Chloromedia: A Standard Representation of Data for Media-Based Applications

http://brandonio21.com/chloromedia
August 18, 2016
Version 1.0

BACKGROUND

On devices, especially mobile devices, there are many apps which need to retrieve the media on a user's device. Each one of these apps, however, has its own implementation of reading the filesystem in order to aggregate all this media. Chloromedia aims to provide an insertable datalayer for these applications.

USE CASE

As an example, take LeafPic. LeafPic is an open-source media gallery app for Android devices. In LeafPic, the author painstakingly creates his own model to represent media on the device. Keeping his end-goal in mind, however, he creates a hectic data model which is intertwined with the gallery itself. Further extensions are near impossible. Further, LeafPic's model differs from all other gallery app models.

GOALS

Chloromedia aims to provide a clean, intuitive, and optimized solution to the "media application data layer" problem. It will accomplish these goals in the following way:

Clean	Follow modern Object Oriented Principles and Clean Coding principles
Intuitive	Have a minimalistic API with proper naming. Functions should be very small.
Optimized	By default, the API is load-on-demand and asynchronous.

NON-GOALS

Chloromedia aims to provide a clean media data-layer. It does not aim to manage the entire stack (it will not be a media gallery itself). Nor does Chloromedia provide a data-layer for anything beyond media. In short, Chloromedia will provide ways to access media and manipulate media without having to know about the implementation details of various media stores.

THE API - OVERVIEW

MediaProvider

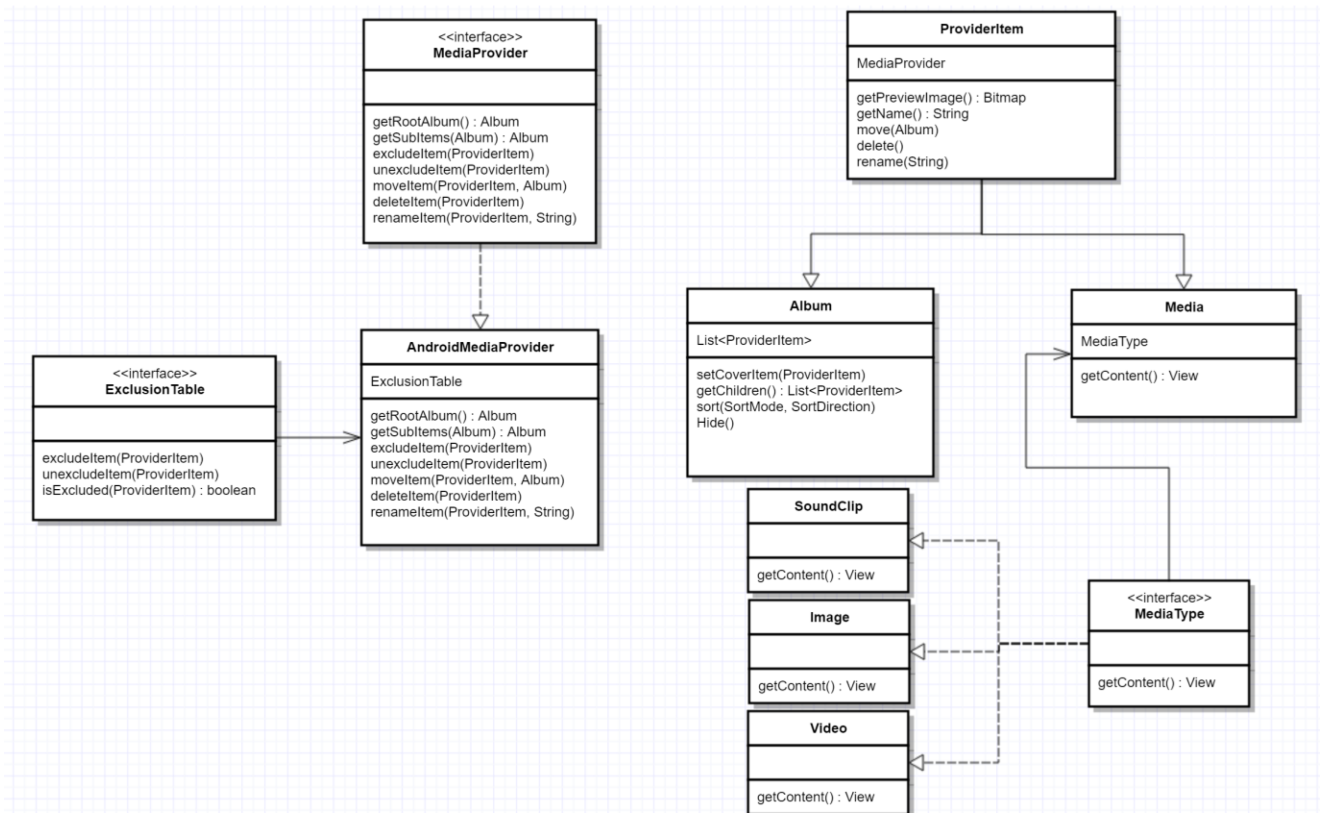
The media provider will, in essence, provide the media. It will connect to a source of media (a local filestore, a remote fileshare, etc.) and allow the user to retrieve media. Since the MediaProvider is the sole connection to the source of media, it will also house the functionality to interact with the media in file-related ways.

Keep in mind that as far as Chloromedia is concerned, media are not files. However, media do have the ability to be renamed, moved, and deleted.

Album
An album is a collection of media. Much like a photo album is used to store and organize photos, a Chloromedia album is used to keep track of and organize media. Of course, an album may also contain another album. Albums are not actionable – that is to say that they are simply a collection of other objects that may be sorted. One should not expect to "display" an album to a user. However, an album does have contents that may be displayed. An album may also have an image to represent itself.

Media
Media are the core of the API. They are videos, images, gifs, sound clips, etc. They are always contained within an album and may be acted upon. Each piece of media also has a type (video, image, gif, etc.). All media of the same type shall be displayed in the same manner.

THE API – UML



<<interface>>
MediaProvider
getRootAlbum() : Album
getSubItems(Album) : Album
excludeItem(ProviderItem)
unexcludeItem(ProviderItem)
moveItem(ProviderItem, Album)
deleteItem(ProviderItem)
renameItem(ProviderItem, String)
ProviderItem
MediaProvider
getPreviewImage() : Bitmap
getName() : String
move(Album)
delete()
rename(String)
<<interface>>
ExclusionTable
excludeItem(ProviderItem)
unexcludeItem(ProviderItem)
isExcluded(ProviderItem) : boolean
AndroidMediaProvider
ExclusionTable
getRootAlbum() : Album
getSubItems(Album) : Album
excludeItem(ProviderItem)
unexcludeItem(ProviderItem)
moveItem(ProviderItem, Album)
deleteItem(ProviderItem)
renameItem(ProviderItem, String)
Album
List<ProviderItem>
setCoverItem(ProviderItem)
getChildren() : List<ProviderItem>
sort(SortMode, SortDirection)
Hide()
Media
MediaType
getContent() : View
SoundClip
getContent() : View
Image
getContent() : View
Video
getContent() : View
<<interface>>
MediaType
getContent() : View

TECHNICAL SPECIFICATIONS

MediaProvider
All MediaProvider calls should be asynchronous. Thus, calls that are relayed to the ExclusionTable must also be asynchronous. All methods should return an AsyncTask which the user can then choose to execute at a later time. Further, the user may be able to display a loading spinner.

All AsyncTask methods of the Media Provider should update progress in terms of the items it is loading. For instance, getSubItems() should return, as its progress, a ProviderItem when it finishes loading. This will allow the user to more dynamically populate their user-interface.

getRootAlbum()

Shall first create a new Album object (the Root album). Shall then scan the top level of the file system asynchronously. When a folder is encountered, its second level is scanned until a valid media file is found. Upon which, the folder is added to the Root album and its cover image is set from the media file found.

If media files are found on the top level, they are also added to the Root album.

If no media files are found on the second level, repeated levels can be scanned for media files in a breadth-first search manner.

Note that files are determined to be media by their file extension, so no reading of file contents is necessary.

Finally, also note that all items should be checked against the exclusion table. If they are excluded (ExclusionTable.isExcluded shall be an O(1) operation), they shall not be added to the root album.

The root album should then be returned.

getSubItems(Album)

Shall use the same algorithm described in getRootAlbum() to find sub-items for the provided folder. Note that traversing to the provided album's location must be O(1). Therefore, if the filestore being used requires the ProviderItem to hold additional metadata, a custom *ProviderItemMetadata* implementation must be created.

moveItem(ProviderItem, Album)

Moves the provider item provided by the first argument into the album provided by the second argument. Do note that the second argument may be the Root album (which would move the item to the top level). This shall also move all of the children of the item if it is an album. All metadata shall be updated. Note that for all "file" operations, the parent of the specified item may need to also be altered.

deleteItem(ProviderItem)

Removes the specified item from the filesystem. If the specified item is an album, all children shall also be removed. Note that for all "file" operations, the parent of the specified item may need to also be altered.

renameItem(ProviderItem, String)

Renames the specified file on a filesystem level. If the specified file is an album, all children will be moved into the newly named Album. Note that for all "file" operations, the parent of the specified item may need to also be altered.

excludeItem(ProviderItem)

Shall add the item to the exclusion table, preventing it from being loaded. The parent of this object should be immediately notified.

unexcludeItem(ProviderItem)

Shall remove the item from the exclusion table, allowing it to be loaded. The parent of this object should be immediately notified.

ProviderItem

A ProviderItem is an item that was retrieved by the MediaProvider. A ProviderItem shall not be created and shall only be supplied by the MediaProvider. The ProviderItem holds a small amount of data that aids with its display as well as delegate methods for MediaProvider operations. The ProviderItem also contains ProviderItemMetadata, which allows the MediaProvider to better operate on it.

All ProviderItem operations should be O(1) (except those that delegate to other objects).

getPreviewImage()

Returns a Bitmap which corresponds to the image displayed as a preview. This should be an O(1) operation that should not need to consult the MediaProvider for more data.

getDate()

An O(1) operation to get the date representation of the item. This date is not strictly defined, but must be useful to the user. In most cases, this is the date of creation of the media.

getName()

Returns a display-friendly name which corresponds to the name displayed. This should be an O(1) operation that should not need to consult the MediaProvider for more data.

In most cases, the name returned can simply be the name provided by the MediaProvider. This name must be updated when the user renames the item.

getMetadata<T>(String)

Each ProviderItem should hold a map of {key: MetaData}, where the MetaData extends the ProviderItemMetadata class. This metadata should be used by the MediaProvider in order to perform its operations and should not be exposed to the user.

As an example, a java “File” object may be used as metadata for media loaded from the filesystem.

ProviderItemMetadata

The ProviderItemMetadata serves as a grouping of Metadata useful for a single type of MediaProvider. This object can be retrieved from a ProviderItem in order to perform MediaProvider operations on it.

Album : ProviderItem

The album is a collection of ProviderItems that also maintains some state. Along with containing the functionality of a ProviderItem, the album also allows the user to manipulate the state.

setCoverItem(ProviderItem)

Changes the preview image of the album to be the preview image of the provided item.

getChildren()

An asynchronous method that returns the children of the album. Note that the children of the album may not yet be loaded (Chloromedia defines a lazy-loaded interface).

If the media has not yet been loaded, the MediaProvider should be consulted. If the media has been loaded, it should simply be returned in an O(1) fashion.

getChildrenFromProvider()
Refreshes the list of children using the MediaProvider. This method is called by *getChildren()* if the contents have not yet been loaded, but may also be called by the end-user. (Imagine a pull-to-refresh). This method should simply invoke the MediaProvider's *getSubItems()*.

sort(SortMode, SortDirection)
Sorts the contents of the album and returns the contents. This should be done asynchronously. All subsequent calls to *getChildren()* and *getChildrenFromProvider()* will return data sorted in this fashion.

SortMode = {NAME, TYPE, DATE}; SortDirection = {ASCENDING, DESCENDING}

Media : ProviderItem
Media is an item provided by the MediaProvider that can also be directly viewed. A Media object has both a Preview image and a full representation that shall be loaded into its own "View".

getContentView()
Delegates to the *getView()* of its type.

MediaType
Since each type of media may be displayed differently, but all instances of a type of media should be displayed in the same fashion, the MediaType defines how a certain type of media should be displayed

getView(Media)
Returns a new view populated with operations pertaining to the Media. The MediaProvider may be consulted to read the entirety of the media.

The view must have operations pertaining to the media. For instance, if the Media is a Video, the View must contain a play button. If the media is an image, the View should be zoomable.