About This Article

DRM is a technology that enables content providers to distribute, promote and sell digital contents in a secure way. This document provides an introduction to DRM and Android DRM Framework APIs.

Scope:

This article is intended for Android developers wishing to develop mobile applications that use DRM API. It assumes good knowledge of Android and the Java programming language. This article focuses on DRM and therefore explaining Android technology is out of the scope of this documentation.

Introduction

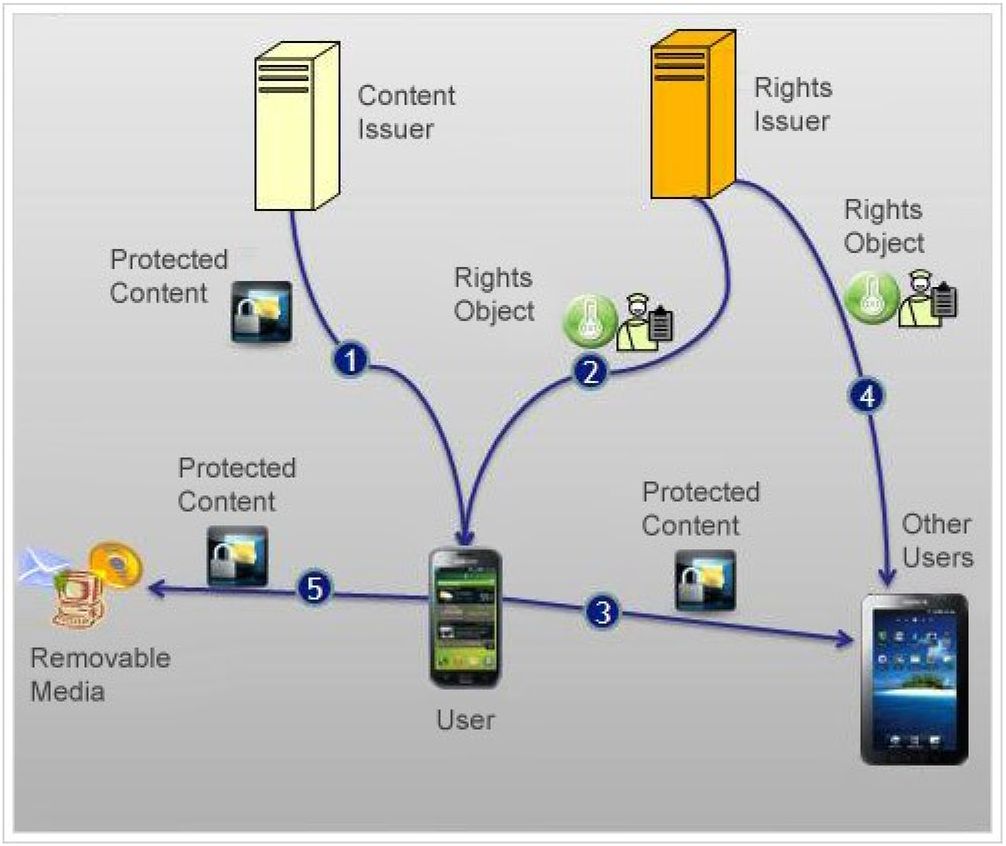
Digital Rights Management, or DRM is a technology that enables content providers to distribute, promote and sell digital contents in a secure way. DRM technology allows the content distributor to control the use of it's digital content (like music, ringtones, wallpapers etc) by preventing access, copying or conversion to other formats by end users.

DRM Content protection is achieved by defining different usage rights for the content DRM is applied to. DRM creates an essential foundation of trust between authors and consumers that is a prerequisite for robust market development.

DRM can help protect the interests of the content owner. DRM is not only useful for protecting content, but also selling and distributing content.

DRM

As shown in Figure 1, from the digital rights manager's point of view, the following functional entities have been identified



Screen 1: DRM Ecosystem

•DRM Agent
A DRM Agent embodies a trusted entity in a device. This trusted entity is responsible for enforcing permissions and constraints associated with DRM Content, controlling access to DRM Content, etc.
•Content Issuer
The content issuer is an entity that delivers DRM Content. OMA DRM defines the format of the DRM Content delivered to the DRM Agents, and the way the DRM Content can be transported from a content issuer to a DRM Agent using different transport mechanisms.
The content issuer may do the actual packaging of DRM Content itself, or it may receive pre-packaged content from some other source.
•Rights Issuer
The rights issuer is an entity that assigns permissions and constraints to DRM Content, and generates Rights Objects.

A Rights Object is an XML document expressing the permissions and constraints associated with a piece of DRM Content.

Rights Objects govern how DRM Content may be used – DRM Content cannot be used without an associated Rights Object, and may only be used as specified by the Rights Object.
•User
A user is the human user of DRM Content. Users can only access DRM Content through a DRM Agent.
•Off-device Storage

DRM Content is inherently secure, and may be stored by users off-device - for example in a network store, a PC, on removable media or similar.

This may be used for backup purposes, to free up memory in a device, and so on. Similarly, Rights Objects that only contain stateless permissions may be stored off-device.

OMA DRM

OMA DRM is an open digital rights management standard published by the Open Mobile Alliance. Most companies in the mobile industry, including many of the most popular operators and manufactures, take OMA DRM as their DRM standard.

Two OMA DRM standards have been released: OMA DRM 1.0 was released in September 2002 and OMA DRM 2.0 was published in March 2006.

OMA DRM 1.0

The OMA DRM 1.0 standard was released in September 2002 and is widely used in many mobile devices. It defines three application models:

- Forward-lock
- Combined Delivery
- Separate Delivery

Forward-lock

Forward Lock is frequently used for ring tones and wallpaper subscription and can effectively prevent illegal copying of files. In Forward Lock mode, the content is packaged and sent to the mobile terminal as a DRM message.

The mobile terminal could use the content, but could not forward it to other devices or modify it.

The file extension for a Forward Locked file is .dm, which includes the header and the encoded (but not the encrypted) content in it.

Combined Delivery

Combined Delivery is an extension of Forward Lock. In Combined Delivery mode, the digital rights are packaged with a content object in the DRM message.

The user can use the content as defined in the rights object, but cannot forward or modify it. The rights object defines the number of times and length of time that the content can be used thus enabling the preview feature.

The file extension for a Combined Delivery file is also .dm, which includes the header, the Rights Object and encoded content.

Separate Delivery

In the Separate Delivery mode, the content and rights are packaged and delivered separately. The content is encrypted into DRM Content Format (DCF) using a symmetric cryptograph method and can be transferred in an unsafe way such as Bluetooth, IrDA or via Email.

The Rights Object (RO) and the Content Encryption Key (CEK) are packaged and transferred in a safe way such as an unconfirmed Wireless Application Protocol (WAP) push. The terminal (mobile) is allowed to forward the content message but not the rights message.

Superdistribution is a Separate Delivery application which encourages digital content to be transferred freely and is typically distributed over public channels. But the content recipient has to contact the retailer to get the Rights object and CEK to use or preview the content.

The encrypted content file type extension is .dcf (DRM Content Format); the right file extension is .dr or .drc.

Defects in OMA DRM 1.0

The OMA DRM 1.0 model is designed for the mobile industry and is based on the assumption that the mobile terminal is reliable. In the Forward-lock mode and the Combined Delivery mode, the content is not encrypted. In the Separate Delivery mode, the symmetric encryption key is not encrypted. The media content can be stolen if the mobile terminal is hacked or the Right Object message with the CEK is revealed.

OMA DRM 2.0

With respect to the first version, OMA DRM 2 provides additional features and a significantly higher level of security, to protect high-value digital content like polyphonic ringtones, mp3 audio files or video clips on mobile terminals. Security in OMA DRM 2 is based on a Public Key Infrastructure (PKI) for key distribution and symmetric encryption algorithms for content protection. The OMA DRM 2 standard consists of three documents.

The DRM specification defines the communication protocol ROAP, content format for protected media files (Digital Content Format DCF) and the Rights Expression Language (REL), which describes permissions and constraints to govern usage of protected content.

Content and license information is delivered to the DRM Agent in two separate logical entities: Content Object (aka DCF although the acronym describes the file format rather than the file itself) and Rights Object (RO).

The DCF contains one or more containers that comprise encrypted digital content alongside descriptive meta-data such as author, title and a URL the user can visit in order to obtain a license that allows him/her to unlock the content. The Rights Object is realized as an XML file that describes permissions <play> </play> , <display> </display> , <execute> </execute> , <print> </print> , <export> </export> and constraints <count> </count> , <datetime> > granted to the DRM Agent when accessing a specific DCF. </datetime>

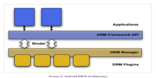
Android DRM Framework

The Android platform provides an extensible DRM framework that lets applications manage rights-protected content according to the license constraints that are associated with the content. The DRM framework supports many DRM schemes; which DRM schemes a device supports is up to the device manufacturer.

The Android DRM framework is implemented in two architectural layers as shown in Figure 2.

- •A DRM framework API, which is exposed to applications through the Android application framework and runs through the Dalvik VM for standard applications.
- •A native code DRM manager, which implements the DRM framework and exposes an interface for DRM plug-ins (agents) to handle rights management and decryption for various DRM schemes.

•



Screen 2: Android DRM Architecture

The Android DRM Framework is present in android.drm package. android.drm package consists of classes and interfaces as shown below in tables.

Interfaces	Description
DrmManagerClient.OnErrorListener	Interface definition for a callback that receives information about DRM framework errors.
DrmManagerClient.OnEventListener	Interface definition for a callback that receives information about DRM processing events.
DrmManagerClient.OnInfoListener	Interface definition for a callback that receives status messages and warnings during registration and rights acquisition.
DrmStore.ConstraintsColumns	Interface definition for the columns that represent DRM constraints.

Classes	Description
DrmConvertedStatus	An entity class that wraps converted data, conversion status, and the offset for appending the header and body signature to the converted data.
DrmErrorEvent	An entity class that is passed to the onError() callback.
DrmEvent	A base class that is used to send asynchronous event information from the DRM framework.
DrmInfo	An entity class that describes the information required to send transactions between a device and an online DRM server.
DrmInfoEvent	An entity class that is passed to the onInfo() callback.
DrmInfoRequest	An entity class that is used to pass information to an online DRM server.
DrmInfoStatus	An entity class that wraps the result of communication between a device and an online DRM server.
DrmManagerClient	The main programming interface for the DRM framework.
DrmRights	An entity class that wraps the license information retrieved from the online DRM server.
DrmStore	Defines constants that are used by the DRM framework.
DrmStore.Action	Defines actions that can be performed on rights-protected content.
DrmStore.DrmObjectType	Defines DRM object types.
DrmStore.Playback	Defines playback states for content.
DrmStore.RightsStatus	Defines status notifications for digital rights.
DrmSupportInfo	An entity class that wraps the capability of each DRM plug-in (agent), such as the MIME type and file suffix the DRM plug-in can handle.
DrmUtils	A utility class that provides operations for parsing extended metadata embedded in DRM constraint information.
DrmUtils.ExtendedMetadataParser	Utility that parses extended metadata embedded in DRM constraint information.
ProcessedData	An entity class that wraps the result of a processDrmInfo() transaction between a device and a DRM server.

Interfaces and classes present in android.drm package provide the following:

•Provides classes for managing DRM content and determining the capabilities of DRM Plug-ins.

•Determining which DRM plug-ins (agents) are installed on a device

•Retrieving information about specific plug-ins, such as the MIME types and file suffixes they support

- Retrieving license constraints for rights-protected content
- Checking whether a user has the proper rights to play or use rights-protected content.
- Associating rights-protected content with its license so you can use the Media Player API to play the content

DRM Framework APIs are exposed through DrmManagerClient class. Each DrmManagerClient gets an unique ID to operate with the DrmManager.

Although each DRM plug-in may require a different sequence of API calls, the general call sequence for an application is as follows:

Register the device with an online DRM service.

First use the acquireDrmInfo() method to acquire the registration information, and then use the processDrmInfo() method to process the registration information.

```
DrmManagerClient mDrmManager; mDrmManager = new DrmManagerClient(context);
```

Acquire the license that's associated with the rights-protected content.
First use the acquireDrmInfo() method to acquire the license information, and then using the processDrmInfo() method to process the license information. You can also use the acquireRights() method.

```
DrmInfoRequest drmInfoRequest; rightsAcquisitionInfo = new
DrmInfoRequest(DrmInfoRequest.TYPE_RIGHTS_ACQUISITION_INFO, MIME);
mDrmManager.acquireRights(drmInfoRequest);
```

Extract constraint information from the license.

You can use the getConstraints() method to do this.

```
ContentValues constraintsValues = mDrmManager.getConstraints(String path, int action)
ContentValues constraintsValues = mDrmManager.getConstraints(Uri uri, int action)
```

Associate the rights-protected content with its license.

You can use the saveRights() method to do this.

```
int successFailure = mDrmManager.saveRights(DrmRights  drmRights, String  rightsPath, String
contentPath)
```

After you make an association between the rights-protected content and its license, the DRM manager automatically handles rights management for that content.
Some of the other useful methods present in the android.drm packages are:

acquireDrmInfo

Retrieves information for registering, unregistering, or acquiring rights.

```
DrmInfo   drmInfo = mDrmManager.acquireDrmInfo (DrmInfoRequest  drmInfoRequest)
```

processDrmInfo

Processes the given DRM information based on the information type.

```
int successFailure = mDrmManager.processDrmInfo(DrmInfo  drmInfo)
```

canHandle

Checks whether the given MIME type or URI can be handled.

```
boolean canhandle = mDrmManager.canHandle(Uri  uri, String  mimeType) boolean canhandle =
mDrmManager.canHandle(String path, String mimeType)
```

checkRightsStatus

Checks whether the given rights-protected content has valid rights for the specified Action.

```
int status = mDrmManager.checkRightsStatus(filePath)
```

saveRights

Saves rights to a specified path and associates that path with the content path.

```
int successFailure =  mDrmManager.saveRights(DrmRights drmRights,String  rightsPath, String
contentPath)
```

removeRights

Removes all the rights information of every DRM plug-in (agent) associated with the DRM framework.

```
mDrmManager.removeAllRights() int  successFailure = mDrmManager.removeRights(String path)
int successFailure = mDrmManager.removeRights(Uri uri)
```

getAvailableDrmEngines

Retrieves information about all the DRM plug-ins (agents) that are registered with the DRM framework.

```
String[] engines = mDrmManager.getAvailableDrmEngines()
```

setOnErrorListener, setOnEventListener, setOnInfoListener

Registers a callback, which is invoked when the DRM framework sends error/event/Info notifications respectively.

```
mDrmManager.setOnEventListener(new DrmManagerClient.OnEventListener() {
        public void onEvent(DrmManagerClient client, DrmEvent event) {
                switch (event.getType()) {
                case DrmEvent.TYPE_DRM_INFO_PROCESSED:
                  //INFO PROCESSED
                break;
                }      }      });

mDrmManager.setOnErrorListener(new DrmManagerClient.OnErrorListener() {
            public void onError(DrmManagerClient client, DrmErrorEvent event) {
                switch (event.getType()) {
                 case DrmErrorEvent. TYPE_RIGHTS_NOT_INSTALLED:
                //RIGHTA NOT INSTALLED
                break;
                }

mDrmManager.setOnInfoListenr(new DrmManagerClient.OnInfoListener() {
public void onInfo(DrmManagerClient client, DrmInfoEvent event) {
               if (event.getType() == DrmInfoEvent.TYPE_RIGHTS_INSTALLED) {
                    //RIGHTS INSTALLED
                }     }      });
```
ref:http://developer.samsung.com/android/technical-docs/DRM-in-Android