Spotlight Query Programming Guide

Carbon > File Management

2006-03-08

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Figures and Tables

Introduction to Spotlight Query Programming Guide

Spotlight provides several application programming interfaces that allow your application to search for files based on metadata. The level of interaction your application requires with the search results will often dictate the API that you chose.

The simplest way to provide Spotlight support in your application is to use the Spotlight search window. Using this API an application can display the standard Spotlight search window, optionally providing a search string. The search results are presented directly to the user and are not available to the application. This is a good choice if your application isn't search oriented, but you want to allow users to search for contextual terms using Spotlight.

For applications that need to create queries and interact with the results there are two APIs available. The Spotlight metadata framework provides a low-level query API, `MDQuery`, that allows an application to search for files based on metadata values. `MDQuery` is completely configurable, allowing you to run synchronous and asynchronous queries and provides fine-grain control of the frequency of results batching.

The Cocoa frameworks's `NSMetadataQuery` class provides a high-level Objective-C interface to the `MDQuery` API. This class allows you to construct queries using a subset of the `NSPredicate` classes, and execute the queries asynchronously. `NSMetadataQuery` supports Cocoa bindings, allowing you to display the results without writing any significant amount of glue code. As well, `NSMetadataQuery` allows an application to specify the grouping of the results into multiple subcategories. `NSMetadataQuery` does not support synchronous queries and provides minimal update notifications as data is collected.

Who Should Read This Document

Spotlight is a fundamental feature of Mac OS X, and all developers should be familiar with its capabilities. Many applications, at a minimum, should offer users the ability to search for selected text using the Spotlight search window.

Organization of This Document

The following articles cover key concepts in understanding how Spotlight can be used to query metadata:

- "Querying Metadata With Spotlight" (page 9) provides a conceptual overview of searching for files using Spotlight.
- "Displaying the Spotlight Search Window" (page 13) describes how to present the standard Spotlight search window.
- "Query Expression Syntax" (page 15) provides a description of Spotlight's query language.

See Also

There are other technologies, not fully covered in this document, that are fundamental to integrating metadata into your applications. Refer to these documents for more details:

- *Spotlight Overview* covers the conceptual details surrounding Spotlight's metadata usage.
- *Spotlight Importer Programming Guide* describes the plug-ins that extract metadata from document files.
- *Spotlight Metadata Attributes Reference* describe the metadata attributes provided by Apple.

Querying Metadata With Spotlight

In order for your application to search Spotlight metadata, you must create a query using one of the available Spotlight APIs. The Core Services framework provides `MDQuery`, a Core Foundation–style opaque type, while the Foundation framework provides `NSMetadataQuery`, an Objective-C class. Regardless of the API you choose, the concepts are the same.

Defining a Search Expression

The first step in creating a query is defining a search expression that returns the desired results. If you are using `MDQuery` to execute the query, create your search expression as a `CFString` using the syntax described in "Query Expression Syntax" (page 15). If you are using `NSMetadataQuery`, create an `NSPredicate` object that defines the search expression.

Configuring the Query

Once you have defined your search expression, you configure the query itself. All of the following steps are optional.

You can specify an array of metadata attributes called the value list attributes. The query collects the values of these attributes into uniqued lists that can be used to summarize the results of the search.

How you specify the sort order of the results depends on which query API you are using. `MDQuery` allows you to provide an array of metadata attribute names that are used as the primary sort key, the secondary sort key, and so on. This is a simple value comparison. If you require a more advanced sorting capability, you can specify a sort comparator function as a callback. If you are using `NSMetadataQuery`, you specify the sort order of the results by providing an array of sort descriptors.

An application limits where search results are collected from by specifying a search scope. The search scope is provided to the query as an array of predefined location constants, URLs, and directory paths. The predefined location constants provide convenient values for restricting a query to the user's home directory, locally mounted volumes and the user's home directory, or remote mounted volumes.

Note: It is important to remember that, while file-system metadata is available on all volumes, other metadata attributes are not. CDs, DVDs, disk images and System directories are not indexed by Spotlight. A user can also explicitly exclude results from being returned for specified locations by using Spotlight preferences.

Running the Query

Once you have created and configured a query object, you can execute the query itself. When running, a query typically has two phases: an initial results gathering phase and a live-update phase.

During the initial results gathering phase, the existing Spotlight system store is searched for files that match the search expression. The query sends notifications as the results are returned in batches. The query sends the application a notification when the initial results gathering phase has completed.

If the query has been configured to provide live-updates, your application receives notifications as the results of the query change in response to files having been changed on disk.

Using the Returned Results

Before your application interacts with the returned results, it must first temporarily disable updates of the results. You can disable updates during the initial gathering phase of a search or during the live-update phase.

An application determines the number of results that have been returned, and then can access individual result items by their indexed position in the results. The result items are returned as an object instance that encapsulates the attributes for the file. By default `MDQuery` returns `MDItem` instances, and `NSMetadataQuery` returns `NSMetadataItem` instances. Your application can then retrieve the values of the metadata attributes from these objects.

You can also retrieve the value of a specific metadata attribute at an index in the results if the attribute was specified as a value list attribute, is a sorting key, or is used as a grouping attribute.

When your application has finished accessing the results, it should reenable updates.

Customizing the Returned Results

You can provide callback functions to override the results returned by a query.

You can implement a callback that allows you to examine the attribute name and value that is to be returned to the query and provide an alternate value. This is useful, for example, if your application needs to provide a localized representation of certain returned values.

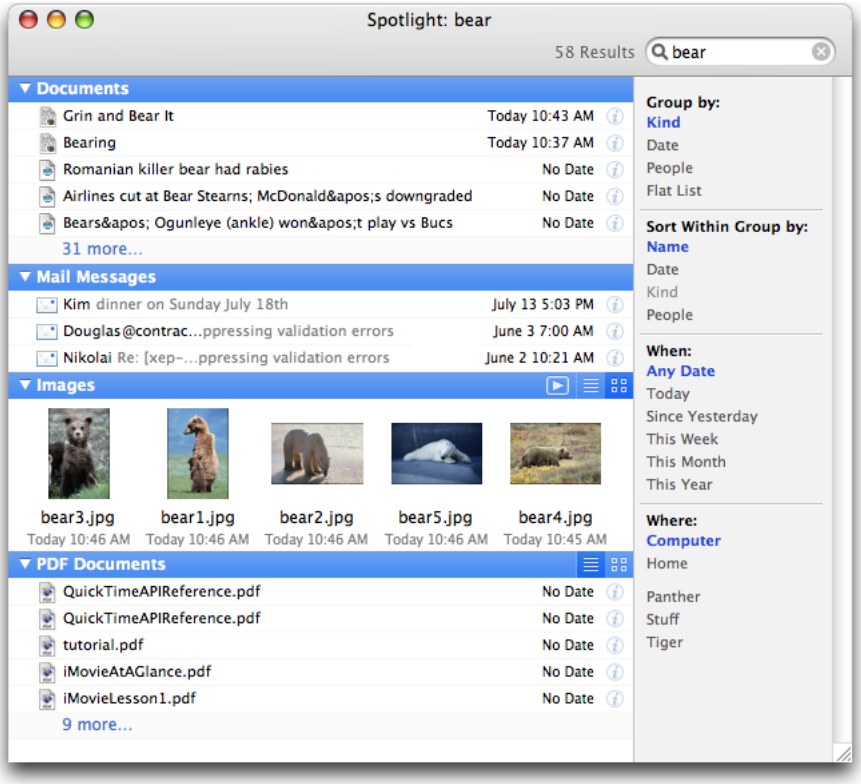
You can also provide a callback that returns an entirely different class of object to encapsulate the individual result items (rather than the default `MDQuery` or `NSMetadataItem` instances). This allows you to replace the generic instances with classes that implement custom functionality. These custom objects are returned when

your application asks the query for a result at a specific index. The objects returned by your callback are not used by the query mechanism, so there is no requirement that your custom objects implement any specific protocols.

Displaying the Spotlight Search Window

Applications can provide users direct interaction with Spotlight by displaying the standard Spotlight search interface, shown in Figure 1.

Figure 1 Spotlight search window



The Carbon `HISearchWindowShow` function provides a simple interface to the Spotlight search window. The following code fragment demonstrates extracting a string value from an `NSSearchField` and displaying the search interface.

```
OSStatus resultCode=noErr;

resultCode=HISearchWindowShow((CFStringRef)[sender stringValue],kNilOptions);

if (resultCode != noErr) {
    // failed to open the panel
    // present an error to the user
    }
```

The search window is presented using the default display settings that the user has configured in System Preferences.

Query Expression Syntax

Spotlight's query expresion syntax is a simplified form of filename globbing familiar to shell users. Queries have the following format:

```
attribute == value
```

where `attribute` is a Spotlight metadata attribute (see "*Spotlight Metadata Attributes Reference*") or a custom metadata attribute defined by an importer.

For example, to query Spotlight for all the files authored by "Steve" the query would look like the following:

```
kMDItemAuthors == "Steve"wc
```

The available comparison operators are listed in Table 1.

Table 1 Comparison operators

Operator	Description
==	equal
!=	not equal
<	less than (available for numeric values and dates only)
>	greater than (available for numeric values and dates only)
<=	less than or equal (available for numeric values and dates only)
>=	greater than or equal (available for numeric values and dates only)
inRange(attributeName,minValue,maxValue)	numeric values within the range of minValue through maxValue in the specified attributeName

Characters such as " and ‘ in the value string should be escaped using the \ character.

The search value in the example has the suffix "`wc`". These are modifiers that specify how the Comparison is made. Table 2 describes the available Comparison modifiers.

Table 2 Value Comparison modifiers

Modifier	Description
c	The Comparison is case insensitive.
d	The Comparison is insensitive to diacritical marks.

Modifier	Description
w	The Comparison is word based, and also detects transitions from lower-case to upper-case.

Table 3 shows several examples that use the Comparison modifiers.

Table 3 Value Comparison modifier examples

Search value	Results
"Paris"	Matches “Paris” but not “paris” nor “I love Paris”.
"Paris"c	Matches “Paris”, “paris”, but not “I love Paris”.
"Paris"wc	Matches “Paris”, “paris”, “I love Paris”, “paris-france.jpg”, but not “Comparison”.
"Window"w	Matches “MyWindowClass” and “Broken Window”, but not “NSWindow”.
"Frédéric"	Matches “Frédéric” but not “Frederic”.
"Frédéric"cd	Matches “Frédéric” and “Frederic” regardless of the word case.

Using the wildcard character (*) you can match substrings at the beginning of a string, end of a string, or anywhere within the string. Table 4 shows several common usages.

Table 4 Using wildcards

Value	Result
"paris*"	Matches attribute values that begin with “paris”. For example, matches “paris” and “parisol”, but not “comparison”.
"*paris"	Matches attribute values that end with “paris”.
"*paris*"	Matches attributes that contain "paris" anywhere within the value. For example, matches “paris”, “parisol” and “Comparison”.
"paris"	Matches attribute values that are exactly equal to “paris”.

Note: Typically, queries should use the “w” modifier to do smart word matching rather than relying on the wildcard character.

Queries can be combined using a C-like syntax for AND (&&) and OR (||). For example, to restrict a Spotlight query to audio files authored by “Steve” the query would be:

```
kMDItemAuthors == "Steve"wc && kMDItemContentType == "audio"wc
```

Parenthesis can be used to further group query matching. For example, to search for audio files authored by “Steve” or “Kevin” the query would be:

```
(kMDItemAuthors == "Kevin"wc || kMDItemAuthors == "Steve"wc) &&
 kMDItemContentType == "audio"wc
```

You can expand this search to include video files using the following query:

```
(kMDItemAuthors == "Kevin"wc || kMDItemAuthors == "Steve"wc ) &&
 (kMDItemContentType == "audio"wc || kMDItemContentType == "video"wc )
```

You can also create queries that use the date and time as the search value. The date and time value is formatted as a floating-point value that is compatible with CFDate, seconds relative to January 1, 2001.

Additionally, the `$time` variable is provided that can be used to specify values relative to the current time, as shown in Table 5.

Table 5 $time variable expressions

Time variable	Description
`$time.now`	The current date and time.
`$time.today`	The current date.
`$time.yesterday`	Yesterday's date.
`$time.last_week`	Dates in before this week.
`$time.this_week`	Dates in the current week.
`$time.this_month`	Dates in the current month.
`$time.this_year`	Dates in the current year.
`$time.now(NUMBER)`	The date and time by adding a positive or negative value, in seconds, to the current time.
`$time.today(NUMBER)`	The date by adding a positive or negative value, in days, to the current day
`$time.this_week(NUMBER)`	Dates by adding a positive or negative value, in weeks, to the current week.
`$time.this_month(NUMBER)`	Dates by adding a positive or negative value, in months, to the current month.
`$time.this_year(NUMBER)`	Dates by adding a positive or negative value, in years, to the current year.
`$time.iso(ISO-8601-STR)`	The date by parsing the specified ISO-8601-STR compliant string.

Using the `$time` variable you can restrict a search to find only files that have been changed in the last week using the following query:

```
((kMDItemAuthors == "Kevin"wc || kMDItemAuthors = "Steve"wc) &&
 (kMDItemContentType == "audio"wc || kMDItemContentType = "video"wc)) &&
 (kMDItemFSContentChangeDate > $time.last_week)
```

Note: The value of the `$time` variable is set when the query is first executed. It does not update to the current time as the query continues to run.

Document Revision History

This table describes the changes to *Spotlight Query Programming Guide*.

Date	Notes
2006-03-08	Added $time.last_week to the query expression table.
2005-08-11	Added usage note about inRange query expression. Corrected minor typos.
2005-04-29	Added conceptual article that describes how to create Spotlight queries. Added usage note about $time to "Query Expression Format."
	New document that describes how to add Spotlight searching to your applications.