](#)

#### **A Desktop Quick Reference**

By [Michael Beam](#), [James Duncan Davidson](#)

#### [Table of Contents](#)

#### [Index](#)

#### [Sample Chapter](#)

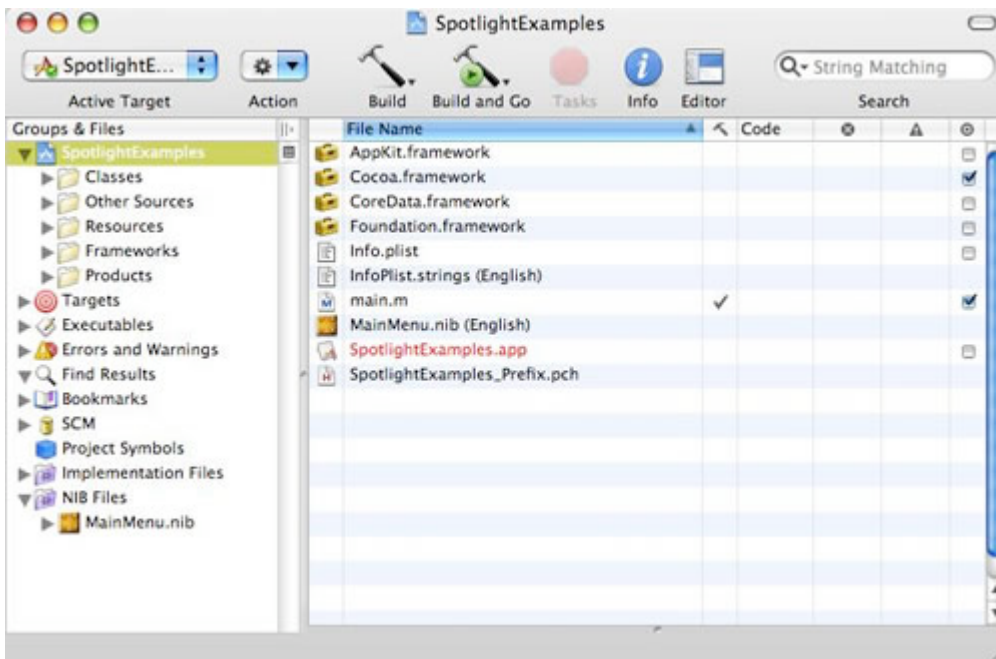
#### [Read Online--Safari](#)

Search this book on Safari:

Go

Only This Book

☐ Code Fragments only



*Create a new project in Xcode*

Our controller class needs outlets and connections, so let's add those. Replace your controller files with the following ones. (We're simply adding an outlet that'll correspond to a button and an action that contains a Carbon-level call to open up Spotlight's search window.)

For "Controller.h":

```
// Controller.h
// SpotlightExamples

#import <Cocoa/Cocoa.h>

@interface Controller : NSObject {
    IBOutlet NSButton* openSearchWindowButton;
}

- (IBAction)openSearchWindowAction:(id)sender;
@end
```

For "Controller.m":

```
// Controller.m
// SpotlightExamples

#import "Controller.h"

@implementation Controller

- (IBAction)openSearchWindowAction:(id)sender
{
    OSStatus resultCode=noErr;

    //Replace "Search Text" with user input
    resultCode=
        HISearchWindowShow((CFStringRef)@"Search Text", kNilOptions);
}
```

```

    if (resultCode != noErr) {
        NSLog(@"Failed to open search window");
        //Could use NSAlert class to display interactive dialog
    }
}
@end

```

Let's go ahead and instantiate the controller.

- Instantiate the Controller
  - Open up the main menu of your application by expanding the "NIB Files" folder and double clicking on "MainMenu.nib"
  - Drag the "Controller.h" file in Xcode down onto Interface Builder's "MainMenu.nib" panel.
  - In the "Classes" tab of Interface Builder's "MainMenu.nib" panel, select "NSObject" and then "Controller"
  - From Interface Builder's "Classes" menu, choose "Instantiate Controller"



*Drag and drop your "Controller.h" file onto Interface Builder's main palette and then instantiate it.*

You should now see your controller under the "Instances" tab of Interface Builder as a blue cube. The little yellow exclamation point reminds us that at least one outlet is not set, so let's take care of that.

- Add a button and make the connections
  - From Interface Builder's "Cocoa-Controls" palette, drag an NSButton onto your application's main menu window. Double click and rename it "Open Search Window"
  - Ctrl-click and drag from the blue controller cube onto the button; release and choose "Connect" for the "openSearchWindowButton" outlet.
  - Ctrl-click and drag from the button onto the blue controller cube; release, and choose "Connect" for the "openSearchWindowAction" action.

Now you can run the application, click on the button, and get the search window to appear. As you can see, there's just a simple Carbon-level call to make this happen. To make this feature useful, your application might offer drag-and-drop functionality to allow a user to drag over a file for some sort of processing.



*Your initial project can display the Spotlight search window.*

You can get the project file for this first portion [here](#).

## Examine a Specific File's Metadata

Our next topic investigates Spotlight's ability to examine a specific file's metadata, and it's surprisingly easy. To illustrate, let's enhance our example project. For the sake of time, we'll stay focused and add a text view object that displays the metadata for a specific file. In your own applications, you'd probably be doing something more involved. You can find plenty of good Cocoa tutorials that illustrate broader topics involving user interaction, file browsers, etc. here on MacDevCenter if you need that additional context.

Modify your controller files (changes are in bold):

For "Controller.h":

```
// Controller.h
// SpotlightExamples

#import <Cocoa/Cocoa.h>

@interface Controller : NSObject {
    IBOutlet NSButton* openSearchWindowButton;

    IBOutlet NSTextView* metadataInfoView;
    IBOutlet NSButton* displayMetadataButton;
}

- (IBAction)openSearchWindowAction:(id)sender;

- (IBAction)displayMetadataAction:(id)sender;

@end
```

For "Controller.m":

```
// Controller.m
// SpotlightExamples

#import "Controller.h"

@implementation Controller
```

```

- (IBAction)openSearchWindowAction:(id)sender
{
    OSStatus resultCode=noErr;

    //Replace "Search Text" with user input
    resultCode=HISearchWindowShow((CFStringRef)@"Search Text", kNilOptions);
    if (resultCode != noErr) {
        NSLog(@"Failed to open search window");
        //Could use UIAlertView class to display interactive dialog
    }
}

- (IBAction)displayMetadataAction:(id)sender
{
    //create a CF-compliant object representing a file and its metadata using
    //a Carbon level call

    //add in a path to an existing file on your system
    CFStringRef path = CFSTR("/Users/matthew/temp.txt");
    MDItemRef item = MDItemCreate(kCFAllocatorDefault, path);

    //pull out the metadata attribute names
    CFArrayRef attributeNames = MDItemCopyAttributeNames(item);

    //use toll-free bridging to load up an NSArray for convenience
    NSArray* array = (NSArray*)attributeNames;
    NSEnumerator *e = [array objectEnumerator];
    id arrayObject;

    //placeholders
    NSMutableString *info = [NSMutableString stringWithCapacity:50];
    CFTypeRef ref;

    while ((arrayObject = [e nextObject]))
    {
        ref =
        MDItemCopyAttribute(item, (CFStringRef)[arrayObject description]);

        //cast to get an NSObject for convenience
        NSObject* tempObject = (NSObject*)ref;

        [info appendString:[arrayObject description]];
        [info appendString:@" = "];
        [info appendString:[tempObject description]];
        [info appendString:@"\n"];
    }

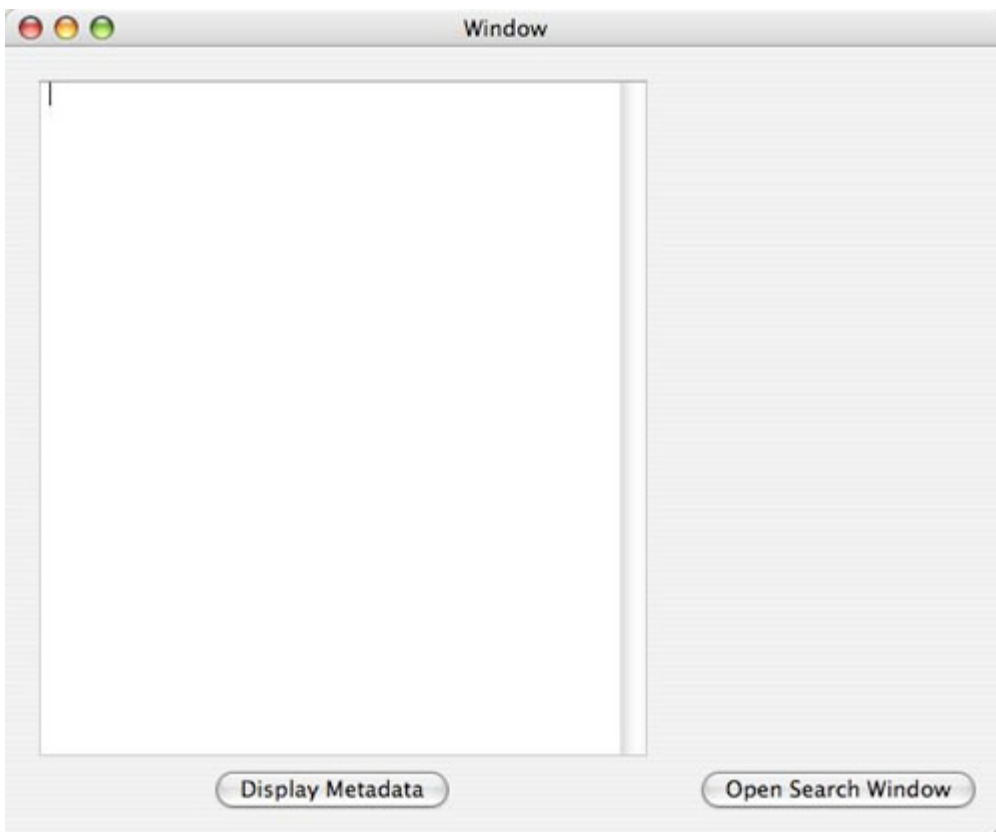
    //set the info in the text view
    [metadataInfoView insertText:info];
}
@end

```

*You'll need to update your instantiated controller to reflect changes in your source code by dragging the "Controller.h" file onto the main Interface Builder palette and choose to "Replace" when prompted. Without this change, the controller won't recognize the new outlets and action you just added.*

- Spruce up your application's main window in Interface Builder
  - Add an NSButton and rename it "Display Metadata"
  - Add an NSTextView (from the "Cocoa-Text" tab)
  - Set the outlet and action for the new button
  - Set the outlet for the text view the same as with the button (it doesn't have any actions)

With all that done, ensure that you've specified a file that exists on your system in method `openSearchWindowAction:`. I just created a temporary text file for purposes of illustration. Again, your application would be doing all sorts of fancy user interaction here; we're being simple on purpose. "Build and Go" to see the action. If something's not working, double check your outlets and connections. You can always add `NSLog` messages to help troubleshoot.



*Your project is now capable of examining a file's metadata.*

You can get the project file for this second portion [here](#).

## Query the Spotlight Server

So far, we've displayed the Spotlight search window and examined a specific file's metadata. While interesting and useful, these features still leave plenty to be sought. Querying the Spotlight server helps to fill this void and is one of the ways Spotlight really struts its stuff. In our example application, we'll use Spotlight to query the entire file system for mail messages the same way we could in the Spotlight search window. Take a moment to review the docs on [NSMetadataQuery](#) and [NSPredicate](#) if you haven't already; these are the cornerstones. Everything else is from the standard Model View Controller repertoire.

Let's update your controller class again. Changes are in bold.

For "Controller.h"

```
// Controller.h
// SpotlightExamples

#import <Cocoa/Cocoa.h>

@interface Controller : NSObject {
    IBOutlet NSButton* openSearchWindowButton;

    IBOutlet NSTextView* metadataInfoView;
    IBOutlet NSButton* displayMetadataButton;

    IBOutlet NSButton* startSearchButton;
    IBOutlet NSButton* stopSearchButton;
    IBOutlet NSTextField* numHitsField;
    NSMetadataQuery* q;
    NSTimer* t;
}

- (IBAction)openSearchWindowAction:(id)sender;

- (IBAction)displayMetadataAction:(id)sender;

- (id)init;
- (void)awakeFromNib;
- (void)dealloc;
- (IBAction)startSearchAction:(id)sender;
- (IBAction)stopSearchAction:(id)sender;
- (void)stopSearching;
- (void)updateResults:(NSTimer*)timer;

@end
```

For "Controller.m"

```
// Controller.m
// SpotlightExamples

#import "Controller.h"

@implementation Controller
- (IBAction)openSearchWindowAction:(id)sender
{
    OSStatus resultCode=noErr;

    //Replace "Search Text" with user input
    resultCode=
        HISearchWindowShow((CFStringRef)@"Search Text", kNilOptions);
    if (resultCode != noErr) {
        NSLog(@"Failed to open search window");
        //Could use UIAlertView class to display interactive dialog
    }
}

- (IBAction)displayMetadataAction:(id)sender
```



```

{

    //create a CF-compliant object representing
    //a file and its metadata using a Carbon level call

    //add in a path to an existing file on your system
    CFStringRef path = CFSTR("/Users/matthew/temp.txt");
    MDItemRef item = MDItemCreate(kCFAllocatorDefault, path);

    //pull out the metadata attribute names
    CFArrayRef attributeNames = MDItemCopyAttributeNames(item);

    //use toll-free bridging to load up an NSArray for convenience
    NSArray* array = (NSArray*)attributeNames;
    NSEnumerator *e = [array objectEnumerator];
    id arrayObject;

    //placeholders
    NSMutableString *info =
        [NSMutableString stringWithCapacity:50];

    CTypeRef ref;

    while ((arrayObject = [e nextObject]))
    {
        ref =
            MDItemCopyAttribute(item, (CFStringRef)[arrayObject description]);

        //cast to get an NSObject for convenience
        NSObject* tempObject = (NSObject*)ref;

        [info appendString:[arrayObject description]];
        [info appendString:@" = "];
        [info appendString:[tempObject description]];
        [info appendString:@"\n"];
    }

    //set the info in the text view
    [metadataInfoView insertText:info];
}

- (id)init
{
    if (self = [super init])
    {
        //release in dealloc
        q = [[NSMetadataQuery alloc] init];
    }
    return self;
}

- (void)awakeFromNib
{
    [q setDelegate: self];

    //remember to unregister for notifications
    [[NSNotificationCenter defaultCenter]
        addObserver:self
        selector:@selector(stopSearching)
        name:NSMetadataQueryDidFinishGatheringNotification
        object:nil];
}

```

```

- (void)dealloc
{
    [q release];

    [[NSNotificationCenter defaultCenter]
     removeObserver:self
     name:NSMetadataQueryDidFinishGatheringNotification
     object:nil];

    [super dealloc];
}

- (IBAction)startSearchAction:(id)sender
{
    //whatever query you want. emlX corresponds
    //to mail messages. you could easily
    //configure search terms from user interaction.
    NSPredicate *p =
        [NSPredicate predicateWithFormat:@"kMDItemKind == 'emlX'", nil];
    [q setPredicate:p];

    //optionally set search scopes
    //[q setSearchScopes:
    //    [NSArray arrayWithObject:@"/Users/matthew/Library/Mail/"]];

    //start the query and use the run loop
    //to process the search progress.
    if ([q startQuery])
    {
        t =
            [NSTimer scheduledTimerWithTimeInterval:0.25
             target:self
             selector:@selector(updateResults:)
             userInfo:q
             repeats:YES];

        //NSRunLoop retains the timer
        [[NSRunLoop currentRunLoop]
         addTimer:t
         forMode:NSDefaultRunLoopMode];
    }
    else
    {
        NSLog(@"Error. Could not start query. Weird.");
    }
}

- (IBAction)stopSearchAction:(id)sender
{
    [self stopSearching];
}

//called via the NSMetadataQueryDidFinishGatheringNotification
//and/or the stopSearchAction: method
- (void)stopSearching
{
    //don't invalidate a timer more than once
    if (![q isStopped])
    {
        //NSLog(@"Finito. Num results = %d", [q resultCount]);
        [self updateResults:t];
        [q stopQuery];
        [t invalidate];
    }
}

```

```

    }
}

- (void)updateResults:(NSTimer*)timer
{
    NSString *tempString =
    [NSString stringWithFormat:@"%d", [[timer userInfo] resultCount], nil];

    [numHitsField setStringValue:tempString];
    //NSLog(@"%d", [[timer userInfo] resultCount]);
}

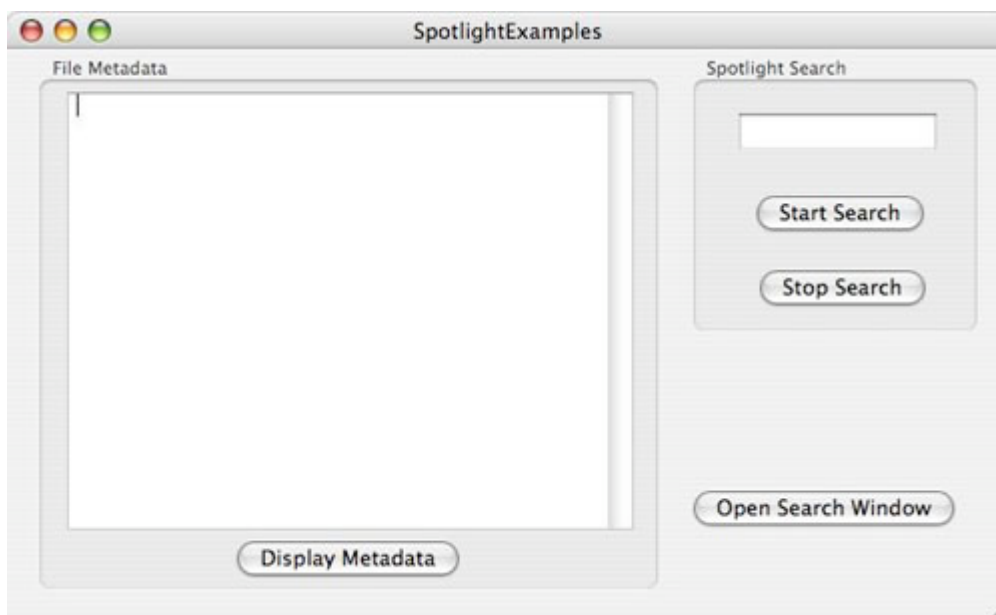
@end

```

Like last time, drag and drop your header file onto Interface Builder's palette so that your instantiated controller reflects the changes. On your application's main menu, you'll need to:

- Add two NSButtons and an NSTextField and connect their outlets
  - Rename one NSButton "Start Search"
  - Rename one NSButton "Stop Search"
  - Add the NSTextField
  - Connect their outlets as you've been previously doing with Ctrl-click drags.
- Set the actions for the "Start Search" button and "Stop Search" button
  - Connect "Start Search" to `startSearchAction:`
  - Connect "Stop Search" to `stopSearchAction:`
- Do any finishing touches (optional)
  - Group controls together with an NSBox
  - Title the main window

If you decide to group controls using an NSBox, you have to delete your existing controls, drag "fresh" controls from Interface Builder's palette over onto the box, and then re-establish the outlets and connections. This only takes a few moments, and makes things look a lot less chaotic. Although the application you've developed is fairly pedagogical, the concepts and code snippets used are the same that you'd use in more sophisticated circumstances.



*Your project can now query the entire filesystem.*

You can get the project file for this final portion [here](#).

## Command Line Tools

Before creating a Spotlight plugin next time, you might like to know that you don't necessarily have to be a *Cocoa* programmer to benefit from Spotlight. Apple provides several very useful command line tools that you can use in shell scripts to query and manipulate metadata. Here's a few you might find handy:

- `mdls`: lists the metadata attributes for the specified file
- `mdfind`: finds files matching a given query
- `mdimport`: imports file hierarchies into the metadata datastore
- `mdutil`: manages the metadata stores used by Spotlight

Since this is a Cocoa-oriented tutorial, we won't work through command-line examples. You'll have no problems getting acquainted by using the man pages or Apple's documentation. These command-line tools are very useful for debugging an importer (as we'll see next time) or for use in your Perl, Bash, or other scripting routines that can benefit from the metadata awareness.

On a final note about command line tools, realize that you aren't constrained to the ones Apple provides. You can create your own custom tools using Spotlight's Carbon-level API available to you. In fact, the Cocoa-level API we've primarily been using is simply a wrapper around these Carbon-level calls. The concepts are exactly the same, and you might even enjoy the simplicity gained from the absence of human interaction (every once in a while).

## Next Time

For next time, you'll want to skim the documentation on [Introduction to Spotlight Importers](#) and ponder how in the world we'll get Spotlight to be aware of notes in Stickies. Until then.

*[Matthew Russell](#) is a computer scientist from middle Tennessee; and serves Digital Reasoning Systems as the Director of Advanced Technology. Hacking and writing are two activities essential to his renaissance man regimen.*

---

Return to the [Mac DevCenter](#)

Copyright © 2009 O'Reilly Media, Inc.