Spotlight Importer Programming Guide

Carbon > File Management

2007-05-27

Contents

Tables and Listings

Introduction to Spotlight Importer Programming Guide

Spotlight metadata importers allow Mac OS X to extract metadata from custom document formats.

Who Should Read This Document

You should read this document if your application saves custom document types to disk. All applications that support saving documents to disk should consider providing Spotlight support by implementing a metadata importer.

Organization of This Document

The following articles cover key concepts in understanding how metadata importers work:

- "Extracting Metadata from Documents" (page 9) describes the role of the metadata importer and its components.
- "Assigning Values to Metadata Attributes" (page 11) provides an overview of the Spotlight metadata attributes and creating your own custom attributes.
- "Spotlight Importer Schema Format" (page 13) describes the format of a Spotlight importer schema file.

These articles contain tasks that teach you how to implement metadata importers:

- "Spotlight Importer Performance" (page 25) describes important performance considerations.
- "Writing a Spotlight Importer" (page 17) describes how to write a metadata importer.
- "Troubleshooting Spotlight Importers" (page 27) describes how to test and diagnose problems with your Spotlight importers.

See Also

There are other aspects of Spotlight metadata, not covered by this document,that are fundamental to implementing a metadata importer. For example, this document does not explain the commonly used metadata keys or provide guidelines on using those keys to their full potential. Refer to these documents for more details:

- *Spotlight Overview* provides a conceptual overview of Spotlight.
- *Spotlight Metadata Attributes Reference* describes the metadata keys Apple provides.

Extracting Metadata from Documents

For Spotlight searching to work, it has to have metadata. While some metadata (modification dates, display name, path name) is easy to gather for a given file, most of the interesting data is embedded inside the file. To gather this embedded information you must provide a Spotlight importer.

What Is a Spotlight Importer?

A Spotlight importer is a small plug-in bundle that you create to extract information from files created by your application. Spotlight importers are used by the Spotlight engine to gather information about new and existing files.

Note: It is imperative that developers provide metadata importers for their own custom document formats. Spotlight metadata importers improve the user experience greatly by making sure your documents can be found during searches.

Spotlight importers parse your document format for relevant information and assigning that information to the appropriate metadata keys. Keys help index the content in the data store and facilitate searches. Xcode includes a project template that provides the required CFPlugin support, as well as templates for the required schema file.

Spotlight importers typically reside within your application's bundle in the subdirectory `MyApp.app/Contents/Library/Spotlight`. They can also be installed in `~/Library/Spotlight`, `/Library/Spotlight`, and `Framework/PlugIn`. System provided importers reside in `/System/Library/Spotlight`.

Associating a Spotlight Importer With Documents

Spotlight importers are associated with document types by specifying the uniform type identifiers (UTIs) from which they extract data. For more information on Uniform Type Identifiers see *Uniform Type Identifiers Overview*.

The supported UTI types are specified in the importer's `Info.plist` file, contained within the plug-in bundle. An importer can support a single document type or multiple document types. The function in the importer that is called for each file is passed the UTI type of the file and can adjust its extraction means as appropriate.

Additional Guidelines

Avoid the use of external files to store metadata content. All critical metadata should be in the same file as the data. The system store of metadata should be considered volatile.

A Spotlight importer must run entirely without interaction. You should not attempt to present any user interface or expect that the window server is running.

You should not expect your application to be running when your metadata importer is called. Importers can be called at any time to extract metadata from a file. Your metadata importer should be able to extract the information without any assistance from the application that created the file.

It is important to let users know what metadata you include in your file formats and what information you extract for searching. For example, users may not want their user ID or other personal information embedded in files they distribute externally. Consider giving the user an option to save a copy of the file without metadata for external distribution, or disable the extraction of metadata that has security implications.

Assigning Values to Metadata Attributes

Spotlight defines standard metadata attributes that provide a wide range of options for storing your document's metadata. In order for users to be able to find data easily, it is important to use existing keys whenever possible.

Spotlight's Metadata Attributes

Spotlight provides predefined metadata attributes for the following:

- File system attributes. For example, file size, owner, and modification date. These are extracted from the file system automatically by Spotlight.
- Image related attributes. For example, bits per sample, color space, pixel height, and width.
- Video related attributes. For example, codec, video bit rate, and audio bit rate.
- Audio related attributes. For example, sample rate, track number, composer, and time signature.
- Attributes common to many applications. For example, authors, city, organization, email addresses, and headline.

The Spotlight provided metadata attributes are documented in *Spotlight Metadata Attributes Reference*.

In addition to the data-specific attributes Spotlight provides a general text attribute (`kMDItemTextContent`) that importers can populate with a text representation of a document's content. Applications can create queries that reference this attribute, but are not able to read the value of this attribute directly.

You should avoid creating new metadata attributes if an existing key would be appropriate. For example, if your document tracks the photographer of an image, use the `kMDItemAuthors` attribute rather than defining a custom photographer key. Or, if your document includes a company name, use the `kMDItemOrganizations` attribute.

See "Assigning Values to Metadata Attributes" (page 22) in "Writing a Spotlight Importer" (page 17) for an example of how to assign values to metadata attributes.

Localizing Metadata Attribute Values

A Spotlight importer can provide localized values for an attribute by returning a dictionary object instead of a string value. The dictionary must contain keys that correspond to the localized languages. For example "en" for English, "fr" for French, etc. The value for each key should be the corresponding localized attribute value.

Note: In Mac OS X v10.4 only attributes that return a single string value can provide localized attribute values. Attributes that return multiple values in an array can not be localized.

Defining Custom Attributes

If none of the existing Spotlight attributes are appropriate or adaptable to your metadata, you can define a custom metadata attribute. An importer specifies the name of the custom attribute, as well as the type of data it contains, in its `schema.xml` file.

Attribute Naming Conventions

Custom metadata attributes must have unique names. To ensure this you use the reverse DNS naming convention as a prefix for keys that are specific to your document types, replacing "." with "_" characters. For example, the Mail program would prefix its custom attributes with `com_apple_mail`.

Defining the Value Object Type

You must specify the type of object that is returned in your custom attribute. The supported types are: CFString, CFNumber, CFBoolean, and CFDate.

Returning Multiple Values in an Attribute

Attributes that return an array of objects rather than a single object are said to be multivalued. If your custom attributes can contain multiple objects, you should declare them as multivalue in your importer's schema file and always return an array, even if it contains only a single instance.

Attribute Display Names and Descriptions

Spotlight importers that declare custom metadata attributes should also provide a display name and description for each attribute. These strings are contained in the file `schema.strings` in your importer bundle.

The file must be UTF-16 text encoded formatted as a standard strings file. The display name keys correspond to the custom metadata attribute's name. The description string is specified by appending "`.Description`" to the key name. Listing 1 shows a sample schema.strings file.

Listing 1 Sample importer's schema.strings file

```
"com_apple_myCocoaDocumentApp_myCustomDocument_notes" = "Notes";
"com_apple_myCocoaDocumentApp_myCustomDocument_notes.Description" = "What it is
 you're supposed to remember.";
```

You can localize `schema.strings` files using the standard conventions.

Spotlight Importer Schema Format

For Spotlight to know what attributes an importer supports, it must provide a schema file. The schema file describes the attributes that the importer populates, describes the attributes that applications should use to provide a preview of the document's metadata, and specifies any custom metadata attributes that your documents require.

The Schema.xml File

The schema is specified in an XML schema file called `schema.xml` within your Spotlight importer bundle.

The following XML fragment shows the general format of the file.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema version="1.0" xmlns="http://www.apple.com/metadata"
                      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                      xsi:schemaLocation="http://www.apple.com/metadata
file:///System/Library/Frameworks/CoreServices.framework/Frameworks/Metadata.framework/
Resources/MetadataSchema.xsd">
    <attributes>
    ...
   </attributes>
    <types>
        <type name="SUPPORTED_UTI_TYPE">
            <allattrs>
        ...
            </allattrs>
            <displayattrs>
        ...
            </displayattrs>
        </type>
    </types>
</schema>
```

Specifying Custom Attributes

Custom attributes for your Spotlight importer are declared as `attribute` elements that are children of the `attributes` element. The XML attributes for the `attribute` element are shown in Table 1.

Table 1 Attributes of attribute element

Attributes	Description
name	The name of the custom metadata attribute. The metadata attributes are prefixed with the reverse DNS naming schema, replacing "." with "_" for key-value coding compatibility.
type	The data type that the attribute returns. Only the following CF types are supported: CFString, CFNumber, CFBoolean and CFDate.
multivalued	If the importer returns an array of values for this metadata attribute this attribute should be "true". If this attribute is omitted, "false" is assumed.
uniqued	If the importer returns only a small number of possible values for an attribute, space in the system store can be saved by setting this attribute to "true". If this attribute is omitted, "false" is assumed. This attribute is optional, and should only be specified when there is a very small number of values possible for the attribute.
nosearch	If set to "true" this attribute is only searched when it is specifically declared as a target metadata attribute in the search string. If this attribute is omitted, "false" is assumed and all wildcard attribute searches will include the values of this metadata attribute.

The following is an example XML fragment for the `attributes` element of a schema.

```
<attributes>
    <attribute name="com_apple_myCocoaDocumentApp_myCustomDocument_notes"
multivalued="false" type="CFString"/>
</attributes>
```

Specifying the Attributes for a Document

There is a single `type` element for each document type that your importer can read. The XML attributes for the `type` element are shown in Table 2 (page 14).

Table 2 Attributes of type element

Attributes	Description
name	The Uniform Type Identifier declared for the document type.

A `type` element specifies the metadata attributes that it returns in the `allattrs` element, separating each name with whitespace. The `allattrs` element should contain all the elements related to your custom document.

The metadata attributes to be displayed for previewing for a document—for example in Finder's Get Info panel—are listed within the `displayattrs` element, separating each name with whitespace.

The following is an example XML fragment for a `types` element of a schema.

```
<types>
    <type name="com.apple.mycocoadocumentapp.mycustomdocument">
```

```
        <allattrs>
            com_apple_myCocoaDocumentApp_myCustomDocument_notes
        </allattrs>
        <displayattrs>
            com_apple_myCocoaDocumentApp_myCustomDocument_notes
        </displayattrs>
    </type>
</types>
```

Writing a Spotlight Importer

Spotlight importers should be provided by all applications that support custom document formats. A Spotlight importer parses your document format for relevant information and assigning that information to the appropriate metadata keys.

An example metadata importer that extracts metadata from a custom document is included in the `/Developer/Examples/Metadata/ImporterExample`. This example is referred to throughout this article.

Creating the Metadata Importer Project

Xcode provides a project template, Metadata Importer, that provides the functionality commonly shared by importers.

This template creates a project with the required frameworks, a template for the `Info.plist`, a template for the schema file, a template for the localizable `schema.strings` file, a template for the `main.c` file that contains the necessary CFPlugin implementation and `GetMetadataForFile.c`, a skeleton implementation of the required callback function. The target creates a CFPlugin bundle with an `mdimporter` extension.

In addition to writing the extraction code, you'll need to modify the templates to specify the document types your importer handles and list the keys your importer provides.

Assigning a Unique ID to the Import Function

Each plug-in factory that can import metadata must have a unique identification number associated with it. Typically, there is only a single plug-in factory for each metadata importer, as a single function can handle many document types.

When you create a new metadata importer project, Xcode creates a UUID for your importer. Here is the UUID xCode generated for the sample metadata importer.

```
8AED83B3-C412-11D8-85A3-000393D59866
```

This value is used in the importer's `Info.plist`, as well as the `main.c` file. Listing 1 shows the Info.plist template that was generated by Xcode.

Listing 1 Metadata importer Info.plist template

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
 "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>CFBundleDevelopmentRegion</key>
    <string>English</string>
    <key>CFBundleDocumentTypes</key>
    <array>
        <dict>
            <key>CFBundleTypeRole</key>
            <string>MDImporter</string>
            <key>LSItemContentTypes</key>
            <array>
                <string>SUPPORTED_UTI_TYPE</string>
            </array>
        </dict>
    </array>
    <key>CFBundleExecutable</key>
    <string>MyCustomImporter</string>
    <key>CFBundleIconFile</key>
    <string></string>
    <key>CFBundleIdentifier</key>
    <string>com.apple.yourcfbundle</string>
    <key>CFBundleInfoDictionaryVersion</key>
    <string>6.0</string>
    <key>CFBundlePackageType</key>
    <string>BNDL</string>
    <key>CFBundleSignature</key>
    <string>????</string>
    <key>CFBundleVersion</key>
    <string>1.0</string>
    <key>CFPlugInDynamicRegisterFunction</key>
    <string></string>
    <key>CFPlugInDynamicRegistration</key>
    <string>NO</string>
    <key>CFPlugInFactories</key>
    <dict>
        <key>8AED83B3-C412-11D8-85A3-000393D59866</key>
        <string>MetadataImporterPluginFactory</string>
    </dict>
    <key>CFPlugInTypes</key>
    <dict>
        <key>8B08C4BF-415B-11D8-B3F9-0003936726FC</key>
        <array>
            <string>8AED83B3-C412-11D8-85A3-000393D59866</string>
        </array>
    </dict>
    <key>CFPlugInUnloadFunction</key>
    <string></string>
</dict>
</plist>
```

The `CFPlugInFactories` entry is a dictionary that associates the metadata importer host ID to the UUIDs of the plug-in factory function an importer requires. The `CFPluginInTypes` dictionary contains keys that associate the UUID of the factory function to the function. In both locations, Xcode inserted the newly generated UUID.

Here is the relevant line from the `main.c` template that Xcode created.

Listing 2 Setting the importer ID in main.c

```
#define PLUGIN_ID "8AED83B3-C412-11D8-85A3-000393D59866"
```

Associating an Importer with Document Types

An importer must be associated with the document types that it can import. You do this by specifying the Uniform Type Identifiers (UTIs) that correspond to the supported documents.

The supported UTIs are specified in the `LSItemContentTypes` array in the importer's `Info.plist`. The template in Listing 1 (page 17) includes a placeholder, `SUPPORTED_UTI_TYPE`, that you should replace with the UTI that your importer handles. If more than one document type is supported you can add additional string entries to the `LSItemContentTypes` array in the `Info.plist`. In the example importer, the `SUPPORTED_UTI_Type` is `com.apple.mycocoadocumentapp.mycustomdocument`.

Note: If an importer reads metadata from a document package you must add `com.apple.package` to the array of UTIs declared in the `UTTypeConformsTo` entry.

If your application does not define a UTI for its document types, you can declare one in your importer's `Info.plist` by adding the `UTExportedTypeDeclarations` key. Standalone importers that don't correspond to an application should declare the UTIs that they support by specifying a `UTImportedTypeDeclarations` key. The `UTImportedTypeDeclarations` format is the same as the `UTExportedDeclarations` format shown in Listing 3 (page 19). See "Uniform Type Identifier Concepts" for more information on declaring UTIs.

Listing 3 UTExportedTypeDeclarations format

```
<key>UTExportedTypeDeclarations</key>
<array>
    <dict>
        <key>UTTypeIdentifier</key>
        <string>com.yourcompany.yourUTI</string>
        <key>UTTypeReferenceURL</key>
        <string>http://www.company.com/yourproduct</string>
        <key>UTTypeDescription</key>
        <string>Your Document Kind String</string>
        <key>UTTypeConformsTo</key>
        <array>
            <string>public.data</string>
            <string>public.content</string>
        </array>
        <key>UTTypeTagSpecification</key>
        <dict>
            <key>com.apple.ostype</key>
            <string>XXXX</string>
            <key>public.filename-extension</key>
            <array>
                <string>xxxx</string>
            </array>
        </dict>
    </dict>
</array>
```

Specifying Metadata Attributes

You need to specify the metadata attributes that your metadata importer returns by modifying the project's `schema.xml` file. This is an XML Schema document that provides details on the returned attributes and allows you to specify custom metadata keys as well.

Listing 4 shows the `schema.xml` template generated by Xcode.

Listing 4 Metadata Importer schema.xml template

```
<?xml version="1.0" encoding="UTF-8"?>

<schema version="1.0" xmlns="http://www.apple.com/metadata"
                      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                      xsi:schemaLocation="http://www.apple.com/metadata
file:///System/Library/Frameworks/CoreServices.framework/Frameworks/Metadata.framework/Resources/MetadataSchema.xsd">
    <note>
        Custom attributes that this metadata importer supports.  Below
    is an example of a multivalued string attribute.  Other types
    are CFNumber, CFDate, CFBoolean and CFData.
    </note>
    <attributes>
        <attribute name="com_Foo_YourAttrName" multivalued="true"
type="CFString"/>
    </attributes>

    <types>
        <type name="SUPPORTED_UTI_TYPE">
            <note>
        The keys that this metadata importer handles.
            </note>
            <allattrs>
        com_Foo_YourAttrName
            </allattrs>
            <displayattrs>
        com_Foo_YourAttrName
            </displayattrs>
        </type>
    </types>
</schema>
```

You must edit this template to suit your metadata importer.

1. Replace the `SUPPORTED_UTI_TYPE` placeholder with the appropriate UTI type for your document.

2. Edit the `attributes` element, editing or removing the `attribute` elements as required.

   The metadata keys are prefixed with the reverse DNS naming schema, replacing "_" with "." for key-value coding compatibility. Each of these custom metadata values return a single CFString as specified by the `type` attribute of the `attribute` element.

   Metadata importers can only return the following CF types: CFString, CFNumber, CFBoolean, and CFDate. If a key returns an array of values, the `type` attribute specifies the CF type and the `attribute` element must include a `multivalued` attribute with a value of `true`.

If your importer does not require custom metadata keys, you can remove the `attributes` element entirely.

3. Edit the `allattrs` element so that it contains all your metadata keys.
4. Edit the `displayattrs` element so that it contains a subset of your metadata keys that are recommended for previewing.
5. Edit the `schema.strings` file to provide display name and description strings for your custom metadata keys.

Listing 5 shows the `schema.xml` file that is included with the sample metadata importer project.

Listing 5 schema.xml file for the sample metadata importer

```
<?xml version="1.0" encoding="UTF-8"?>

<schema version="1.0" xmlns="http://www.apple.com/metadata"
                      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                      xsi:schemaLocation="http://www.apple.com/metadata
file:///System/Library/Frameworks/CoreServices.framework/Frameworks/Metadata.framework/Resources/MetadataSchema.xsd">
    <note>
        Custom attributes that this metadata importer supports.  Below
    is an example of a multivalued string attribute.  Other types
    are CFNumber, CFDate, CFBoolean and CFData.
    </note>
    <attributes>
        <attribute name="com_apple_myCocoaDocumentApp_myCustomDocument_notes"
multivalued="false" type="CFString"/>
    </attributes>

    <types>
        <type name="com.apple.mycocoadocumentapp.mycustomdocument">
            <note>
        The keys that this metadata importer handles.
            </note>
            <allattrs>
        com_apple_myCocoaDocumentApp_myCustomDocument_notes
            </allattrs>
            <displayattrs>
        com_apple_myCocoaDocumentApp_myCustomDocument_notes
            </displayattrs>
        </type>
    </types>
</schema>
```

The sample metadata importer declares one new attribute key, `com_apple_myCocoaDocumentApp_myCustomDocument_notes`. The key name is prefixed with the reverse DNS naming schema, replacing "_" with "." for key-value coding compatibility. Each of these custom metadata values return a single CFString as specified by the `type` attribute of the `attribute` element.

Listing 6 shows the schema.strings file for the sample metadata importer.

Listing 6 Sample importer's schema.strings file

```
"com_apple_myCocoaDocumentApp_myCustomDocument_notes" = "Notes";
```

```
"com_apple_myCocoaDocumentApp_myCustomDocument_notes.Description" = "What it is
 you're supposed to remember.";
```

The command-line tool `mdcheckschema` performs a simple validation on a schema and is useful when testing your own importer schema for validity.

Assigning Values to Metadata Attributes

When metadata is extracted for a file, the `GetMetadataForFile` function is called. The function is passed the plug-in interface, a mutable dictionary that you'll add the metadata attribute keys and values to, the UTI type of the target file, and the full path to the target file.

Listing 7 shows the `GetMetadataForFile` skeleton implementation provided by Xcode in `GetMetadataForFile.c`.

Listing 7 GetMetadataForFile template implementation

```
Boolean GetMetadataForFile(void* thisInterface,
              CFMutableDictionaryRef attributes,
              CFStringRef contentTypeUTI,
              CFStringRef pathToFile)
{
    /* Pull any available metadata from the file at the specified path */
    /* Return the attribute keys and attribute values in the dict */
    /* Return true if successful, false if there was no data provided */

    #warning To complete your importer please implement the function
GetMetadataForFile in GetMetadataForFile.c
    return false;
}
```

Your implementation of this function should extract the metadata from the file and insert it into the dictionary with the appropriate keys and values. If it successfully returns metadata, the function should return with a value of `true`. If no metadata was extracted, you should return `false`.

The example's custom document format is a simple property list containing the author, title and reminder notes. Note that the example makes use of Objective-C and the Foundation class NSDictionary to read the dictionary from the file. In order to use Objective-C in your `GetMetadataForFile` implementation you must rename `GetMetadataForFile.c` to `GetMetadataForFile.m`. Listing 8 shows the `GetMetadataForFile` implementation of the example metadata importer.

Listing 8 Objective-C implementation of GetMetadataForFile for the sample metadata importer

```
Boolean GetMetadataForFile(void* thisInterface,
              CFMutableDictionaryRef attributes,
              CFStringRef contentTypeUTI,
              CFStringRef pathToFile)
{
    /* Pull any available metadata from the file at the specified path */
    /* Return the attribute keys and attribute values in the dict */
    /* Return true if successful, false if there was no data provided */
    Boolean success=NO;
    NSDictionary *tempDict;
    NSAutoreleasePool *pool;

    // Don't assume that there is an autorelease pool around the calling of this
 function.
    pool = [[NSAutoreleasePool alloc] init];
    // load the document at the specified location
    tempDict=[[NSDictionary alloc] initWithContentsOfFile:(NSString *)pathToFile];
    if (tempDict)
    {
    // set the kMDItemTitle attribute to the Title
    [(NSMutableDictionary *)attributes setObject:[tempDict objectForKey:@"title"]
                          forKey:(NSString *)kMDItemTitle];

    // set the kMDItemAuthors attribute to an array containing the single Author
    // value
    [(NSMutableDictionary *)attributes setObject:[NSArray
arrayWithObject:[tempDict objectForKey:@"author"]]
                          forKey:(NSString *)kMDItemAuthors];

    // set our custom document notes attribute to the Notes value
    // (in the real world, you'd likely use the kMDItemTextContent attribute,
however that
    // would make it hard to demonstrate using a custom key!)
    [(NSMutableDictionary *)attributes setObject:[tempDict objectForKey:@"notes"]

forKey:@"com_apple_myCocoaDocumentApp_myCustomDocument_notes"];

    // return YES so that the attributes are imported
    success=YES;

    // release the loaded document
    [tempDict release];
    }
    [pool release];
    return success;
}
```

Spotlight Importer Performance

A Spotlight importer is called upon as files are created, copied, and modified, so performance is crucial. Importers should be able to extract metadata from documents quickly and with minimal effort.

Avoid burying metadata deep inside a file, especially if finding that metadata would be computationally intensive later. If needed, define your file format so that relevant information is in the header or in an easily accessible location.

It is also vital that your importer does not leak memory, as this can contribute to performance problems.

Troubleshooting Spotlight Importers

Rare is the project that is flawless from the start. Troubleshooting a Spotlight importer can be difficult given that it is run by the system automatically as required and is run outside the development environment.

This article describes how to explicitly run your metadata importer for testing and provides a number of techniques for troubleshooting problems.

Where should I install my Spotlight importer?

Your Spotlight importer typically resides within your application's bundle in the subdirectory `MyApp.app/Contents/Library/Spotlight`. Importers can also be installed in one of the following locations:

```
~/Library/Spotlight
/Library/Spotlight
```

If your importer is not part of an application bundle, you should create an installer package that installs the importer in one of the above locations. In order to have existing files imported, you will need to have a `postinstall` script for your installer that includes the following command, specifying your importer install location:

```
/usr/bin/mdimport -r InstallDirectory/YourPlug-In
```

When will the Spotlight importer in my application bundle re-index files?

When a user first runs your application, the Spotlight importer is found and Spotlight will begin importing any existing documents. If you update your application and the Spotlight importer, you should ensure that the importer bundle has a different date stamp. When the updated application is run for the first time Spotlight will re-index the existing files with the new importer.

Note: This behavior was added in the Mac OS X 10.4.1.

How can I determine if my Spotlight importer is being found?

Running the `mdimport` command (located in `/usr/bin`) with the `-L` option returns a list of all the currently recognized importers and their paths.

```
/usr/bin/mdimport -L

2005-01-16 02:56:37.634 mdimport[673] Paths: id(501) (
    "/System/Library/Spotlight/RichText.mdimporter",
    "/System/Library/Spotlight/Image.mdimporter",
    "/System/Library/Spotlight/Audio.mdimporter",
    "/System/Library/Spotlight/Font.mdimporter",
    "/System/Library/Spotlight/PDF.mdimporter",
    "/System/Library/Spotlight/Chat.mdimporter",
    "/System/Library/Spotlight/iCal.mdimporter",
    "/System/Library/Spotlight/Mail.mdimporter",
    "/System/Library/Spotlight/QuickTime.mdimporter",
    "/System/Library/Spotlight/vCard.mdimporter",
    "/Users/me/Library/Spotlight/MyCustomImporter.mdimporter",
    "/System/Library/Spotlight/QuartzComposer.mdimporter",
    "/System/Library/Spotlight/PS.mdimporter",
    "/System/Library/Spotlight/SystemPrefs.mdimporter",
    "/System/Library/Spotlight/Application.mdimporter"
)
```

Why isn’t my bundled importer being found?

If your importer resides within your application’s wrapper, it may not be found automatically during testing. Importers are detected when the bundle’s modification date is changed. You can explicitly register your application by specifying the `-f` flag to `lsregister`. The `lsregister` tool is found in `/System/Library/Frameworks/ApplicationServices.framework/Versions/A/Frameworks/LaunchServices.framework/Versions/A/Support/`.

```
lsregister -f MyApp.app
```

Another possibility is that your application may be untrusted. Spotlight importers are not loaded from untrusted applications. Launching the application for the first time causes the application to be trusted.

I've updated my importer and copied it to a Spotlight directory, but the old importer is still being used

New Spotlight importers are detected by comparing the date of the top-level .mdimporter directory. If the date is the same as a previously loaded importer, the new importer is not detected. If you copy the updated importer to the Spotlight directory using `cp -r` the change is not noted by Spotlight. The solution is to either remove the existing importer before copying the updated version, or use the `touch` command on the importer's .mdimporter directory to explicitly update the date.

How do I test my importer?

You can test your Spotlight importer using the `mdimport` command (located in `/usr/bin`). Run `mdimport` with the debug level set to 2 and specify a file that you can import data from:

```
/usr/bin/mdimport -d2 test.myCustomDocument
```

This command produces out like this:

```
2005-01-16 02:59:04.930 mdimport[678] Import
'/Users/me/Documents/test.myCustomDocument'
type 'com.apple.mycocoadocumentapp.mycustomdocument'
using 'file://localhost/Users/me/Library/Spotlight/MyCustomImporter.mdimporter/'
2005-01-16 02:59:04.931 mdimport[678] Sending attributes
of '/Users/me/Documents/test.myCustomDocument' to server.
Attributes: '{
    "_kMDItemImporterCrashed" = <null>;
    "com_apple_metadata_modtime" = 127555123.1940155;
    "com_apple_myCocoaDocumentApp_myCustomDocument_notes" = "Remember to feed
the cats!";
    kMDItemAuthors = ("Tori”,”Simon",”Daniel”);
    kMDItemContentType = "com.apple.mycocoadocumentapp.mycustomdocument";
    kMDItemContentTypeTree = ("com.apple.mycocoadocumentapp.mycustomdocument",
 "public.data", "public.item");
    kMDItemDisplayName = {"" = "test.myCustomDocument"; };
    kMDItemKind = {en = DocumentType; };
    kMDItemTitle = "Be sure to remember to...";
}'
```

The first line of the output indicates the file that is being imported, as well as the UTI that the file maps to. The remaining lines list the attribute keys and values that were imported from the file.

You should ensure that all the appropriate metadata keys that your importer returns are present in the output. You'll notice that a number of metadata keys specific to the file system are also available for each file. These are provided by the metadata system and are not your responsibility.

How do I debug my importer using gdb?

You can debug your importer by running `mdimport` under `gdb`.

The following command will load `mdimport` under `gdb`:

```
gdb mdimport
```

Once `mdimport` has started set a breakpoint on your import function:

```
b MyImporterGetAttributesFromFileFunction
```

Then start the `mdimport` process, specifying the file to import:

```
r /path/to/my/test/file
```

What does the system think the UTI is for a document?

You can determine the UTI that the system thinks belongs to your file by using the `mdimport` command with a debug level of 1:

```
/usr/bin/mdimport -d1 test.myCustomDocument
```

The output shows the UTI type that the system has determined for the file:

```
 2005-01-16 03:00:07.212 mdimport[683] Import '/Users/me/Documents/
test.myCustomDocument'
 type 'com.apple.mycocoadocumentapp.mycustomdocument'  using
'file://localhost/Users/me/Library/Spotlight/MyCustomImporter.mdimporter/'
```

The type should match the UTI that your importer supports.

Running mdimport returns nothing

If running `mdimport` with a debug level of 1 returns no output, you should ensure that the file you're attempting to import is not in the `/tmp` directory or some other System directory. Files in those locations are not imported.

Running mdimport returns an unexpected UTI

If running `mdimport` returns a UTI other than one you expect, you'll need to ensure that the file you're attempting to import is actually the type of file you think it is. The UTI of a file is determined by the extension or file type.

It is also possible that a dynamic UTI is returned:

```
2005-01-16 03:01:16.989 mdimport[691] Import
'/Users/me/Documents/test.myCustomDocument'
 type 'dyn.ah62d4rv4ge8048pdsz31k55rqv10g7prqz1hkqu' no mdimporter
```

Typically, the return of a dynamic UTI indicates that the file is not mapping to a known UTI. You should check that:

1. The test file is the correct file type.
2. The test file has the correct filename extension or file type set.
3. If your application is declaring the UTI type for the document, that the application's `Info.plist` file has the correct entries in the `UTExportedTypeDeclarations` entry as shown here:

```
<key>UTExportedTypeDeclarations</key>
<array>
    <dict>
        <key>UTTypeConformsTo</key>
        <array>
            <string>public.data</string>
        </array>
        <key>UTTypeDescription</key>
        <string>My Document Type</string>
        <key>UTTypeIdentifier</key>
        <string>com.apple.mycocoadocumentapp.mycustomdocument</string>
        <key>UTTypeTagSpecification</key>
        <dict>
            <key>public.filename-extension</key>
```

```
                <array>
                    <string>myCustomDocument</string>
                </array>
            <key>com.apple.ostype</key>
            <string>T78q</string>
            </dict>
        </dict>
    </array>
```

4. Ensure that your application lives in a location where Launch Services can detect the mappings. Running the application also ensures that the mappings are made.

mdimport does not return my metadata attributes

If running `mdimport` with a debug level of 3 does not list any of your custom metadata attributes, you should check that:

1. Your metadata importer is being found using the `mdimport -L` command.

2. Your metadata importer's `Info.plist` file has the correct plug-in type for metadata importers in the `CFPlugInTypes` entry. The key should be `8B08C4BF-415B-11D8-B3F9-0003936726FC`:

```
<key>CFPlugInTypes</key>
<dict>
    <key>8B08C4BF-415B-11D8-B3F9-0003936726FC</key>
    <array>
        <string>8AED83B3-C412-11D8-85A3-000393D59866</string>
    </array>
</dict>
```

3. The UUID that you created for your importer is unique and is in both the `CFPlugInFactories` and `CFPluginTypes` entries of the importer's `Info.plist` file. Here, the UUID is `8AED83B3-C412-11D8-85A3-000393D59866`:

```
<key>CFPlugInFactories</key>
<dict>
    <key>8AED83B3-C412-11D8-85A3-000393D59866</key>
    <string>MetadataImporterPluginFactory</string>
</dict>
<key>CFPlugInTypes</key>
<dict>
    <key>8B08C4BF-415B-11D8-B3F9-0003936726FC</key>
    <array>
        <string>8AED83B3-C412-11D8-85A3-000393D59866</string>
    </array>
</dict>
```

4. You have the correct UTI type for your importer listed in the `LSItemContentTypes` entry in the importer's `Info.plist` file:

```
<key>CFBundleDocumentTypes</key>
<array>
    <dict>
        <key>CFBundleTypeRole</key>
```

```
                <string>MDImporter</string>
                <key>LSItemContentTypes</key>
                <array>
                    <string>com.apple.mycocoadocumentapp.mycustomdocument</string>
                </array>
            </dict>
        </array>
```

5. Your UTI is all lowercase in the `Info.plist` and the `schema.xml` files.

6. If your importer reads metadata from a document package ensure that the `UTTypeConformsTo` entry in the importer's `Info.plist` includes `com.apple.package` as a UTI.

7. Your `schema.xml` file is valid.

   You can test whether your `schema.xml` file is well formed by running the command `mdcheckschema` (located in `/usr/bin`).

   ```
   /usr/bin/mdcheckschema
   ~/Library/Spotlight/MyCustomImporter.mdimporter/Contents/Resources/schema.xml

   /Users/me/Library/Spotlight/MyCustomImporter.mdimporter/Contents/Resources/schema.xml
    : succesfully parsed.
   ```

8. Your implementation of `GetMetadataForFile` is populating the dictionary with the correct metadata entries and is returning `true`.

9. You return only CFTypes of CFString, CFNumber, CFBoolean, and CFDate as attribute values. If an attribute is specified as multivalued, you must return a CFArray of the expected CFType.

What are the imported metadata attributes for a specific file?

You can determine the metadata attributes and values in the system store for a file by using the `mdls` command:

```
mdls /Applications/iTunes.app

/Applications/iTunes.app -------------
kMDItemAttributeChangeDate = 2005-01-16 03:03:14 -0500
kMDItemContentType         = "com.apple.application-bundle"
kMDItemContentTypeTree     = (
    "com.apple.application-bundle",
    "com.apple.application",
    "public.executable",
    "com.apple.bundle",
    "public.directory",
    "public.item",
    "com.apple.package"
)
kMDItemCopyright           = "iTunes 4.7, Copyright 2000-2004 Apple Computer,
Inc."
kMDItemDisplayName         = "iTunes"
kMDItemFSContentChangeDate = 2005-01-08 18:17:52 -0500
kMDItemFSCreationDate      = 2005-01-08 18:17:52 -0500
```

```
kMDItemFSCreatorCode       = 0
kMDItemFSFinderFlags       = 0
kMDItemFSInvisible         = 0
kMDItemFSLabel             = 0
kMDItemFSName              = "iTunes.app"
kMDItemFSNodeCount         = 1
kMDItemFSOwnerGroupID      = 80
kMDItemFSOwnerUserID       = 0
kMDItemFSSize              = 0
kMDItemFSTypeCode          = 0
kMDItemID                  = 64286
kMDItemKind                = "Application"
kMDItemLastUsedDate        = 2005-01-16 01:01:10 -0500
kMDItemUsedDates           = (
    2005-01-08 18:17:52 -0500,
    2005-01-13 19:00:00 -0500,
    2005-01-15 19:00:00 -0500
)
kMDItemVersion             = "4.7"
```

You can also specify a specific metadata attribute to return the value of:

```
mdls -name kMDItemContentType /Applications/iTunes.app

/Applications/iTunes.app -------------
kMDItemContentType = "com.apple.application-bundle"
```

Why isn't Spotlight finding my document bundles when they are saved by my application?

If your application saves its documents as a bundle, you must take precautions to ensure that Spotlight doesn't attempt to import your document before all the data is written to the bundle.

See "Spotlight and Document Bundles" for additional details.

Document Revision History

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

Date	Notes
2007-05-27	Clarified role of displayattrs and the Finder.
2006-11-07	Added additional debugging information to the Troubleshooting article.
2006-04-04	Added discussion of the kMDItemTextContent attribute to "Assigning Values to Metadata Attributes." Clarified that the GetMetdataForFile example is Objective-C.
2006-03-08	Updated to reflect Spotlight compatibility with Rosetta.
2006-02-07	Added note about recompiling as a universal binary.
2005-09-08	Added new troubleshooting information about Spotlight importer bundle timestamps and document packaging UTI types.
2005-08-11	Added discussion about localizing attribute values. Added template for UTExportedDeclarations. Added note about Objective-C use in importers.
2005-07-07	Added additional Schema.xml attributes. Added Installer.app package post-install script information for standalone importers.
2005-04-29	Updated to reflect the current Xcode project template. Added troubleshooting information. Changed title from "Metadata Importers." First public version.
2004-06-29	Corrected reference to `MetadataImporterPluginFactory` in "Writing a Spotlight Importer" (page 17).
	Added schema.xml template listing to "Writing a Spotlight Importer" (page 17).
2004-06-28	New document that describes the role of metadata importers and how to write them.