

# Using Media Cataloger

Documentation »  
ACCESS Linux Platform Native Development »  
Content Management »  
Using Media Cataloger

This chapter discusses how to use the Media Cataloger APIs.

**NOTE:** See the related reference material for the API in the "Content Management" Module and the `/usr/include/alp/mediaselector_api.h` File Reference. The online reference guide has the most current information about the API.

Overview [^TOP^](#)

The Media Cataloger, described in detail in Chapter 5, "Introducing Media Cataloger," controls the processes that collect and maintain file metadata stores. from the device or an external storage card. The Cataloger provides applications access to that information through the public data model. To specify search areas, and to retrieve the stored metadata about files, an application must be able to interact with the catalog. The set of APIs (`alp_ms_cat_*`) described in this chapter provide facilities for applications to interact with the catalog created and maintained by Media Cataloger.

The Cataloger has defined standard directories to search. However, if an application has created or stored specific directories associated with its media, it can add (or remove) those directories to the Cataloger's search. At a finer level of control, it can add (or remove) a single file to the file index, or copy a single file. If necessary, an application can request a catalog update. The set of `alp_ms_cat_get_registered_*` APIs provide the mechanism for an applications to traverse the catalog. An application can also check to see what media formats are recognized by the Cataloger.

Working with Media Cataloger APIs [^TOP^](#)

The following sections describe the functions used for cataloger tasks:

- "Registering or Unregistering a Directory."
- "Enumerating through the Registered Directories."
- "Updating the Catalog."
- "Indexing or Unindexing a Single File."
- "Copying a File."
- "Renaming a File".

- "Checking for Media Format Support."

## Registering or Unregistering a Directory ^TOP^

Use the following functions when your application needs to register or unregister a specific directory for the Cataloger to search and maintain. This allows your application to store files in a particular directory and have them added to (or deleted from) the Catalog's search.

The Cataloger controls registration of directories in the internal database that the Tree Walker uses to index media files. By default, the Tree Walker only monitors media files in a 'My Media' folder for the user, and users and applications should use that folder whenever possible to store media. But if an application wants to use a private directory, it must register (or unregister) that directory using a Cataloger API. The API maintains this list in a table in the Cataloger's private application database.

NOTE: Applications should always use the `alp_mf_get_media_path` function, described "Finding File Locations for Well-behaved Applications," to determine the standard location to put files.

Be aware that when you call `alp_ms_cat_register_directory`, to add the directory path, the Cataloger does an update, which is time consuming and can affect performance.

Use `alp_ms_cat_register_directory` for (persistent) paths on the device. Do not call it for (removable) paths on cards; this action returns an error code.

The value of path is the directory you want to add to the catalog. It should always be a path on the device never a path on the card. (Card mounts are implicitly registered. Attempting to register a card directory returns an error code, but reduces performance.)

The function returns the error `ALP_STATUS_MS_CAT_INVALID_PATH` if the path parameter is

- NULL.
- Not rooted at /.
- Not an actual directory.
- Is (or is under) a mounted card.

Registration is recursive. If `/foo` is registered, so is `/foo/bar`. Explicitly registering both `/foo` and `/foo/bar` causes files in `/foo/bar` to be cataloged twice!

NOTE: `home` is an alias for `/var/home` (which is registered). Do not register any directories under either `/home` or `/var/home`.

## alp\_ms\_cat\_register\_directory Function ^TOP^

### Purpose

Adds path to Cataloger media paths and does a recursive scan for media files.

### Prototype

```
alp_status_t alp_ms_cat_register_directory (  
    const char *path  
)
```

## alp\_ms\_cat\_unregister\_directory Function ^TOP^

### Purpose

Removes path from Cataloger and data from metadata db.

### Prototype

```
alp_status_t alp_ms_cat_unregister_directory (  
    const char *path  
)
```

## Enumerating through the Registered Directories ^TOP^

Use the following functions to initialize, step through, and clean up after enumerating over the registered directories.

## alp\_ms\_cat\_get\_registered\_begin Function ^TOP^

### Purpose

Enumerates over the registered directories, initializes enumeration.

### Prototype

```
alp_status_t alp_ms_cat_get_registered_begin (  
    AlpMSRegDirH *handle
```

)

alp\_ms\_cat\_get\_registered\_next\_dir Function ^TOP^

Purpose

Enumerates over the registered directories, gets next entry.

Prototype

```
alp_status_t alp_ms_cat_get_registered_next_dir (  
    AlpMSRegDirH handle,  
    char **directory  
)
```

alp\_ms\_cat\_get\_registered\_end Function ^TOP^

Purpose

Cleans up after enumeration over the registered directories.

Prototype

```
alp_status_t alp_ms_cat_get_registered_end (  
    AlpMSRegDirH handle  
)
```

Updating the Catalog ^TOP^

The `alp_ms_cat_update` function runs a Tree Walker on the specified directory, to immediately update a volume's category database.

The Cataloger normally launches a Tree Walker thread in response to certain system events (such as inserting a card); but this API gives finer control to applications. It tells the Tree Walker to begin indexing a certain directory, and it should be called whenever an application updates media files (more than one) so that the change can be immediately reflected in the appropriate meta database. Otherwise, the change won't be reflected until another application or system event forces the update.

If your application is only adding a single file, use `alp_ms_cat_index_file` for better performance.

alp\_ms\_cat\_update Function ^TOP^

## Purpose

Call when the contents of a registered directory change, to update the catalog. (When you add a single file, `alp_ms_cat_index_file` is significantly faster than `alp_ms_cat_update()`).

## Prototype

```
alp_status_t alp_ms_cat_update (  
    const char *path  
)
```

## Indexing or Unindexing a Single File ^TOP^

Use the following functions to add or remove a single file from the selection by indexing or unindexing it in the catalog

The `alp_ms_cat_index_file` and `alp_ms_cat_unindex_file` functions perform asynchronous operations. The return value `ALP_STATUS_OK` means the request was submitted. Before using these functions, your application must subscribe to receive the broadcast message when the index/unindex is completed. Typically, you should unsubscribe as soon as you get the result message

All status-returns are via `ALP_NOTIFY_EVENT_MS_INDEX_FILE` or `ALP_NOTIFY_EVENT_MS_UNINDEX_FILE` broadcast from `ALP_MEDIASELECTOR_APP_ID`. The details pointer points to a (variable-length) `AlpNotifyEventIndexFileResults` or `AlpNotifyEventUnindexFileResults` structure.

`alp_ms_cat_unindex_file` is much more efficient if the file exists when this function is called. If the file has already been deleted, `alp_ms_cat_unindex_file` falls back to calling `alp_ms_cat_update` on the lowest surviving 'leaf' directory - calling `alp_ms_cat_update` is more efficient.

## alp\_ms\_cat\_index\_file Function ^TOP^

### Purpose

Adds a single file to the catalog, and extracts its metadata.

### Prototype

```
alp_status_t alp_ms_cat_index_file (  
    const char *filename  
)
```

## alp\_ms\_cat\_unindex\_file Function ^TOP^

### Purpose

Removes a single file and its metadata from catalog.

### Prototype

```
alp_status_t alp_ms_cat_unindex_file (  
    const char *filename  
)
```

### Copying a File ^TOP^

Use the `alp_ms_cat_copy_file` function to copy files to and from different cards and the device. You can rename the copied file by providing a new file name in the destination path. It can call an optional progress callback

The function returns IPC errors. errors if `existing_filename` does not exist, if `new_filename` is illegal, or the optional callback returns a non-zero.

The function blocks until the Cataloger has copied the media file and its metadata. It does the copy in the calling process, then passes a request to `mediacatd` to copy the metadata (and adjust the copy's ownership and timestamps.) It calls the optional callback at least once (that is, it is always called on copy-completion, even if the source file is smaller than the copy buffer size) with filename and file size information: it is up to the calling application to map this to percent complete. It does not call the callback to signal metadata copy completion, but does blocks until it has copied the file and `mediacatd` has copied the metadata.

## alp\_ms\_cat\_copy\_file Function ^TOP^

### Purpose

Copies a single file and its metadata to a new location, optionally with a new filename.

### Prototype

```
alp_status_t alp_ms_cat_copy_file(  
    const char* new_filename,  
    const char* existing_filename,  
    alp_ms_cat_file_copy_callback callback,  
    void* data  
)
```

## alp\_ms\_cat\_file\_copy\_callback ^TOP^

## Purpose

Optional, progress reporting callback for `alp_ms_cat_file_copy_callback`.

## Prototype

```
typedef int (
    *alp_ms_cat_file_copy_callback
) (
    const char* destination_filename,
    const char* source_filename,
    int64_t bytes_copied,
    int64_t total_bytes,
    void* data
);
```

## Renaming a File <sup>^TOP^</sup>

Use the `alp_ms_cat_rename_file` function to rename a single file.

The function renames a single file and its metadata, optionally broadcasting an `ALP_NOTIFY_EVENT_MS_RENAME_FILE` event that contains an `AlpNotifyEventFileRenamed` structure with the original URL and the new filename.

The new\_filename does not need any path component. An extension is optional, but must match the existing\_url if present.

Use the `alp_mf_get_file_url` function to find the URL for the existing file.

The notification\_app provides the name of the notifying application. Use NULL or "" to specify no notification.)

The function returns IPC errors, and errors with the URLs or files.

## alp\_ms\_cat\_rename\_file Function <sup>^TOP^</sup>

### Purpose

Rename a single file and its metadata, optionally broadcasting an `ALP_NOTIFY_EVENT_MS_RENAME_FILE` event

that contains an `AlpNotifyEventFileRenamed` structure with the original URL and the new filename.

#### Prototype

```
alp_status_t alp_ms_cat_rename_file(  
    const char* new_filename,  
    const char* existing_url,  
    const char* notification_app  
)
```

#### Checking for Media Format Support <sup>^TOP^</sup>

Use the `alp_ms_cat_extension_supported` function to programmatically check to determine if a file type is recognized by the Cataloger. It does not determine if the actual media type is supported by the system in general (for example, for playback or display). However, it is likely that if type is recognized, it is also supported.

#### `alp_ms_cat_extension_supported` Function <sup>^TOP^</sup>

#### Purpose

Checks to see if the Media Cataloger indexes a certain media format.

#### Prototype

```
bool alp_ms_cat_extension_supported (  
    const char *extension  
)
```

{moscomment}