RCS Device API 1.5 Specification
Version 2.0
16 October 2014

This is a Non-binding Permanent Reference Document of the GSMA

Security Classification: Non-confidential

Copyright Notice

Disclaimer

Antitrust Notice

Table of Contents

1 Introduction

1.1 Overview

This document defines the architecture and a set of standardized Application Programming Interfaces (API) to develop RCS user experience (UX), use RCS services and develop IP Multimedia Sub-system (IMS)-based services.

1.2 Scope

The scope of this document covers the APIs along with security limitations for the functionalities defined in [PRD RCC.60].

1.3 Definitions

Term	Description
3rd Party Applications	Applications that are not part of the RCS Client and developed by companies or individuals other than Mobile Network Operators (MNO) and Original Equipment Manufacturers (OEM).
Core Applications	Applications that are part of the RCS Client.
Trusted Applications	Applications using the IMS API, developed by trusted parties (MNOs and OEMs).
IMS Stack	Component responsible for implementing IMS protocol suite and core services.
RCS Client	Complete software package that passed RCS accreditation.
Service API	APIs that expose Standard Services and can be used in multiple instances without any restrictions.
Privileged Client API	API shall expose key functionalities which are necessary for the proper working of the RCS client.
IMS API	APIs that are exposed by the IMS Stack.
Standard Services	Services that are identified by feature tags, as defined by RCS Specification.

1.4 Abbreviations

Term	Description
AIDL	Android Interface Definition Language
API	Application Programming Interfaces
CD	Capability Discovery
CS	Circuit Switched
FT	File Transfer
ID	Identifier
IM	Instant Messaging
IMS	IP Multimedia Sub-system
IS	Image Share
MIME	Multipurpose Internet Mail Extensions

Term	Description
MNO	Mobile Network Operator
MSISDN	Mobile Subscriber Integrated Services Digital Network Number
MSRP	Message Session Relay Protocol
OEM	Original Equipment Manufacturer
OMA	Open Mobile Alliance
QCIF	Quarter Common Intermediate Format
RCS	Rich Communication Services
RTCP	Real-Time Control Protocol
RTP	Real-Time Protocol
SDK	Software Development Kit
SIMPLE	SIP (Session Initiation Protocol) Instant Message and Presence Leveraging Extensions
SIP	Session Initiation Protocol
URI	Uniform Resource Identifier
UX	User Experience

1.5 References

Ref	Doc Number	Title
[1]	[PRD RCC.60]	joyn Blackbird Product Definition Document http://www.gsma.com/rcs/wp-content/uploads/2013/10/Blackbird-Product-Descrption-Document-v2.0.pdf
[2]	[RFC 2119]	“Key words for use in RFCs to Indicate Requirement Levels”, S. Bradner, March 1997. Available at http://www.ietf.org/rfc/rfc2119.txt

1.6 Conventions

“The key words “must”, “must not”, “required”, “shall”, “shall not”, “should”, “should not”, “recommended”, “may”, and “optional” in this document are to be interpreted as described in [RFC 2119].”

2 API Architecture

2.1 Architecture Overview

The RCS Client architecture is composed of several sub-systems, organised into functional layers as shown in the diagram below.

The fundamental enabling component is the **IMS Stack** which contains the protocol suite (Session Initiation Protocol [SIP], Message Session Relay Protocol [MSRP], Real-Time Protocol [RTP]/Real-Time Control Protocol [RTCP], Hyper-Text Transfer Protocol [HTTP], etc.) and core services (IMS Session Management, Registration, etc.). The functionality of this component is governed by the IMS specifications.

Above IMS there are the **Rich Communication Services** (**RCS) Enablers**, comprising the functionality to enable RCS-based Chat, Video and Image sharing, File Transfer and other RCS services. The functionality of this layer is governed by the GSMA RCS specifications.

Access to these functional layers is mediated by **RCS Services API**. Client applications and services access the underlying functionality exclusively through this interface. The RCS service API logic access for client applications to the RCS services (Open Mobile Alliance [OMA] SIP Instant Message and Presence Leveraging Extensions [SIMPLE] Instant Messaging [IM], GSMA Video Share, GSMA Image Share, etc.).

The RCS **Core Applications or OEM UX** are the (typically embedded) applications that provide the end-user's access to RCS services. The Core Applications make use of the **RCS Services API** and also expose a **UX API** (a subset of the Service API) whereby any other applications can programmatically invoke operations that are interactively fulfilled by the Core Applications.

The architecture is intended to enable **RCS Extension** to make direct use also of the RCS Service API, enabling programmatic access to the RCS services. The RCS Service API is scoped so as to make access by Third Party Applications possible subject to those applications having the appropriate permission.

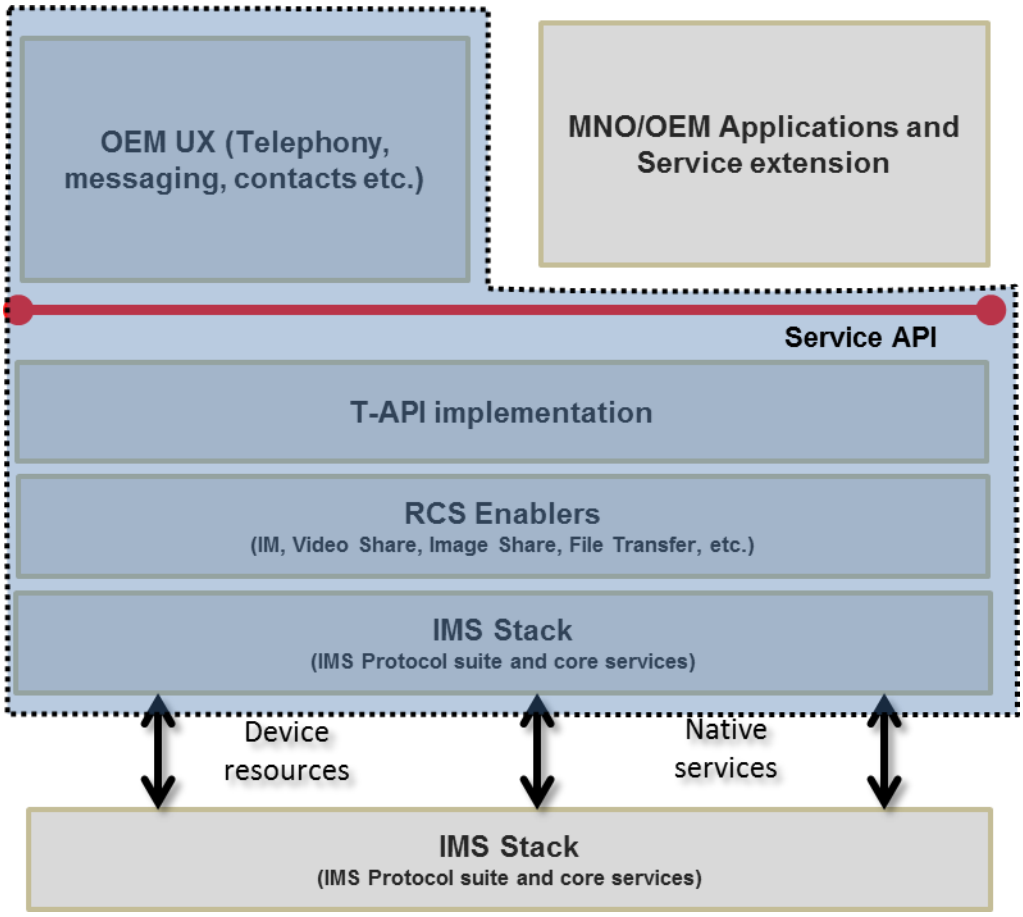

Figure 1: General Architecture Overview

2.1.1 API Descriptions

2.1.1.1 Service APIs

RCS Service APIs provide a functional interface to the RCS enablers, enabling the Core Applications and Third Party Applications to interoperate with other RCS devices whilst relying on the stack to ensure conformance to the RCS Specification.

There are two types of service APIs:

- The UX API which is a high level API enabling other installed applications to link to the native RCS services or applications.
- The core service APIs offering lower level APIs for the following
    - Capabilities service API
    - Chat service API
    - File Transfer service API
    - Video Share service API
    - Image Share service API
    - Geoloc Share service API
    - History service API
    - MultiMedia Session service API
    - File Upload API
    - Client Connector

NOTE: For Video Share and Image Share functionality to be fully available, a call needs to be ongoing with a RCS contact posessing Video Share and Image Share capabilities.

Each service API is based on a Client/Server model using the Android Interface Definition Language (AIDL) Android interface to communicate between the application using the service and the RCS service or stack implementing the service. So many applications can connect in parallel to the core RCS service.

The APIs in this layer also expose common RCS functionality for capability fetching and retrieving parts of the RCS network configuration required for UI elements.

In case of Android OS, client enables to interface the native RCS service functionality with 3rd party applications on the device.

2.1.2 Applications Types

Applications types can be divided into three broad categories:

- OEM applications
- MNO applications
- Third party applications

An application may use:

- RCS Extensions to provide service-over-service functionality. These applications use additional parameterization defined in the Service APIs, and may have their own feature tags not defined by the RCS Specification.
- Core Services that are included in the RCS Client. These components must undergo GSMA accreditation as part of the RCS Client. Core Applications may use Service APIs (such as IM) and can have overlapping functionality with RCS Extensions.

3 API concepts

3.1 Servers and Listeners

RCS APIs are provided with a client/server model. At any time, for a service, there may be zero or more clients. At any time, a client may be connected to zero or more services.

Prior to requesting a service, a client connects to that service.

Servers provide RCS services to the clients and notify the registered clients with the events through listeners. Clients request RCS services from the servers by invoking the appropriate API(s). Servers notify clients of RCS events by invoking the appropriate listener (callback functions). For RCS events that a client is required to monitor, the client must supply the listener to the server.

For each service, this document describes all server APIs as well as the set of events that are available for that service.

3.1.1 Service

Prior to using a service, a RCS client invokes the appropriate API to create the service. At this time, the client can also register for events by supplying the appropriate listener functions.

Once the service is created, the service communicates/notifies its clients about the service availability and/or service-specific functionality changes through the listener supplied by the clients.

At any time, a service may have zero or more sessions associated with it.

When a service is no longer needed, the client can destroy the service by invoking the appropriate API for that service. When a service is destroyed, all the service sessions associated with that service are also terminated.

3.1.2 Service Session

A service session is established based on external triggers, e.g. a user attempting to establish a call or upon receipt of an event from the RCS service about a request from a remote user. When a service session is to be established, the appropriate API is invoked. At the time of establishment of a service session, the client registers for events by supplying appropriate listener functions. Each service session is associated with a RCS service.

After the service session is created, the RCS service communicates/notifies its clients about the session state through the listeners supplied by the clients.

At any time, the client can terminate a service session by invoking the appropriate API, for example, based on user action or based on events from the RCS server that indicate a change in the session state.

3.2 Service Version/Available/Unavailable

Each service is associated with a specific client. Services are designed to allow each service to have its own service version and its availability/unavailability attribute independently. Each service follows the same template API to provide versioning information and has the same type of listener functions through which the service informs its clients about specific service status.

4 Android API

See also a detailed Javadoc of the Android API from the RCJTA web site (https://code.google.com/p/rcsjta/).

4.1 Components Interaction

Each of the Terminal APIs for Android defines their interaction individually and how they can be used by an Android Application.

4.1.1 New service application

When an Android application wants to define a new service, it needs to add its feature tag as meta-data value in its Android Application Manifest. The RCS Service Tag also needs to be accompanied by the feature tag. Refer to [PRD RCC.60].

4.1.2 Constraints

Following constraints apply:

1. Only a single RCS Stack can be active on a device. This constraint limits the possibilities for deployment of additional RCS Stacks with the Terminal API, as they cannot replace the package of a previously installed stack on the device. This constraint could be avoided if the Terminal APIs could be retrieved dynamically instead of static package reference.
2. When multiple applications are present, that support the same type of service notifications, multiple notification may be placed in the Notification Tray, if each application handles the broadcasted intent.
3. Trusted application can only run if any IMS Stack is running. This means trusted applications can only be dynamically registered/de-registered. They are not allowed to be part of initial registration application set. Any exceptions need to be carefully considered.

4.2 Security

Most of the RCS APIs provide access to sensitive functionality, either because they enable access to privacy-sensitive information or because they can cause charges to be incurred for network and service usage. In addition, certain APIs expose the internal functionality of the stack, and abuse of those APIs could compromise the integrity of the stack or the RCS services.

4.2.1 Service API Access Control

The Service APIs are sensitive and their abuse could compromise the integrity of the stack or the RCS services. Access is therefore restricted so that they may only be used by authorised RCS Extensions, through OEM signing, embedding in system folder, or another solution mutually agreed between MNO and OEM.

Where the Service API exposes privacy-sensitive information or may trigger service charges, the user must grant permission for any application to use that API. The permissions are defined for each service API in the sections that follow.

4.3 UX API

This API offers:

- Intents which permit to link RCS applications with other third party applications installed on the device.
- Methods to discover existing RCS services on the device and their activation states.

4.3.1 Package

Package name **com.gsma.services.rcs**

4.3.2 Methods and Callbacks

Class **RcsUtils**:

- Method: returns the list of RCS services installed on the device (except myself). An application is identified as a RCS service by including an intent filter with the ACTION_VIEW _SETTINGS action in the Manifest.xml of the application(eg. Android service).

```
  <intent-filter>
    <category android:name="android.intent.category.LAUNCHER"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <action
android:name="com.gsma.services.rcs.action.VIEW_SETTINGS"/>
  </intent-filter>

static List<ResolveInfo> getRcsServices(Context context)
```

- Method: detects if a particular RCS service is activated. The result is returned asynchronously via a broadcast receiver. The RCS service is identified by the object ResolveInfo recovered via the method getRcsServices . Each RCS service should implement a Broadcast receiver with the following Intent filter in order to return its current status:

```
  <intent-filter>
  <action android:name="<service package name>.service
.action.GET_STATUS"/>
  </intent-filter>
```

- The action should start with the specific service package name and should terminate with ACTION_GET_STATUS.

```
static void isRcsServiceActivated(Context ctx, ResolveInfo appInfo,
BroadcastReceiver receiverResult)
```

- Method: Load the settings activity of a particular RCS service. The RCS service activity is identified by the object ResolveInfo recovered via the method getRcsServices .

```
static void loadRcsServiceSettings(Context context, ResolveInfo
appInfo)
```

4.3.3 Intents

Class **Intents.Service**:
This class offers intents to link applications to the RCS service.

- Intent: load the settings activity to enable or disable the RCS service.

```
static final String ACTION_VIEW_SETTINGS =
“com.gsma.services.rcs.action.VIEW_SETTINGS”;
```

- Intent: get the RCS service status.

```
static final String ACTION_SERVICE_GET_STATUS =
“.service.action.GET_STATUS”;
```

This Intent contains the following extras:

```
“service”: (String) name of the service.
“status”: (boolean) status of the service.
```

Class **Intents.Chat**:
This class offers Intents to link applications to RCS applications for chat services.

- Intent: load the chat application to view a chat conversation. This Intent takes into parameter a Uniform Resource Identifier (URI) on the chat conversation (i.e. content://chats/chat_ID). If no parameter found the main entry of the chat application is displayed.

```
static final String ACTION_VIEW_ONE_TO_ONE_CHAT =
"com.gsma.services.rcs.action.VIEW_ONE_TO_ONE_CHAT"
```

This Intent contains the following extra:

```
“uri”: (android.net.Uri) uri of the one to one chat conversation.
```

- Intent: load the chat application to send a new chat message to a given contact. This Intent takes into parameter a contact URI (i.e. content://contacts/people/contact_ID). If no parameter the main entry of the chat application is displayed.

```
static final String ACTION_SEND_ONE_TO_ONE_CHAT_MESSAGE =
"com.gsma.services.rcs.action.SEND_ONE_TO_ONE_CHAT_MESSAGE"
```

This Intent contains the following extra:

```
"uri": (android.net.Uri) uri of the contact.
```

- Intent: load the group chat application. This Intent takes into parameter an URI on the group chat conversation (i.e. content://chats/chat_ID). If no parameter is found the main entry of the group chat application is displayed.

  ```
  static final String ACTION_VIEW_GROUP_CHAT =
  "com.gsma.services.rcs.action.VIEW_GROUP_CHAT"
  ```

  This Intent contains the following extra:

  ```
  "uri": (android.net.Uri) uri of the group chat conversation.
  ```

- Intent: load the group chat application to start a new conversation with a group of contacts. This Intent takes into parameter a list of contact URIs (i.e. content://contacts/people/contact_ID). If no parameter, the main entry of the group chat application is displayed.

  ```
  static final String ACTION_INITIATE_GROUP_CHAT =
  "com.gsma.services.rcs.action.INITIATE_GROUP_CHAT"
  ```

  This Intent contains the following extra:

  ```
  "uris": (List<android.net.Uri>) List of uris of the contacts.
  ```

Class **Intents.FileTransfer**:

This class offers Intents to link applications to RCS applications for file transfer services.

- Intent: load the file transfer application to view a file transfer. This Intent takes into parameter a URI on the file transfer (i.e. content://filetransfers/ft_ID). If no parameter is found, the main entry of the file transfer application is displayed.

  ```
  static final String ACTION_VIEW_FILE_TRANSFER =
  "com.gsma.services.rcs.action.VIEW_FILE_TRANSFER"
  ```

  This Intent contains the following extra:

  ```
  "uri": (android.net.Uri) uri of the file transfer.
  ```

- Intent: load the file transfer application to start a new file transfer to a given contact. This Intent takes into parameter a contact URI (i.e. content://contacts/people/contact_ID). If no parameter, the main entry of the file transfer application is displayed.

  ```
  static final String ACTION_INITIATE_ONE_TO_ONE_FT =
  "com.gsma.services.rcs.action.INITIATE_ONE_TO_ONE_FT"
  ```

  This Intent contains the following extra:

  ```
  "uri": (android.net.Uri) uri of the contact.
  ```

- Intent: load the group chat application to start a new conversation with a group of contacts and send a file to them. This Intent takes into parameter a list of contact URIs (i.e. content://contacts/people/contact_ID). If no parameter, the main entry of the group chat application is displayed.

```
static final String ACTION_INITIATE_GROUP_FILE_TRANSFER =
"com.gsma.services.rcs.action.INITIATE_GROUP_FILE_TRANSFER"
```

This Intent contains the following extra:

```
"uris": (List<android.net.Uri>) List of uris of the contacts.
```

NOTE: for Intents using a contact URI as a parameter, if the contact has several phone numbers which are RCS compliant, then the application receiving the Intent should request to the user to select which phone number should be used by the service.

NOTE: sharing during a call (image & video) are part of the native dialler application and may be only visible when a call is established, in this case there is no public Intent to initiate a sharing.

4.4 Services API

4.4.1 Overview

This section contains all the Service APIs. Each of the presented APIs may have a Core Application using it, but a separate 3rd Party Application can also use it. Each API exposes all its functionality on a high level and does put constraints on the invoking application as to the preconditions and order of method calls. All Service APIs are stateless, meaning that any part of the API can be used without first satisfying any preconditions.

4.4.2 Access Control

Each of the services requires one or more permissions to be held by the calling application; the permissions associated by each service are defined in the sections that follow.

The permissions are organised on a service-by-service basis and at a sufficiently fine-grained level – e.g. the ability to read contact details from the address book - that the user can make a meaningful choice when confronted with a request at the install prompt. The user is not asked to give blanket approval to a very broad permission such as the ability to read any user data.

4.4.3 Common architecture

The RCS terminal API contains the following service API:

- Capability service API
- Chat API
- File Transfer API
- Video Share service API
- Image Share service API
- Geoloc Share service API
- History service API

- Multimedia Session service API
- File Upload API

Each service API is based on a Client/Server model using the Android Interface Definition Language (AIDL) Android interface to communicate between the application using the service and the RCS service or stack implementing the service. So many applications can connect in parallel to the core RCS service.

4.4.3.1 Package

Package name **com.gsma.services.rcs**

4.4.3.2 Methods and Callbacks

Class **RcsService**:

Each service API should extend the abstract class RcsService.

- Enum: directions of a chat message, geolocation, filetransfer, imageshare, videoshare etc..

```
enum Direction { INCOMING(0), OUTGOING(1), IRRELEVANT(2) }
```

- Enum: Read status of a chat message or a file transfer.

```
enum ReadStatus { UNREAD(0), READ(1) }
```

- Constructor: instantiates a service API. This method takes in parameter a service event listener which permits to monitor the connection to the RCS service. The parameter context is an Android context which permits to initiate the binding with the corresponding service.

```
RcsService(Context ctx, RcsServiceListener listener)
```

- Method: connects to the API. This method permits to bind to the service.

```
void connect()
```

- Method: disconnects from the API. This method permits to unbind from the service.

```
void disconnect()
```

- Method: returns “true” if connected to the service, else returns “false”.

```
boolean isServiceConnected()
```

- Method: returns true if service registered to the RCS service platform, else returns false.

```
boolean isServiceRegistered()
```

- Method: adds a listener on service registration event.

```
void addEventListener(RcsServiceRegistrationListener listener)
```

- Method: removes a listener on service registration event.

```
void removeEventListener(RcsServiceRegistrationListener listener)
```

- Method: returns the version of the service (see constants from class RcsService.Build.VERSION_CODES).

```
int getServiceVersion()
```

Interface **RcsServiceListener**:

- Method: callback called when service is connected. This method is called when the service is well connected to the RCS service (binding procedure successful): this means the methods of the API may be used.

```
void onServiceConnected()
```

- Method: callback called when service has been disconnected. This method is called when the service is disconnected from the RCS service (e.g. service deactivated).

```
void onServiceDisconnected(ReasonCode reasonCode)
```

- Enum: the reason code of the service disconnection.

```
enum ReasonCode { INTERNAL_ERROR(0), SERVICE_DISABLED(1),
CONNECTION_LOST(2) }
```

Class **RcsServiceRegistrationListener**:

- Method: callback called when a service is registered to the RCS platform. This method is called when the terminal is registered to the RCS/IMS service platform.

```
void onServiceRegistered()
```

- Method: callback called when a service is unregistered from RCS platform. This method is called when the terminal is not registered to the RCS service platform.

```
void onServiceUnregistered()
```

Class **RcsServiceConfiguration:**

This class represents the particular configuration of RCS Service.

- Enum: the messaging client mode.

```
enum MessagingMode { NONE(0), INTEGRATED(1), CONVERGED(2),
SEAMLESS(3) }
```

- Enum: the messaging method.

```
enum MessagingMethod { AUTOMATIC(0), RCS(1), NON_RCS(2) }
```

- Method: returns True if the RCS service is activated, else returns False. The service may be activated or deactivated by the end user via the RCS settings application.

```
static boolean isServiceActivated(Context ctx)
```

- Method: returns the display name associated to the RCS user account. The display name may be updated by the end user via the RCS settings application.

```
static String getMyDisplayName(Context ctx)
```

- Method: set the display name associated to the RCS user account.

```
static void setMyDisplayName(Context ctx, String name)
```

- Method: returns the user contact identifior (i.e. username part of the IMPU).

```
static ContactId getMyContactId(Context ctx)
```

- Method: returns the user Country Code.

```
static String getMyCountryCode(Context ctx)
```

- Method: returns the user Country Area Code.

```
static String getMyCountryAreaCode(Context ctx)
```

- Method: returns “true” if RCS configuration is valid

```
static boolean isConfigValid(Context ctx)
```

- Method: returns the messaging client mode.

```
static MessagingMode getMessagingUX(Context ctx)
```

- Method: returns the default messaging method.

```
static MessagingMethod getDefaultMessagingMethod(Context ctx)
```

- Method: set the default messaging method.

```
static void setDefaultMessagingMethod(Context ctx, MessagingMethod
method)
```

4.4.3.3 Common Content Providers

A content provider is used to store the RCS service configuration persistently.

Class **RcsServiceConfiguration:**

URI to access the provider data:

```
static final Uri CONTENT_URI =
"content://com.gsma.services.rcs.provider.setting/setting"
```

The “KEY” column below is defined as the unique primary key and can be referenced with adding a path segment to the CONTENT_URI + “/” + <primary key>

Column name definition constants to be used when accessing this provider:

```
static final String KEY = “key”
static final String VALUE = “value”
```

The content provider has the following table and columns:

SETTING

Data	Data Type	Description
KEY	String (primary key)	Key of the Rcs configuration parameter
VALUE	String	Value of the Rcs configuration parameter

Possible values for the KEY fields:

- MY_DISPLAY_NAME,
- MY_CONTACT_ID,
- MY_COUNTRY_CODE,
- MY_COUNTRY_AREA_CODE,
- CONFIGURATION_VALIDITY,
- MESSAGING_MODE,
- DEFAULT_MESSAGING_METHOD

Class **GroupDeliveryInfoLog**:

URI constant to be able to query the provider data (Note that only read operations are supported since exposing write operations would conflict with the fact that the stack is performing write operations internally to keep the data matching the current situation):

```
static final Uri CONTENT_URI =
"content://com.gsma.services.rcs.provider.groupdeliveryinfo/groupdeliveryin
fo"
```

The “ID” column together with the “CONTACT” column below is defined as the unique primary key * but can’t be referenced by adding a path segment to the CONTENT_URI.

Column name definition constants to be used when accessing this provider:

```
static final String ID = "id"
static final String CONTACT = "contact"
static final String CHAT_ID = "chat_id"
static final String TIMESTAMP_DELIVERED = "timestamp_delivered"
static final String TIMESTAMP_DISPLAYED = "timestamp_displayed"
static final String STATUS = "status"
static final String REASON_CODE = "reason_code"
```

The content provider (common to both group chat messages and group file transfers) has the following columns:

GROUPDELIVERYINFO

Data	Data Type	Description
ID	String (part of primary key*)	Unique Id of the chat message (“msg_id”) or file transfer (“ft_id”)
CONTACT	String (part of primary key*)	ContactId formatted number of the remote contact of the group chat message or the group file transfer
CHAT_ID	String (not null)	Id of chat room
TIMESTAMP_DELIVERED	Long	Time when message delivered. If 0 means not delivered.
TIMESTAMP_DISPLAYED	Long	Time when message displayed. If 0 means not displayed.
STATUS	Integer	See enum GroupDeliveryInfo.Status for the list of statuses.
REASON_CODE	Integer	See enum GroupDeliveryInfo.ReasonCode for the list of reason codes.

- Enum: states associated with the group delivery info provider.

```
enum Status { UNSUPPORTED(0), NOT_DELIVERED(1), DELIVERED(2),
DISPLAYED(3), FAILED(4) }
```

- Enum: reason code associated with the group delivery info provider.

```
enum ReasonCode { UNSPECIFIED(0), FAILED_DELIVERY(1), FAILED_DISPLAY(2)
}
```

4.4.3.4 Exceptions

Class **RcsServiceException**:

This generic class must be thrown from a service API when the requested operation failed to fully complete its scope of responsibility and none of the more specified exceptions below can be thrown. This exception is not to be defined as an abstract exception neither are any of the more specific exceptions below intended to extend this exception. The client must be able to trust that in case of any failure whatsoever, and none of the more specific exceptions below are thrown, this exception will be thrown as a kind of default exception to signify that some error occurred that does not necessarily need to be more specific than that.

Class **RcsServiceNotAvailableException**:

This class is thrown when a method of the service API is called and the service API is not bound to the RCS service (e.g. RCS service not yet started or API not yet connected).

Class **RcsServiceNotRegisteredException**:

This class is thrown when a method of the service API using the RCS service platform is called and the terminal which requires that the RcsCoreService is registered and connected to the IMS server like for instance initiateGroupChat(,,,) is not registered to the RCS service

platform (e.g. not yet registered) It is not thrown when a service API method is called that could fully perform its scope of responsibility without having to be connected to the IMS, like for instance calling getConfiguration() on a service.

Class **RcsContactFormatException (RuntimeException)**:

This class is thrown when the specified contact format String is not supported or not well formatted.

Class **RcsMaxAllowedSessionLimitReachedException**:

This class is thrown if the message/filetransfer/imageshare/geolocationshare etc (all the types) cannot be sent/transferred/resent or a new groupchat invitation cannot be sent right now since the limit of allowed ongoing sessions has already been reached and the client needs to wait for at least one session to be released back to the stack first.

Class **RcsPermissionDeniedException**:

This class is thrown when a method of the service API is called that is not allowed right now. This can be for multiple reasons like it is not possible to call accept() on a file transfer invitation that has previously already been rejected, the file trying to be sent is not allowed to be read back due to security aspects or any other operation that fails because the operation is not allowed or has been blocked for some other reason.

Class **RcsPersistentStorageException**:

This class is thrown when a method of the service API is called to persist data or read back persisted data failed. This can be because the underlying persistent storage database (or possibly further on a CPM cloud) reported an error such as no more entries can be added perhaps because disk is full, or just that a SQL operation failed or even a unsuccessful read operation from persistent storage.

Class **RcsUnsupportedOperationException (RuntimeException)**:

This class is thrown when a method of the service API is called that is not supported (i.e. does not make sense within the scope of the use case) like trying to call pauseTransfer() on a non pausable file transfer that does not support that, etc.

Class **RcsIllegalArgumentException (RuntimeException)**:

This class is thrown when a method of the service API is called with one or multiple illegal input parameters. Such as calling a method and passing null as a parameter in the case that null is not valid for that parameter or a file uri that does not point to any existing file or a file that is bigger than max size limit or a group chat id that must not refer to a non existing group chat unless that is specifically otherwise specified in the method description etc.

NOTE: For more detailed information about exactly which method call in the API can throw which exceptions above see the javadoc

4.4.3.5 Permissions

Access to the Services API requires the com.gsma.services.rcs.READ_RCS_STATE permission. This is a new permission, analogous to READ_PHONE_STATE, covering general access to the RCS stack state.

This permission is additionally required to access any of the specific services, since use of those services implicitly reveals information about the current network and stack state

4.4.3.6 Intents

Intent broadcasted when the service is up.

```
com.gsma.services.rcs.action.SERVICE_UP
```

Intent broadcasted when the service has been provisioned with success and the service may connect to the service platform.

```
com.gsma.services.rcs.action.SERVICE_PROVISIONED
```

4.4.4 Capability API

This API allows for querying the capabilities of a user or users and checking for changes in their capabilities:

- Read the supported capabilities locally by the user on its device.
- Retrieve all capabilities of a user.
- Checking a specific capability of a user.
- Refresh capabilities for all contacts.
- Registering for changes to a user/users ‘s capabilities
- Unregistering for changes to a user/users ‘s capabilities
- Define scheme for registering new service capabilities based on manifest defined feature tags.

This API may be accessible by any application (third party, MNO, OEM). The RCS extensions are controlled internally by the RCS service.

Note: there is the same API between File transfer and File Transfer over HTTP. So from an API perspective there is the same capability for both mode (MSRP and HTTP) and it is transparent for the user.

4.4.4.1 Capability Discovery API calling flow

The Capability Discovery (CD) service provides the API through which the user can get the capabilities of other contacts and also "announce" its own capabilities.

The figures in this section contains basic call flows of the CD service API.

The following is an example that shows the retrieval of the capabilities of a list of remote contacts.

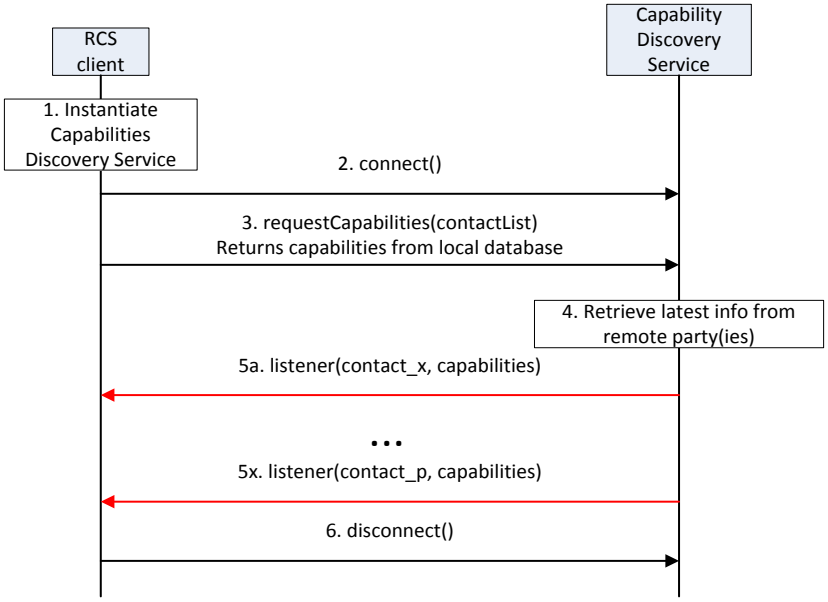

Figure 2: Get the capabilities of a list of remote contacts

1. The RCS client instantiates a service instance of the Capability Discovery Service. At this time, it also specifies the list of listener functions.
2. The RCS client establishes a connection with the Capability Discovery Service. The Capability Discovery Service associates the listener with this RCS client.
3. The RCS client constructs a list of contacts for which it wants to get the latest capabilities. It invokes the API to get the capabilities of these contacts by providing the contact list as parameter. The Capability Discovery Service returns the requested information from the local database.
4. Additionally, the Capability Discovery Service initiates procedures with the remote parties to retrieve the latest capabilities.
5. When the updated capability information is available for a contact, the listener function(s) are invoked to inform all the RCS clients that have installed a listener. This step is repeated for each contact for which updated capability information becomes available.
6. Finally, the RCS client, having retrieved the contact information, disconnects from the capability discovery service. At this time, the Capability Service discards all listeners associated with this client.

4.4.4.2 Package

Package name **com.gsma.services.rcs.capability**

4.4.4.3 Methods and Callbacks

Class **CapabilityService**:

This class offers the main entry point to the Capability service which permits to read capabilities of remote contacts, to initiate capability discovery and to receive capabilities updates. Several applications may connect/disconnect to the API.

A set of capabilities is associated to each MSISDN of a contact.

- Method: connects to the API.

```
void connect()
```

- Method: disconnects from the API.

```
void disconnect()
```

- Method: returns the capabilities supported by the local end user. The supported capabilities are fixed by the MNO and read during the provisioning.

```
Capabilities getMyCapabilities()
```

- Method: returns the capabilities of a given contact from the local database. This method doesn't request any network update to the remote contact. The parameter contactId supports the following formats: MSISDN in national or international format, SIP address, SIP-URI or Tel-URI. If no matching contact capabilities are found then null is returned.

```
Capabilities getContactCapabilities(ContactId contact)
```

- Method: requests capabilities to a remote contact. This method initiates in the background a new capability request to the remote contact by sending a SIP OPTIONS. The result of the capability request is sent asynchronously via callback method of the capabilities listener. A capability refresh is only sent if the timestamp associated to the capability has expired (the expiration value is fixed via MNO provisioning). The parameter contactId supports the following formats: MSISDN in national or international format, SIP address, SIP-URI or Tel-URI. If the format of the contact is not supported an exception is thrown. The result of the capability refresh request is provided to all the clients that have registered the listener for this event.

```
void requestContactCapabilities(ContactId contact)
```

- Method: requests capabilities for a group of remote contacts. This method initiates in the background new capability requests to the remote contact by sending a SIP OPTIONS. The result of the capability request is sent asynchronously via callback method of the capabilities listener. A capability refresh is only sent if the timestamp associated to the capability has expired (the expiration value is fixed via MNO provisioning). The parameter contact supports the following formats: MSISDN in national or international format, SIP address, SIP-URI or Tel-URI. If the format of the contact is not supported an exception is thrown. The result of the capability refresh request is provided to all the clients that have registered the listener for this event.

```
void requestContactCapabilities(Set<ContactId> contacts)
```

- Method: requests capabilities for all contacts existing in the local address book. This method initiates in the background new capability requests for each contact of the address book by sending SIP OPTIONS. The result of a capability request is sent asynchronously via callback method of the capabilities listener. A capability refresh is only sent if the timestamp associated to the capability has expired (the expiration value is fixed via MNO provisioning). The result of the capability refresh request is provided to all the clients that have registered the listener for this event.

```
void requestAllContactsCapabilities()
```

- Method: registers a listener for receiving capabilities on any contact.

```
void addCapabilitiesListener(CapabilitiesListener listener)
```

- Method: unregisters a capabilities listener.

```
void removeCapabilitiesListener(CapabilitiesListener listener)
```

- Method: registers a capabilities listener for receiving capabilities on a list of contacts.

```
void addCapabilitiesListener(Set<ContactId> contacts,
CapabilitiesListener listener)
```

- Method: unregisters a capabilities listener on a list of contacts.

```
void removeCapabilitiesListener(Set<ContactId> contacts,
CapabilitiesListener listener)
```

Class **CapabilitiesListener**:

This class offers callback methods for the listener of capabilities.

- Method: callback called when new capabilities are received for a given contact. The first argument `contact` contains the canonical representation of the identity of the contact whose capabilities are indicated by the second argument capabilities

```
void onCapabilitiesReceived(ContactId contact, Capabilities
capabilities)
```

Class **Capabilities**:

This class encapsulates the different capabilities which may be supported by the local user or a remote contact.

- Method: returns true if the file transfer is supported, else returns false

```
boolean isFileTransferSupported()
```

- Method: returns true if IM/Chat is supported, else returns false

```
boolean isImSessionSupported()
```

- Method: returns true if image sharing is supported, else returns false

```
boolean isImageSharingSupported()
```

- Method: returns true if video sharing is supported, else returns false

```
boolean isVideoSharingSupported()
```

- Method: returns true if geoloc push is supported, else returns false

```
boolean isGeolocPushSupported()
```

- Method: returns true if the specified feature tag is supported, else returns false. The parameter tag represents the feature tag to be tested.

```
boolean isExtensionSupported(String tag)
```

- Method: returns the list of supported RCS extensions

```
Set<String> getSupportedExtensions()
```

- Method: returns true if it's an automata, else returns false

```
boolean isAutomata()
```

- Method: returns the timestamp of the last capability refresh.

```
long getTimestamp()
```

- Method: returns true if the capability is valid (no need to refresh it), else returns false.

```
boolean isValid()
```

4.4.4.4 Content Providers

A content provider is used to store locally the capabilities of each remote contact. In this case the capabilities may be read even if there is no connection to the RCS platform. There is one entry per remote MSISDN Number.

Class **CapabilitiesLog:**

URI constant to be able to query the provider data (Note that only read operations are supported since exposing write operations would conflict with the fact that the stack is performing write operations internally to keep the data matching the current situation):

```
static final Uri CONTENT_URI =
"content://com.gsma.services.rcs.provider.capability/capability"
```

The "CONTACT" column below is defined as the unique primary key and can be references with adding a path segment to the CONTENT_URI + "/" + <primary key>

Column name definition constants to be used when accessing this provider:

```
static final String CONTACT = "contact"
static final String CAPABILITY_IMAGE_SHARE = "capability_image_share"
static final String CAPABILITY_VIDEO_SHARE = "capability_video_share"
static final String CAPABILITY_FILE_TRANSFER = "capability_file_transfer"
static final String CAPABILITY_IM_SESSION = "capability_im_session"
static final String CAPABILITY_GEOLOC_PUSH = "capability_geoloc_push"
static final String CAPABILITY_EXTENSIONS = "capability_extensions"
static final String AUTOMATA = "automata"
static final String TIMESTAMP = "timestamp"
```

The content provider has the following columns:

Data	Data type	Comment
CONTACT	String (primary key)	ContactId formatted number of contact associated to the capabilities
CAPABILITY_IMAGE_SHARING	Integer	Image sharing capability. Values: 1 (true), 0 (false)
CAPABILITY_VIDEO_SHARING	Integer	Video sharing capability. Values: 1 (true), 0 (false)
CAPABILITY_IM_SESSION	Integer	IM/Chat capability. Values: 1 (true), 0 (false)
CAPABILITY_FILE_TRANSFER	Integer	File transfer capability. Values: 1 (true), 0 (false)
CAPABILITY_GEOLOC_PUSH	Integer	Geolocation push capability. Values: 1 (true), 0 (false)
CAPABILITY_EXTENSIONS	String	Supported RCS extensions. List of features tags semicolon separated (e.g. <TAG1>;<TAG2>;…;TAGn)
AUTOMATA	Integer	Is an automata. Values: 1 (true), 0 (false).
TIMESTAMP	Long	Timestamp of the last capability refresh

4.4.4.5 RCS extensions

A MNO/OEM application can create a new RCS/IMS service by defining a new RCS capability (or RCS extension). This new service is identified by an IARI feature tag which is the unique key to identify the service in the RCS API and to trigger the service internally in the device and to route the service on the network side.

Note: How the IARI feature tags are used in the RCS API is for further study

To create a new capability, the MNO/OEM application should declare the new supported feature tag in its Android Manifest file. Then, when the MNO/OEM application is deployed on the device, the RCS service will detect automatically the new installed application and will take into account the new feature tag in the next capability refreshes, via SIP OPTIONS.

When the MNO/OEM application is removed the RCS service will remove the associated capability from the next capability refreshes via SIP OPTIONS.

The role of the RCS service is to manage the extensions and to take into account the new feature tag or not. This may be done by analysing the certificate of the application supporting the feature tag or by checking the provisioning.

There are two type of extensions:

- Extensions for service provider specific service.
- Extensions for third-party specific service.

For a third-party specific service, the extension should start with the prefix « `+g.3gpp.iari-ref="urn%3Aurn-7%3A3gpp-application.ims.iari.rcs.ext.xxx"`, where “xxx“ is a unique service identifier

encoded in base64 as per [RFC4648] associated to the application implementing the RCS extension.

See the following API syntax to be added in the Android Manifest file:

```
<application>
    <meta-data
        android:name="com.gsma.services.rcs.capability.EXTENSION"

android:value="ext.A5TgS99bJloIUIh1209SJ82B21m87S1B87SBqfS871BS8787SBXBA3P45wjp63tk" />
</application>
```

For a service provider specific service, the extension should start with the prefix « +g.3gpp.iari-ref=”urn%3Aurn-7%3A3gpp-application.ims.iari.rcs.mnc<mnc>.mcc<mcc>.xxxx », where « mnc » is the Mobile Network Code, where « mcc » is the Mobile Country Code and « xxx » a unique service identifier (string) associated to the application implementing the RCS extension.

See the following API syntax to be added in the Android Manifest file:

```
<application>
    <meta-data
        android:name="com.gsma.services.rcs.capability.EXTENSION"
        android:value="mnc01.mcc208.xxx"/" />

</application>
```

Several extensions may be associated per applications, this means the meta-data may contain several tags separated by a semicolon. See the following API syntax:

```
<application>
    <meta-data
        android:name="com.gsma.services.rcs.capability.EXTENSION"
        android:value="ext.xxx;ext.yyy;ext.zzz"/" />

</application>
```

4.4.4.6 Permissions

Access to the Capabilities API is requires the following permissions:

- com.gsma.services.rcs.RCS_READ_CAPABILITIES:
  this is a new permission that governs access to capability information.
- android.permission.READ_CONTACTS:
  this permission is required by any client using the capabilities service, since use of the API implicitly reveals information about past and current contacts for the device.

4.4.5 IM/Chat API

This API exposed all functionality for the Instant Messaging/Chat Service. It allows:

- Sending messages to a contact.
- Starting group chats with an ad-hoc list of participants and an optional subject.
- Joining existing group chats.
- Re-joining existing group chats (this is done implicitly by the implementation when needed).
- Restarting a previous group chat (this is done implicitly by the implementation when needed).
- Extends a 1-1 chat to a group chat.
- Sending messages in a group chat.

- Leaving a group chat.
- Adding participants to a group chat.
- Retrieving information about a group chat (status, participants and their status)
- Receiving notifications about incoming messages, “is-composing” events, group chat invitations and group chat events.
- Accept/reject an incoming chat invitation.
- Displaying chat history (messages and group chats).
- Erasing chat history by a user, by group chat, or by single messages.
- Marking messages as displayed.
- Receiving message delivery reports.
- Read configuration elements affecting IM.
- Message queuing.

NOTE: a group chat is identified by a unique conversation Identifier (ID) which corresponds to the “Contribution-ID” header in the signalling flow. A one to one chat is identified by the ContactId of the remote contact. This permits to have a permanent one to one chat or group chat like user experience.

4.4.5.1 Package

Package name **com.gsma.services.rcs.chat**

4.4.5.2 Methods and Callbacks

Class **ChatService**:

This class offers the main entry point to initiate chat conversations with contacts: 1-1 and group chat conversation. Several applications may connect/disconnect to the API.

- Method: connects to the API.

```
void connect()
```

- Method: disconnects from the API.

```
void disconnect()
```

- Method: returns a one to one chat with the specified contact. If no such ongoing chat exists a reference is returned to a fresh one to one chat so that a call to sendMessage on that will initiate a new invitation to the remote contact.

```
OneToOneChat getOneToOneChat(ContactId contact)
```

- Method: returns a group chat from its unique ID. If no ongoing group chat matching the chatId is found the a reference to a historical chat is returned so that a call to sendMessage on that one can try to rejoin that group chat automatically before sending the message.

```
GroupChat getGroupChat(String chatId)
```

- Method: returns true if it’s possible initiate a new group chat with the specified contactIds right now, else returns false.

```
boolean canInitiateGroupChat(Set<ContactId> contacts)
```

- Method: initiates a group chat with a group of contacts and returns a GroupChat instance. The subject is optional and may be null.

```
GroupChat initiateGroupChat(Set<ContactId> contacts, String subject)
```

- Method: mark a received message as read (ie. displayed in the UI)

```
void markMessageAsRead(String msgId)
```

- Method: returns the configuration for chat service.

```
ChatServiceConfiguration getConfiguration()
```

- Method: deletes all one to one chats from history and abort/reject corresponding sessions if such are ongoing.

```
void deleteOneToOneChats()
```

- Method: deletes all group chats from history and abort/reject corresponding sessions if such are ongoing.

```
void deleteGroupChats()
```

- Method: deletes a one to one chats conversation with a given contact from history and abort/reject corresponding sessions if such are ongoing.

```
void deleteOneToOneChat(ContactId contact)
```

- Method: deletes a group chat conversation from its chat ID from history and abort/reject corresponding sessions if such are ongoing.

```
void deleteGroupChat(String chatId)
```

- Method: delete a message from its message ID from history

```
void deleteMessage(String msgId)
```

- Method: Adds a listener for one to one chat events

```
void addEventListener(OneToOneChatListener listener)
```

- Method: Removes a listener for one to one chat events

```
void removeEventListener(OneToOneChatListener listener)
```

- Method: Adds a listener for group chat events

```
void addEventListener(GroupChatListener listener)
```

- Method: Removes a listener for group chat events

```
void removeEventListener(GroupChatListener listener)
```

- Method: set the parameter that controls whether to respond or not to display reports when requested by the remote. Applicable to one to one chat messages.

```
void setRespondToDisplayReports(boolean enable)
```

- Method: returns a set of message IDs of undelivered chat messages corresponding to the contact if there are any.

```
Set<String> getUndeliveredMessages(ContactId contact)
```

- Method: marks undelivered chat messages to indicate that messages have been processed.

```
void markUndeliveredMessagesAsProcessed(Set<String> msgIds)
```

Class **ChatMessage**:

This class contains chat message information for single and group chats.

- Method: returns the contactId the remote contact of this message.

```
ContactId getContact()
```

- Method: returns the message ID.

```
String getId()
```

- Method: returns the message content.

```
String getContent()
```

- Method: returns the direction of the chat message.

```
com.gsma.services.rcs.RcsService.Direction getDirection()
```

- Method: returns the mime type of the chat message.

```
String getMimeType()
```

- Method: returns the local timestamp of when the chat message was sent and/or queued for outgoing messages or the local timestamp of when the chat message was received for incoming messages..

```
long getTimestamp()
```

- Method: returns the local timestamp of when the chat message was sent and/or queued for outgoing messages or the remote timestamp of when the chat message was sent for incoming messages.

```
long getTimestampSent()
```

Class **GeolocMessage:**

This class contains geoloc message information for single and group chat.

- Method: returns the contact of the remote contact of this message.

  ```
  ContactId getContact()
  ```

- Method: returns the message ID.

  ```
  String getId()
  ```

- Method: returns the geoloc info.

  ```
  Geoloc getGeoloc()
  ```

- Method: returns the direction of the geoloc message.

  ```
  com.gsma.services.rcs.RcsService.Direction getDirection()
  ```

- Method: returns the mime type of the geoloc message.

  ```
  String getMimeType()
  ```

- Method: returns the local timestamp of when the geoloc message was sent and/or queued for outgoing messages or the local timestamp of when the geoloc message was received for incoming messages.

  ```
  long getTimestamp()
  ```

- Method: returns the local timestamp of when the geoloc message was sent and/or queued for outgoing messages or the remote timestamp of when the geoloc message was sent for incoming messages.

  ```
  long getTimestampSent()
  ```

Class **Geoloc:**

This class contains geoloc information.

- Method: returns the label associated to the geoloc.

  ```
  String getLabel()
  ```

- Method: returns the latitude.

  ```
  double getLatitude()
  ```

- Method: returns the longitude.

  ```
  double getLongitude()
  ```

- Method: returns the accuracy of the geoloc info (in meter).

```
float getAccuracy()
```

- Method: returns the expiration date of the geoloc info.

```
long getExpiration()
```

Class **OneToOneChat** :

This class maintains the information related to a 1-1 chat and offers methods to manage the chat conversation.

- Method: open the chat conversation. Note: if it's an incoming pending chat session and the parameter IM SESSION START is 0 then the session is accepted now.

```
void openChat()
```

- Method: returns the remote contactId.

```
ContactId getRemoteContact()
```

- Method: sends a chat message. The method returns a unique message ID. The message is queued if it can't be sent immediately. Note: if it's an incoming pending chat session and the parameter IM SESSION START is 2 then the session is accepted before sending the message. The text parameter is considered as mime type "plain/text" and the ChatMessage will be stored in the message provider as such.

```
ChatMessage sendMessage(String text)
```

- Method: sends a geoloc message. The method returns a unique message ID. The message is queued if it can't be sent immediately. Note: if it's an incoming pending chat session and the parameter IM SESSION START is 2 then the session is accepted before sending the message. The geoloc parameter is considered as mime type "application/geoloc" and the GeolocMessage will be stored in the message provider as such.

```
GeolocMessage sendMessage(Geoloc geoloc)
```

- Method: sends an "is-composing" event. The status should be set to true when typing a message, else set to false. Note: if it's an incoming pending chat session and the parameter IM SESSION START is 1 then the session is accepted before sending the is-composing event.

```
void sendIsComposingEvent(boolean status)
```

- Method: resend a message which previously failed.

```
void resendMessage(String msgId)
```

Class **OneToOneChatListener**:

This class offers callback methods on 1-1 chat events.

- Method: Callback called when a message status/reasonCode is changed.

```
void onMessageStatusChanged(ContactId contactId, String msgId,
ChatLog.Message.Content.Status status,
ChatLog.Message.Content.ReasonCode reasonCode)
```

- Method: Callback called when a “is-composing” event has been received. If the remote is typing a message the status is set to true, else it is false.

```
void onComposingEvent(ContactId contact, boolean status)
```

- Method: callback called when a delete operation completed that resulted in that one or several one to one chat messages was deleted specified by the msgIds parameter corresponding to a specific contact.

```
void onMessagesDeleted(ContactId contact, Set<String> msgIds)
```

Class **GroupChat**:

This class maintains the information related to a group chat and offers methods to manage the group chat conversation.

- Enum: the Group Chat state.

```
enum State { INVITED(0), INITIATED(1), STARTED(2), ABORTED(3),
FAILED(4), ACCEPTING(5), REJECTED(6) }
```

- Enum: the reason code for the Group Chat.

```
enum ReasonCode { UNSPECIFIED(0), ABORTED_BY_USER(1),
ABORTED_BY_REMOTE(2), ABORTED_BY_SYSTEM(3),
REJECTED_BY_SECONDARY_DEVICE (4), REJECTED_SPAM(5),
REJECTED_MAX_CHATS(6) , REJECTED_BY_USER(7) , REJECTED_BY_REMOTE(8),
REJECTED_TIME_OUT(9), FAILED_INITIATION(10) }
```

- Method: returns the chat ID.

```
String getChatId()
```

- Method: returns the subject of the group chat.

```
String getSubject()
```

- Method: returns the list of participants and associated infos.

```
Set<ParticipantInfo> getParticipants()
```

- Method: returns the direction of the group chat.

```
com.gsma.services.rcs.RcsService.Direction getDirection()
```

- Method: returns the state of the group chat.

```
State getState()
```

- Method: returns the reason code of the group chat.

```
ReasonCode getReasonCode()
```

- Method: open the chat conversation. Note: if it's an incoming pending chat session and the parameter IM SESSION START is 0 then the session is accepted now.

```
void openChat()
```

- Method: returns true if it's possible to send messages in the group chat right now, else returns false. (for instance it is not possible to send additional messages after a group chat has been left willingly by calling the leave()-method above)

```
boolean canSendMessage()
```

- Method: sends a message to the group. This method returns a unique message ID. Note: if it's an incoming pending chat session and the parameter IM SESSION START is 2 then the session is accepted before sending the message or rejoined if session was in timeout. The text parameter is considered as mime type "plain/text" and the ChatMessage will be stored in the message provider as such.

```
ChatMessage sendMessage(String text)
```

- Method: sends a geoloc message. The method returns a unique message ID. Note: if it's an incoming pending chat session and the parameter IM SESSION START is 2 then the session is accepted before sending the message or rejoined if session was in timeout. The geoloc parameter is considered as mime type "application/geoloc" and the GeolocMessage will be stored in the message provider as such.

```
GeolocMessage sendMessage(Geoloc geoloc)
```

- Method: sends a "is-composing" event. The status should be set to true when typing a message, else set to false. Note: if it's an incoming pending chat session and the parameter IM SESSION START is 1 then the session is accepted before sending the is-composing event.

```
void sendIsComposingEvent(boolean status)
```

- Method: returns true if it's possible to add additional participants to the group chat right now, else returns false.

```
boolean canAddParticipants()
```

- Method: returns true if it's possible to add the specified participants to the group chat right now, else returns false.

- `boolean canAddParticipants(Set<ContactId> participants)`

- Method: adds participants to a group chat.

```
void addParticipants(Set<ContactId> participants)
```

- Method: returns the maximum number of participants for a group chat from the group chat info subscription (this value overrides the provisioning parameter).

```
int getMaxParticipants()
```

- Method: Leaves a group chat willingly and permanently. The group chat will continue between other participants if there are enough participants.

```
void leave()
```

Class **GroupChatListener**:

This class offers callback methods on group chat events.

- Method: Callback called when a group chat state/reasonCode is changed.

```
void onStateChanged(String chatId, GroupChat.State state,
GroupChat.ReasonCode reasonCode)
```

- Method: Callback called when a message status/reasonCode is changed.

```
void onMessageStatusChanged(String chatId, String msgId,
ChatLog.Message.Content.Status status,
ChatLog.Message.Content.ReasonCode reasonCode)
```

- Method: callback called when a “is-composing” event has been received. If the remote is typing a message the status is set to true, else it is false.

```
void onComposingEvent(String chatId, ContactId contact, boolean
status)
```

- Method: Callback called when a group delivery info status/reasonCode was changed for a single recipient to a group message.

```
void onMessageGroupDeliveryInfoChanged(String chatId, ContactId
contact, String msgId, GroupDeliveryInfo.Status status,
GroupDeliveryInfo.ResonCode reasonCode)
```

- Method: callback called when a participant status has been changed in a group chat.

```
void onParticipantInfoChanged(String chatId, ParticipantInfo info)
```

- Method: callback called when a delete operation completed that resulted in that one or several group chats was deleted specified by the chatIds parameter.

```
void onDeleted(Set<String> chatIds)
```

- Method: callback called when a delete operation completed that resulted in that one or several group chat messages was deleted specified by the msgIds parameter corresponding to a specific group chat.

```
void onMessagesDeleted(String chatId, Set<String> msgIds)
```

Class **ParticipantInfo**:

This class contains information related to Group Chat Participant.

- Enum: the status of the participant.

```
enum Status { INVITING(0), INVITED(1), CONNECTED(2), DISCONNECTED(3),
DEPARTED(4), FAILED(5), DECLINED(6), TIMEOUT(7) }
```

- Method: returns the contact Id.

```
ContactId getContact()
```

- Method: returns the status of the participant.

```
Status getStatus()
```

Class **ChatServiceConfiguration**:

This class represents the particular configuration of a IM Service.

- Method: returns the “imCapAlwaysOn” configuration. True if Store and Forward capability is supported, False if no Store & Forward capability.

```
boolean isChatSf()
```

- Method: returns the “imWarnSF” configuration. True if a user should be informed when sending a message to an offline user. False if a user should not be informed when sending a message to an offline user. This should be used with imCapAlwaysOn.

```
boolean isChatWarnSf()
```

- Method: returns the time after an inactive chat could be closed.

```
int getChatTimeout()
```

- Method: returns the maximum number of participants in a group chat .

```
int getGroupChatMaxParticipants()
```

- Method: returns the minimum number of participants in a group chat .

```
int getGroupChatMinParticipants()
```

- Method: returns the maximum single chat message's length can have. The length is the number of bytes of the message encoded in UTF-8.

```
int getOneToOneChatMessageMaxLength()
```

- Method: returns the maximum single group chat message's length can have. The length is the number of bytes of the message encoded in UTF-8.

```
int getGroupChatMessageMaxLength()
```

- Method: returns the maximum group chat subject's length can have. The length is the number of bytes of the message encoded in UTF-8.

```
int getGroupChatSubjectMaxLength()
```

- Method: returns the SMS fall-back configuration. True if SMS fall-back procedure is activated, else returns False.

```
boolean isSmsFallback()
```

- Method: return True if the client application should send a displayed report when requested by the remote part. Only applicable to one to one chat messages.

```
boolean isRespondToDisplayReportsEnabled()
```

- Method: returns the is-composing timeout value.

```
int getIsComposingTimeout()
```

- Method: returns the maximum length of a geoloc label.

```
int getGeolocLabelMaxLength()
```

- Method: returns the expiration time of a geoloc info.

```
int getGeolocExpirationTime()
```

- Method: returns True if group chat is supported, else returns False.

```
boolean isGroupChatSupported()
```

Class **ChatLog.GroupChat**:

- Method: utility method to get list of ParticipantInfo objects from its string representation in the ChatLog provider.

```
static List<ParticipantInfo> getParticipantInfo(Context ctx, String
participantsInfo)
```

Class **ChatLog.Message**:

- Method: utility method to get a geoloc object from its string representation in the CONTENT field of the ChatLog provider.

```
static Geoloc getGeoloc(String geoloc)
```

Class **ChatLog.Message.MimeTypes**

```
static final TEXT_MESSAGE = "text/plain"
static final GEOLOC_MESSAGE = "application/geoloc"
static final GROUPCHAT_EVENT = "rcs/groupchat-event"
```

Class **ChatLog.Message.Content**

- Enum : the status of a Content message

```
enum Status { REJECTED(0), QUEUED(1), SENDING(2), SENT(3), FAILED(4),
DELIVERED(5), DISPLAY_REPORT_REQUESTED(6), RECEIVED(7), DISPLAYED(8)
```

- Enum: the reason code for Content message

```
enum ReasonCode { UNSPECIFIED(0), FAILED_SEND(1), FAILED_DELIVERY(2),
FAILED_DISPLAY(3), REJECTED_SPAM(4) }
```

Class **ChatLog.Message.GroupChatEvent**

- Enum : the status of a group chat event message

```
enum Status { JOINED(0), DEPARTED(1) }
```

4.4.5.3 Intents

Intent broadcasted when a new 1-1 chat message has been received. This Intent contains the following extra:

- "messageId": (String) unique message ID of the message.

```
com.gsma.services.rcs.chat.action.NEW_ONE_TO_ONE_CHAT_MESSAGE
```

Intent broadcasted when a new group chat invitation has been received. This Intent contains the following extra:

- "chatId": (String) unique ID of the group chat conversation.

```
com.gsma.services.rcs.chat.action.NEW_GROUP_CHAT
```

Intent broadcasted when there are some undelivered messages detected corresponding to the contact as specified in the intent parameter. This Intent contains the following extra:

- "contact": (ContactId) ContactId of the contact corresponding to the conversation.

```
com.gsma.services.rcs.chat.action.UNDELIVERED_MESSAGES
```

Intent broadcasted when a new group chat message has been received. This Intent contains the following extra:

- “messageId”: (String) unique message id of the message.

```
com.gsma.services.rcs.chat.action.NEW_GROUP_CHAT_MESSAGE
```

4.4.5.4 Content Providers

A content provider is used to store the group chats and the message history persistently. There is one entry per group chat and per chat message.

Class **ChatLog.GroupChat:**

URI constant to be able to query the provider data (Note that only read operations are supported since exposing write operations would conflict with the fact that the stack is performing write operations internally to keep the data matching the current situation):

```
static final Uri CONTENT_URI =
"content://com.gsma.services.rcs.provider.chat/groupchat"
```

The “CHAT_ID” column below is defined as the unique primary key and can be references with adding a path segment to the CONTENT_URI + “/” + <primary key>

Column name definition constants to be used when accessing this provider:

```
static final String CHAT_ID = “chat_id”
static final String STATE = “state”
static final String SUBJECT = “subject”
static final String DIRECTION = “direction”
static final String TIMESTAMP = “timestamp”
static final String REASON_CODE = “reason_code”
static final String PARTICIPANTS = “participants”
```

The content provider has the following tables and columns:

GROUPCHAT

Data	Data Type	Description
CHAT_ID	String (primary key)	Id for chat room
STATE	Integer	State of chat room. See enum GroupChat.State for the list of states.
SUBJECT	String	Subject of the group chat room
DIRECTION	Integer	Status direction of group chat. See enum Direction for the list of directions.
TIMESTAMP	Long	timestamp of the invitation
REASON_CODE	Integer	Reason code associated with the group chat state. See enum GroupChat.ResonCode for the list of reason codes
PARTICIPANTS	String	List of participants and associated status stored as a String parseable with the ChatLog.GroupChat.getParticipantInfos() method.

Class **ChatLog.Message:**

Event log provider member id used when merging the data from this provider with other registered event log provider members data into a common cursor:

```
static final int HISTORYLOG_MEMBER_ID = 1
```

URI constant to be able to query the provider data (Note that only read operations are supported since exposing write operations would conflict with the fact that the stack is performing write operations internally to keep the data matching the current situation):

```
static final Uri CONTENT_URI =
"content://com.gsma.services.rcs.provider.chat/chatmessage"
```

The “MESSAGE_ID” column below is defined as the unique primary key and can be references with adding a path segment to the CONTENT_URI + “/” + <primary key>

Data	Data Type	Description
MESSAGE_ID	String (primary key)	Id of the message
CHAT_ID	String (not null)	Id of chat room
CONTACT	String	ContactId formatted number of remote contact or null if the message is an outgoing group chat message.
CONTENT	String	Content of the message
TIMESTAMP	Long	Time when message inserted
TIMESTAMP_SENT	Long	Time when message sent. If 0 means not sent.
TIMESTAMP_DELIVERED	Long	Time when message delivered. If 0 means not delivered.
TIMESTAMP_ DISPLAYED	Long	Time when message displayed. If 0 means not displayed.
MIME_TYPE	String (not null)	Multipurpose Internet Mail Extensions (MIME) type of message
STATUS	Integer	See enum Message.Content.Status for the list of status.
REASON_CODE	Integer	Reason code associated with the message status. See enum Message.Content.ReasonCode for the list of reason codes
READ_STATUS	Integer	This is set on the receiver side when the message has been displayed. See enum ReadStatus for the list of status.
DIRECTION	Integer	Status direction of message. See enum Direction for the list of directions.

Column name definition constants to be used when accessing this provider:

```
static final String MESSAGE_ID = “msg_id"
static final String CHAT_ID = "chat_id"
static final String CONTACT = "contact"
static final String CONTENT = "content"
```

```
static final String TIMESTAMP = "timestamp"
static final String TIMESTAMP_SENT = "timestamp_sent"
static final String TIMESTAMP_DELIVERED = "timestamp_delivered"
static final String TIMESTAMP_DISPLAYED = "timestamp_displayed"
static final String MIME_TYPE = "mime_type"
static final String STATUS = "status"
static final String REASON_CODE = "reason_code"
static final String READ_STATUS = "read_status"
static final String DIRECTION = "direction"
```

CHATMESSAGE

4.4.5.5 Permissions

Access to the Chat API requires the following permissions:

- com.gsma.services.rcs.RCS_USE_CHAT: this is a new permission that governs access to the chat API, and is required both to receive and to send over an RCS chat session.

- com.gsma.services.rcs.RCS_READ_CHAT: this is a new permission that is required by a client in order to read the chat history from the content provider.

4.4.6 File Transfer API

This API exposes all functionality related to transferring files via the File Transfer Service. It allows:

- Send a file transfer request
- Send a file transfer request with thumbnail (file icon)
- Receive notifications about incoming file transfer and file transfer events.
- Receive notifications upon file delivery
- Retrieve the list of all file transfers and their statuses for a specific contact
- Clean all file transfer history or single file transfers (including the transferred files if possible)
- Monitor a file transfer's progress.
- Cancel a file transfer in progress.
- Accept/reject an incoming file transfer request.
- Read configuration elements affecting a file transfer
- Resume a file transfer
- File transfer queuing.
- Send several files to a list of contacts.

4.4.6.1 Package

Package name **com.gsma.services.rcs.filetransfer**

4.4.6.2 Methods and Callbacks

Class **FileTransferService**:

This class offers the main entry point to transfer files and to receive files. Several applications may connect/disconnect to the API.

- Method: connects to the API.

```
void connect()
```

- Method: disconnects from the API.

```
void disconnect()
```

- Method: returns the list of file transfers in progress.

```
Set<FileTransfer> getFileTransfers()
```

- Method: returns a file transfer from its unique ID. If no ongoing FileTransfer matching the transferId is found then a reference to a historical FileTransfer is returned so that a call to resendTransfer() and other methods still can be performed.

```
FileTransfer getFileTransfer(String transferId)
```

- Method: transfers a file to a contact with an optional file icon.

```
FileTransfer transferFile(ContactId contact, Uri file, boolean
attachFileicon)
```

- Method: returns true if it's possible to initiate file transfer to the group chat specified by the chatId parameter, else returns false.

```
boolean canTransferFileToGroupChat(String chatId)
```

- Method: transfers a file to a group chat with an optional file icon.

```
FileTransfer transferFileToGroupChat(String chatId, Uri file, boolean
attachFileicon)
```

- Method: mark a received file transfer as read (i.e. the invitation or the file has been displayed in the UI).

```
void markFileTransferAsRead(String transferId);
```

- Method: returns the configuration for a File Transfer service.

```
FileTransferServiceConfiguration getConfiguration()
```

- Method: adds a one to one file transfer event listener.

```
void addEventListener(OneToOneFileTransferListener listener)
```

- Method: removes a one to one file transfer event listener.

```
void removeEventListener(OneToOneFileTransferListener listener)
```

- Method: Adds a group file transfer event listener.

```
void addEventListener(GroupFileTransferListener listener)
```

- Method: Removes a group file transfer event listener.

```
void removeEventListener(GroupFileTransferListener listener)
```

- Method: deletes all one to one file transfers history and abort/reject corresponding sessions if such are ongoing.

```
void deleteOneToOneFileTransfers()
```

- Method: deletes all group file transfers from history and abort/reject corresponding sessions if such are ongoing.

```
void deleteGroupFileTransfers()
```

- Method: deletes file transfers corresponding to a given one to one conversation specified by contact from history and abort/reject corresponding sessions if such are ongoing.

```
void deleteOneToOneFileTransfers(ContactId contact)
```

- Method: deletes file transfers corresponding to a given group chat specified by chat id from history and abort/reject corresponding sessions if such are ongoing.

```
void deleteGroupFileTransfers(String chatId)
```

- Method: deletes a file transfer from its unique ID from history and abort/reject corresponding sessions if such are ongoing.

```
void deleteFileTransfer(String transferId)
```

- Method: set the Auto Accept Mode of a File Transfer configuration. The Auto Accept Mode can only be modified by client application if isAutoAcceptChangeable (see **FileTransferServiceConfiguration** class) is true.

```
void setAutoAccept(boolean enable)
```

- Method: set the Auto Accept Mode of a File Transfer configuration while roaming. The AutoAcceptInRoaming can only be modified by client application if isAutoAcceptModeChangeable (see **FileTransferServiceConfiguration** class) is true and if the Auto Accept Mode in normal conditions is true.

```
void setAutoAcceptInRoaming(boolean enable)
```

- Method: set the image resize option for file transfer.

```
void setImageResizeOption(ImageResizeOption option)
```

- Method: returns a set of transfer IDs of undelivered file transfers corresponding to the contactId parameter if there are any.

```
Set<String> getUndeliveredFileTransfers(ContactId contact)
```

- Method: marks undelivered file transfers to indicate that transfers have been processed.

```
void markUndeliveredFileTransfersAsProcessed(Set<String> transferIds)
```

Class **FileTransfer**:

This class maintains the information related to a file transfer and offers methods to manage the transfer.

- Enum: the file transfer state.

```
Enum State { INVITED(0), ACCEPTING(1), REJECTED(2), QUEUED(3),
INITIATED(4), STARTED(5), PAUSED(6) , ABORTED(7), TRANSFERRED(8),
FAILED(9), DELIVERED(10), DISPLAYED(11) }
```

- Enum: the reason code for the file transfer.

```
Enum ReasonCode { UNSPECIFIED(0), ABORTED_BY_USER(1),
ABORTED_BY_REMOTE(2), ABORTED_BY_SYSTEM(3),
REJECTED_BY_SECONDARY_DEVICE(4), REJECTED_TIME_OUT(5),
REJECTED_SPAM(6), REJECTED_LOW_SPACE(7), REJECTED_MAX_SIZE(8),
REJECTED_MAX_FILE_TRANSFERS(9), REJECTED_BY_USER(10),
REJECTED_BY_REMOTE(11), PAUSED_BY_SYSTEM(12), PAUSED_BY_USER(13),
FAILED_INITIATION(14), FAILED_DATA_TRANSFER(15), FAILED_SAVING(16),
FAILED_DELIVERY(17), FAILED_DISPLAY(18),
FAILED_NOT_ALLOWED_TO_SEND(19) }
```

- Method: returns whether the transfer is a group transfer.

```
boolean isGroupTransfer()
```

- Method: Returns the chat ID if the file transfer.

```
String getChatId()
```

- Method: returns the file transfer ID of the file transfer.

```
String getTransferId()
```

- Method: returns the remote contact Id.

```
ContactId getRemoteContact()
```

- Method: returns the direction of the transfer.

```
com.gsma.services.rcs.RcsService.Direction getDirection()
```

- Method: returns URI of the file icon or thumbnail.

```
Uri getFileIcon()
```

- Method: returns the mime type of the file icon.

```
String getFileIconMimeType()
```

- Method: returns URI of the file to be transferred.

```
Uri getFile()
```

- Method: returns the mime type of the file transfer.

```
String getMimeType()
```

- Method: returns the filename of the file to be transferred.

```
String getFileName()
```

- Method: returns the size of the file to be transferred (in bytes).

```
String getFileSize()
```

- Method: returns the MIME type of file to be transferred.

```
String getFileType()
```

- Method: returns the state of the file transfer.

```
State getState()
```

- Method: returns the reason code of the file transfer.

```
ReasonCode getReasonCode()
```

- Method: accepts the file transfer invitation.

```
void acceptInvitation()
```

- Method: rejects the file transfer invitation.

```
void rejectInvitation()
```

- Method: aborts the file transfer.

```
void abortTransfer()
```

- Method: returns true if it’s possible to pause this file transfer right now, else returns false. If this Filetransfer corresponds to a file transfer that is no longer present in the persistent storage false will be returned (this is no error).

```
boolean canPauseTransfer()
```

- Method: pauses the file transfer.

```
void pauseTransfer()
```

- Method: resumes the file transfer.

```
void resumeTransfer()
```

- Method: returns whether you can resend the transfer.

```
boolean canResendTransfer()
```

- Method: resend a file transfer which was previously failed. This is only for 1-1 file transfer, an exception is thrown in case of a file transfer to group.

```
void resendTransfer()
```

- Method: returns the local timestamp of when the file transfer was initiated and/or queued for outgoing file transfers or the local timestamp of when the file transfer invitation was received for incoming file transfers.

```
long getTimestamp()
```

- Method: returns the local timestamp of when the file transfer was initiated and/or queued for outgoing file transfers or the remote timestamp of when the file transfer was initiated for incoming file transfers.

```
long getTimestampSent()
```

Class **OneToOneFileTransferListener**:

This class offers callback methods on file transfer events.

- Method: Callback called when the file transfer status/reasonCode is changed.

```
void onStateChanged(ContactId contact, String transferId,
FileTransfer.State state, FileTransfer.ReasonCode reasonCode)
```

- Method: callback called during the file transfer progress.

```
void onProgressUpdate(ContactId contact, String transferId, long
currentSize, long totalSize)
```

- Method: callback called when a delete operation completed that resulted in that one or several one to one file transfers was deleted specified by the transferIds parameter corresponding to a specific contact.

```
void onDeleted(ContactId contact, Set<String> transferIds)
```

Class **GroupFileTransferListener**

This class offers callback methods on group file transfer events.

- Method: Callback called when the group file transfer status/reasonCode is changed.

```
void onStateChanged(String chatId, String transferId,
FileTransfer.State state, FileTransfer.ReasonCode reasonCode)
```

- Method: Callback called during the transfer progress of group file transfer

```
void onProgressUpdate(String chatId, String transferId, long
currentSize, long totalSize)
```

- Method: Callback called when a group file transfer group delivery info status/reasonCode is changed for a single recipient only.

- `void onDeliveryInfoChanged(String chatId, String transferId, ContactId contact, GroupDeliveryInfo.Status status, GroupDeliveryInfo.ReasonCode reasonCode)`

- Method: callback called when a delete operation completed that resulted in that one or several group file transfers was deleted specified by the transferIds parameter corresponding to a specific group chat.

```
void onDeleted(String chatId, Set<String> transferIds)
```

Class **FileTransferServiceConfiguration**:

This class represents the particular configuration of a FT Service.

- Enum: the image resize option.

```
enum ImageResizeOption { ALWAYS_PERFORM(0), ONLY_ABOVE_MAX_SIZE(1),
ASK(2) }
```

- Method: returns the file size warning of a File Transfer configuration. It can return null if this value was not set by the auto-configuration server (no need to warn).

```
long getWarnSize()
```

- Method: returns the max file size of a File Transfer configuration. It can return null if this value was not set by the auto-configuration server.

```
long getMaxSize()
```

- Method: returns the Auto Accept Mode of a File Transfer configuration.

```
boolean isAutoAcceptEnabled()
```

- Method: returns True if client is allowed to change the Auto Accept mode (both in normal or roaming modes) in file transfer

```
boolean isAutoAcceptModeChangeable()
```

- Method: returns the Auto Accept Mode of a File Transfer configuration while roaming. This parameter is only applicable if auto accept is active for File Transfer in normal conditions (see above isAutoAcceptEnabled)

```
boolean isAutoAcceptInRoamingEnabled()
```

- Method: returns the image resize option

```
ImageResizeOption getImageResizeOption()
```

- Method: returns the max number of simultaneous file transfers.

```
int getMaxFileTransfers()
```

- Method: returns True if group file transfer is supported, else returns False.

```
boolean isGroupFileTransferSupported()
```

4.4.6.3 Intents

Intent broadcasted when a new file transfer invitation has been received. This Intent contains the following extra:

- “transferId”: (String) unique ID of the file transfer.

```
com.gsma.services.rcs.ft.action.NEW_FILE_TRANSFER
```

Intent broadcasted when a file transfer is resumed automatically by the stack. This Intent contains the following extra:

- “transferId”: (String) unique ID of the file transfer.

com.gsma.services.rcs.ft.action.RESUME_FILE_TRANSFER

Intent broadcasted when there are some undelivered file transfers detected corresponding to the contact as specified in the intent parameter. This Intent contains the following extras:

- “contact”: (ContactId) ContactId of the contact corresponding to the conversation.

```
com.gsma.services.rcs.ft.action.UNDELIVERED_FILE_TRANSFERS
```

4.4.6.4 Content Providers

A content provider is used to store the file transfer history persistently. There is one entry per file transfer.

Class **FileTransferLog:**

Event log provider member id used when merging the data from this provider with other registered event log provider members data into a common cursor:

```
static final int HISTORYLOG_MEMBER_ID = 2
```

URI constant to be able to query the provider data (Note that only read operations are supported since exposing write operations would conflict with the fact that the stack is performing write operations internally to keep the data matching the current situation):

```
static final Uri CONTENT_URI =
"content://com.gsma.services.rcs.provider.filetransfer/filetransfer"
```

The “FT_ID” column below is defined as the unique primary key and can be references with adding a path segment to the CONTENT_URI + “/” + <primary key>

Column name definition constants to be used when accessing this provider:

```
static final String FT_ID = "ft_id"
static final String CHAT_ID = "chat_id"
static final String CONTACT = "contact"
static final String FILE = "file"
static final String FILENAME = "filename"
static final String MIME_TYPE = "mime_type"
static final String FILEICON = "fileicon"
static final String FILEICON_MIME_TYPE = "fileicon_mime_type"
static final String DIRECTION = "direction"
static final String FILESIZE = "filesize"
static final String TRANSFERRED = "transferred"
static final String TIMESTAMP = "timestamp"
static final String TIMESTAMP_SENT = "timestamp_sent"
static final String TIMESTAMP_DELIVERED = "timestamp_delivered"
static final String TIMESTAMP_DISPLAYED = "timestamp_displayed"
static final String STATE = "state"
static final String REASON_CODE = "reason_code"
static final String READ_STATUS = "read_status"
```

The content provider has the following columns:

FILETRANSFER

Data	Data type	Comment
FT_ID	String (primary key)	Unique file transfer identifier
CHAT_ID	String (not null)	Id of chat
CONTACT	String	ContactId formatted number of remote contact or null if the filetransfer is an outgoing group file transfer.
FILE	String (not null)	URI of the file
FILENAME	String (not null)	Filename
MIME_TYPE	String (not null)	Multipurpose Internet Mail Extensions (MIME) type of message
FILEICON	String	URI of the file icon
FILEICON_MIME_TYPE	String	MIME type of the file icon

Data	Data type	Comment
DIRECTION	Integer	Incoming transfer or outgoing transfer. See enum Direction.
FILESIZE	Long	File size in bytes
TRANSFERRED	Long	Size transferred in bytes
TIMESTAMP	Long	Date of the transfer
TIMESTAMP_SENT	Long	Time when file is sent. If 0 means not sent.
TIMESTAMP_DELIVERED	Long	Time when file is delivered. If 0 means not delivered.
TIMESTAMP_ DISPLAYED	Long	Time when file is displayed.
STATE	Integer	See note below for the list of states
REASON_CODE	Integer	Reason code associated with the file transfer state. See enum FileTransfer, ReasonCode for possible reason codes.
READ_STATUS	Integer	See note below for the list of statuses

4.4.6.5 Permissions

Access to the File Transfer API is requires the following permissions:

- com.gsma.services.rcs.RCS_FILETRANSFER_RECEIVE: this is a new permission that is required by a client in order to handle the receipt of a file transferred from a remote party.

- com.gsma.services.rcs.RCS_FILETRANSFER_SEND: this is a new permission that is required by a client in order to initiate the transfer of a file transferred to a remote party.

- com.gsma.services.rcs.RCS_FILETRANSFER_READ: this is a new permission that is required by a client in order to read the file transfer history from the content provider.

4.4.7 Image Share API

This API exposes all functionality related to transferring images during a Circuit Switched (CS) call via the Image Share Service. It allows:

- Send an image share request
- Receive notifications about an incoming image share invitation and sharing events.
- Monitors an image share's progress.
- Cancel an image share in progress.
- Accept/reject an incoming image share request.
- Read configuration elements affecting image share.

4.4.7.1 Package

Package name **com.gsma.services.rcs.sharing.image**

4.4.7.2 Methods and Callbacks

Class **ImageSharingService**:

This class offers the main entry point to share images during a CS call, when the call hangs up the sharing is automatically stopped. Several applications may connect/disconnect to the API.

- Method: connects to the API.

```
void connect()
```

- Method: disconnects from the API.

```
void disconnect()
```

- Method: returns the list of image sharings in progress.

```
Set<ImageSharing> getImageSharings()
```

- Method: returns a current image sharing from its unique ID. If no ongoing ImageSharing matching the sharingId if found then a reference to a historical ImageSharing is returned so that calls to the methods on that still can be performed.

```
ImageSharing getImageSharing(String sharingId)
```

- Method: shares an image with a contact. The parameter file contains the URI of the image to be shared (for a local or a remote image). An exception if thrown if there is no on-going CS call.

```
ImageSharing shareImage(ContactId contact, Uri file)
```

- Method: returns the configuration for image share service.

```
ImageSharingServiceConfiguration getConfiguration()
```

- Method: adds a new image share invitation listener.

```
void addEventListener(ImageSharingListener listener)
```

- Method: removes a new image share invitation listener.

```
void removeEventListener(ImageSharingListener listener)
```

- Method: deletes all image sharings from history and abort/reject corresponding sessions if such are ongoing.

```
void deleteImageSharings()
```

- Method: deletes image sharings with a given contact from history and abort/reject corresponding sessions if such are ongoing.

```
void deleteImageSharings(ContactId contact)
```

- Method: deletes an image sharing from its sharing ID from history and abort/reject corresponding sessions if such are ongoing.

```
void deleteImageSharing(String sharingId)
```

Class **ImageSharing**:

This class maintains the information related to an image share and offers methods to manage the sharing.

- Enum: the ImageSharing state.

```
enum State { INVITED(0), INITIATED(1), STARTED(2), ABORTED(3),
FAILED(4), TRANSFERRED(5), REJECTED(6), RINGING(7), ACCEPTING(8) }
```

- Enum: the reason code for the image sharing.

```
enum ReasonCode { UNSPECIFIED(0), ABORTED_BY_USER(1),
ABORTED_BY_REMOTE(2), ABORTED_BY_SYSTEM(3),
REJECTED_BY_SECONDARY_DEVICE(4), REJECTED_TIME_OUT(5)
REJECTED_LOW_SPACE(6), REJECTED_MAX_SIZE(7),
REJECTED_MAX_SHARING_SESSIONS(8), REJECTED_BY_USER(9),
REJECTED_BY_REMOTE(10), FAILED_INITIATION(11), FAILED_SHARING(12),
FAILED_SAVING(13) }
```

- Method: returns the sharing ID of the image sharing.

```
String getSharingId()
```

- Method: returns the remote contact.

```
ContactId getRemoteContact()
```

- Method: returns the URI of the file to be shared.

```
Uri getFile()
```

- Method: returns the filename of-the file to be shared.

```
String getFileName()
```

- Method: returns the size of the file to be shared (in bytes).

```
String getFileSize()
```

- Method: returns the MIME type of file to be shared.

```
String getMimeType()
```

- Method: returns the local timestamp of when the image sharing was initiated for outgoing image sharing or the local timestamp of when the image sharing invitation was received for incoming image sharings.

```
long getTimeStamp()
```

- Method: returns the state of the image share.

```
State getState()
```

- Method: returns the reason code of the image share.

```
ReasonCode getReasonCode()
```

- Method: returns the direction of the sharing.

```
com.gsma.services.rcs.RcsService.Direction getDirection()
```

- Method: accepts image share invitation.

```
void acceptInvitation()
```

- Method: rejects image share invitation.

```
void rejectInvitation()
```

- Method: aborts the sharing.

```
void abortSharing()
```

Class **ImageSharingListener**:

This class offers callback methods on image sharing events.

- Method: callback called when the image sharing state/reasonCode has been changed

```
void onStateChanged(ContactId contact, String sharingId,
ImageSharing.State state, ImageSharing.ReasonCode reasonCode)
```

- Method: callback called during the sharing progress.

```
void onProgressUpdate(ContactId contact, String sharingId, long
currentSize, long totalSize)
```

- Method: callback called when a delete operation completed that resulted in that one or several image sharings was deleted specified by the sharingIds parameter corresponding to a specific contact.

```
void onDeleted(ContactId contact, Set<String> sharingIds)
```

Class **ImageSharingServiceConfiguration**:

This class represents the particular configuration of the Image Sharing Service.

- Method: returns the file size warning of Image Sharing configuration. It returns 0 if this value was not set by the auto-configuration server (no need to warn).

```
long getWarnSize()
```

- Method: returns the max file size of the Image Sharing configuration. It can return 0 if this value was not set by the auto-configuration server.

```
long getMaxSize()
```

4.4.7.3 Intents

Intent broadcasted when a new image sharing invitation has been received. This Intent contains the following extra:

- “sharingId”: (String) unique ID of the image sharing.

```
com.gsma.services.rcs.ish.action.NEW_IMAGE_SHARING
```

4.4.7.4 Content Providers

A content provider is used to store the image sharing history persistently. There is one entry per image sharing.

Class **ImageSharingLog**:

Event log provider member id used when merging the data from this provider with other registered event log provider members data into a common cursor:

```
static final int HISTORYLOG_MEMBER_ID = 3
```

URI constant to be able to query the provider data (Note that only read operations are supported since exposing write operations would conflict with the fact that the stack is performing write operations internally to keep the data matching the current situation):

```
static final Uri CONTENT_URI =
"content://com.gsma.services.rcs.provider.imageshare/imageshare"
```

The “SHARING_ID” column is defined as the unique primary key and can be references with adding a path segment to the CONTENT_URI + “/” + <primary key>

Column name definition constants to be used when accessing this provider:

```
static final String SHARING_ID = "sharing_id"
static final String CONTACT = "contact"
static final String FILE = "file"
static final String FILENAME = "filename"
static final String MIME_TYPE = "mime_type"
static final String DIRECTION = "direction"
static final String FILESIZE = "filesize"
static final String TRANSFERRED = "transferred"
static final String TIMESTAMP = "timestamp"
static final String STATE = "state"
```

```
static final String REASON_CODE = "reason_code"
```

The content provider has the following columns:

IMAGESHARE

Data	Data type	Comment
SHARING_ID	String (primary key)	Unique sharing identifier
CONTACT	String (not null)	ContactId formatted number of the remote contact
FILE	String (not null)	URI of the file
FILENAME	String (not null)	Filename
MIME_TYPE	String (not null)	Multipurpose Internet Mail Extensions (MIME) type of file
DIRECTION	Integer	Incoming sharing or outgoing sharing. See enum Direction
FILESIZE	Long	File size in bytes
TRANSFERRED	Long	Size transferred in bytes
TIMESTAMP	Long	Date of the sharing
STATE	Integer	See enum ImageSharing.State for the list of states
REASON_CODE	Integer	Reason code associated with the image sharing state. enum ImageSharing.ReasonCode for the list of reason codes

4.4.7.5 Permissions

Access to the Image Share API is requires the following permissions:

- com.gsma.services.rcs.RCS_IMAGESHARE_RECEIVE: this is a new permission that is required by a client in order to handle the receipt of an image shared by a remote party.
- com.gsma.services.rcs.RCS_IMAGESHARE_SEND: this is a new permission that is required by a client in order to initiate the sharing of an image with a remote party.
- com.gsma.services.rcs.RCS_IMAGESHARE_READ: this is a new permission that is required by a client in order to read the image share history from the content provider.

4.4.8 Video Share API

This API exposes all functionality related to sharing a live video stream during a CS call via the Video Share Service. It allows:

- Send a video share request.
- Receive notifications about incoming video share invitation and sharing events.
- Cancel an on-going video share.
- Accept/reject an incoming video share request.
- Read configuration elements affecting video share.

- Can use an external codec for video share.
- External codec: Programmer's externally customized codec.

4.4.8.1 Package

Package name **com.gsma.services.rcs.sharing.video**

4.4.8.2 Methods and Callbacks

Class **VideoSharingService**:

This class offers the main entry point to share a live video during a CS call, when the call hangs up the sharing is automatically stopped. Several applications may connect/disconnect to the API.

- Method: connects to the API.

```
void connect()
```

- Method: disconnects from the API.

```
void disconnect()
```

- Method: returns the list of video sharing in progress.

```
Set<VideoSharing> getVideoSharings()
```

- Method: returns a video sharing from its unique ID. If no ongoing VideoSharing matching the sharingId if found then a reference to a historical VideoSharing is returned so that calls to the methods on that still can be performed.

```
VideoSharing getVideoSharing(String sharingId)
```

- Method: shares a live video stream with a contact. The parameter player contains a media player which streams over RTP the live video from the camera. The media player is an interface which permits to have a player implementation independent from the RCS API. An exception is thrown if there is no on-going CS call. It's for the external codec.

```
VideoSharing shareVideo(ContactId contact, VideoPlayer player)
```

- Method: returns the configuration for video share service.

```
VideoSharingServiceConfiguration getConfiguration()
```

- Method: adds a video share event listener.

```
void addEventListener(VideoSharingListener listener)
```

- Method: removes a video share event listener.

```
void removeEventListener(VideoSharingListener listener)
```

- Method: deletes all video sharings from history and abort/reject corresponding sessions if such are ongoing.

```
void deleteVideoSharings()
```

- Method: deletes video sharings associated with a given contact from history and abort/reject corresponding sessions if such are ongoing.

```
void deleteVideoSharings(ContactId contact)
```

- Method: deletes a videosharing from its sharing ID from history and abort/reject corresponding sessions if such are ongoing.

```
void deleteVideoSharing(String sharingId)
```

Class **VideoSharing**:
This class maintains the information related to a video sharing and offers methods to manage the sharing.

```
public class Encoding {
      static String H264 = "H264";
}
```

- Enum: the VideoSharing state.

```
enum State { INVITED(0), INITIATED(1), STARTED(2), ABORTED(3),
FAILED(4), REJECTED(5), RINGING(6), ACCEPTING(7) }
```

- Enum: the reason code for the video sharing.

```
enum ReasonCode { UNSPECIFIED(0), ABORTED_BY_USER(1),
ABORTED_BY_REMOTE(2), ABORTED_BY_SYSTEM(3),
REJECTED_BY_SECONDARY_DEVICE(4), REJECTED_MAX_SHARING_SESSIONS(5),
REJECTED_BY_USER(6) REJECTED_BY_REMOTE(7), REJECTED_TIME_OUT(8),
FAILED_INITIATION(9), FAILED_SHARING(10) }
```

- Enum: The orientation of the video share.

```
enum Orientation { ANGLE_0(0), ANGLE_90(1), ANGLE_180(2),
ANGLE_270(3) }
```

- Method: returns the sharing ID of the video sharing.

```
String getSharingId()
```

- Method: returns the remote contact.

```
ContactId getRemoteContact()
```

- Method: get the remote video descriptor in case the VideoShare direction is incoming, the local video descriptor in use in case of outgoing direction.

```
VideoDescriptor getVideoDescriptor()
```

- Method: returns the state of the video share.

```
State getState()
```

- Method: returns the reason code of the video sharing.

```
ReasonCode getReasonCode()
```

- Method: returns the direction of the sharing:

```
com.gsma.services.rcs.RcsService.Direction getDirection()
```

- Method: accepts video share invitation with a given renderer. It's for the external codec.

```
void acceptInvitation(VideoPlayer player)
```

- Method: rejects video share invitation.

```
void rejectInvitation()
```

- Method: aborts the sharing.

```
void abortSharing()
```

- Method: sets the orientation of the video. It will return true if setting the orientation is succeeded otherwise false.

```
void setOrientation(Orientation orientation)
```

- Method: returns the encoding of the video sharing.

```
String getVideoEncoding()
```

- Method: returns the local timestamp of when the video sharing was initiated for outgoing video sharing or the local timestamp of when the video sharing invitation was received for incoming video sharings.

```
long getTimeStamp()
```

- Method: returns the duration of the video sharing.

```
long getDuration()
```

Class **VideoSharingListener**:
This class offers callback methods on video sharing events.

- Method : Callback called when the sharing state/reasonCode is changed.

```
void onStateChanged(ContactId contact, String sharingId,
VideoSharing.State state, VideoSharing.ReasonCode reasonCode)
```

- Method: callback called when the video descriptor was changed. This is done as a result of a call to VideoShare.setOrientation and VideoShare.setCamera for outgoing VideoShares or when remote VideoDescriptor was change for incoming VideoShares.

```
void onVideoDescriptorChanged(ContactId contact, String sharingId,
VideoDescriptor descriptor)
```

- Method: callback called when a delete operation completed that resulted in that one or several video sharings was deleted specified by the sharingIds parameter corresponding to a specific contact.

```
void onDeleted(ContactId contact, Set<String> sharingIds)
```

Class **VideoDescriptor**:
Class represents an object for video share parameters.

- Constructor: public constructor of a VideoDescriptor.

```
VideoDescriptor(Orientation orientation, int width, int height)
```

- Method: returns the orientation of the video stream.

```
Orientation getOrientation()
```

- Method: returns the width of video frame.

```
int getWidth()
```

- Method: returns the height of video frame.

```
int getHeight()
```

abstract Class **VideoPlayer**:
This class offers an interface to manage the video player instance independently of the RCS service. The video player is implemented on the application side.

- Method: returns the codec information, remoteHost, remotePort as a result of codec negotiation

```
void setRemoteInfo(VideoCodec codec, String remoteHost, int
remotePort)
```

- Method: returns the local RTP port used to stream video.

```
int getLocalRtpPort()
```

- Method: gets the Video Codec

```
VideoCodec getCodec()
```

- Method: returns the list of codecs supported by the player.

```
VideoCodec[] getSupportedCodecs()
```

Class **VideoCodec**:
This class maintains the information related to a video codec.

- Constructor : public constructor of a VideoCodec.

```
VideoCodec(String encoding, int payload, int clockRate, int
frameRate, int bitRate, VideoDescriptor descriptor, String
parameters)
```

- Method: returns the encoding name (e.g. H264).

```
String getEncoding()
```

- Method: returns the codec payload type (e.g. 96).

```
int getPayloadType()
```

- Method: returns the codec clock rate (e.g. 90000).

```
int getClockRate()
```

- Method: returns the codec frame rate (e.g. 10).

```
int getFrameRate()
```

- Method: returns the codec bit rate (e.g. 64000).

```
int getBitRate()
```

- Method: returns the VideoDescriptor.

```
VideoDescriptor getVideoDescriptor()
```

- Method: Returns the list of codec parameters (e.g. profile-level-id, packetization-mode). Parameters are are semicolon separated.

```
String getParameters()
```

Class **VideoSharingServiceConfiguration**:
This class represents the particular configuration of Video Sharing Service.

- Method: returns the maximum authorized duration of the content that can be shared in a VSH session. It can return null if this value was not set by the auto-configuration server.

```
int getMaxTime()
```

4.4.8.3 Intents

Intent broadcasted when a new video sharing invitation has been received. This Intent contains the following extra:

- “sharingId”: unique ID of the sharing.

```
com.gsma.services.rcs.vsh.action.NEW_VIDEO_SHARING
```

4.4.8.4 Content Providers

A content provider is used to store the video sharing history persistently. There is one entry per video sharing.

Class **VideoSharingLog**:

Event log provider member id used when merging the data from this provider with other registered event log provider members data into a common cursor:

```
static final int HISTORYLOG_MEMBER_ID = 4
```

URI constant to be able to query the provider data (Note that only read operations are supported since exposing write operations would conflict with the fact that the stack is performing write operations internally to keep the data matching the current situation):

```
static final Uri CONTENT_URI =
"content://com.gsma.services.rcs.provider.videoshare/videoshare"
```

The “SHARING_ID” column below is defined as the unique primary key and can be references with adding a path segment to the CONTENT_URI + “/” + <primary key>

Column name definition constants to be used when accessing this provider:

```
static final String SHARING_ID = "sharing_id"
static final String CONTACT = "contact"
static final String DIRECTION = "direction"
static final String TIMESTAMP = "timestamp"
static final String STATE = "state"
static final String REASON_CODE = "reason_code"
static final String DURATION = "duration"
static final String VIDEO_ENCODING = "video_encoding"
static final String WIDTH = "width"
static final String HEIGHT = "height"
static final String ORIENTATION = "orientation"
```

The content provider has the following columns:

VIDEOSHARE

Data	Data type	Comment
SHARING_ID	String (primary key)	Unique sharing identifier
CONTACT	String (not null)	ContactId formatted number of the remote contact

Data	Data type	Comment
DIRECTION	Integer	Incoming sharing or outgoing sharing. See enum Direction
TIMESTAMP	Long	Date of the sharing
STATE	Integer	See enum VideoSharing.State for the list of states
REASON_CODE	Integer	Reason code associated with the video sharing state. See enum VideoSharing.ReasonCode for the list of reason codes
DURATION	Long	Duration of the sharing in seconds. The value is only set at the end of the sharing.
VIDEO_ENCODING	String	Encoding of the shared video
WIDTH	Integer	Width of the shared video
HEIGHT	Integer	Height of the shared video
ORIENTATION	Integer	See enum VideoShare.Orientation for the list of orientations.

4.4.8.5 Permissions

Access to the Video Share API requires the following permissions:

- com.gsma.services.rcs.RCS_VIDEOSHARE_RECEIVE: this is a new permission that is required by a client in order to handle the receipt of a video shared by a remote party.
- com.gsma.services.rcs.RCS_VIDEOSHARE_SEND: this is a new permission that is required by a client in order to initiate the sharing of a video with a remote party.
- com.gsma.services.rcs.RCS_VIDEOSHARE_READ: this is a new permission that is required by a client in order to read the video share history from the content provider.

4.4.9 Geoloc Share API

This API exposes all functionality related to share a geoloc during a CS call via the Geoloc Share Service. It allows to:

- Send a geoloc share request
- Receive notifications about incoming geoloc share invitation.
- Monitors a geoloc share's progress.
- Cancel a geoloc share in progress.
- Accept/reject an incoming geoloc share request.

A geoloc contains the following information:

- a label associated to the geoloc info
- latitude
- longitude
- accuracy of the geoloc info (in meter)
- an expiration date of the geoloc info

The shared geoloc is displayed to the end user and also stored in sthe Chat log in order to be displayed afterwards from the “Show us in a map” service.

4.4.9.1 Package

Package name com.gsma.services.rcs.sharing.geoloc

4.4.9.2 Methods and Callbacks

Class **GeolocSharingService:**

This class offers the main entry point to share a geoloc during a CS call, when the call hangs up the sharing is automatically stopped. Several applications may connect/disconnect to the API.

- Method: connects to the API.

```
void connect()
```

- Method: disconnects from the API.

```
void disconnect()
```

- Method: returns the list of geoloc sharing in progress.

```
Set<GeolocSharing> getGeolocSharings()
```

- Method: returns a current geoloc sharing from its unique ID. If no ongoing GeolocSharing matching the sharingId if found then a reference to a historical GeiolocSharing is returned so that calls to the methods on that still can be performed.

```
GeolocSharing getGeolocSharing(String sharingId)
```

- Method: shares a geoloc with a contact. An exception is thrown if there is no ongoing CS call.

```
GeolocSharing shareGeoloc(ContactId contact, Geoloc geoloc)
```

- Method: adds a new geoloc sharing invitation listener.

```
void addEventListener(GeolocSharingListener listener)
```

- Method: removes a new geoloc sharing invitation listener.

```
void removeEventListener(GeolocSharingListener listener)
```

- Method: deletes all geoloc sharings from history and abort/reject corresponding sessions if such are ongoing.

```
void deleteGeolocSharings()
```

- Method: deletes geoloc sharings with a given contact from history and abort/reject corresponding sessions if such are ongoing.

```
void deleteGeolocSharings(ContactId contact)
```

- Method: deletes a geoloc sharing from its sharing ID from history and abort/reject corresponding sessions if such are ongoing.

```
void deleteGeolocSharing(String sharingId)
```

Class **GeolocSharing:**

This class maintains the information related to a geoloc sharing and offers methods to manage the sharing.

- Enum: the GeolocationSharing state.

```
enum State { INVITED(0), INITIATED(1), STARTED(2), ABORTED(3),
FAILED(4), TRANSFERRED(5), REJECTED(6), RINGING(7), ACCEPTING(8) }
```

- Enum: the reason code for the GeolocationSharing

```
enum ReasonCode { UNSPECIFIED(0), ABORTED_BY_USER(1),
ABORTED_BY_REMOTE(2), ABORTED_BY_SYSTEM(3),
REJECTED_BY_SECONDARY_DEVICE (4), REJECTED_MAX_SHARING_SESSIONS(5),
REJECTED_BY_USER(6),  REJECTED_BY_REMOTE(7), REJECTED_TIME_OUT(8),
FAILED_INITIATION(9), FAILED_SHARING(10) }
```

- Method: returns the sharing ID of the geoloc sharing.

```
String getSharingId()
```

- Method: returns the remote contact.

```
ContactId getRemoteContact()
```

- Method: returns the geoloc info to be shared.

```
Geoloc getGeoloc()
```

- Method: returns the local timestamp of when the geoloc sharing was initiated for outgoing geoloc sharing or the local timestamp of when the geoloc sharing invitation was received for incoming geoloc sharings.

```
long getTimeStamp()
```

- Method: returns the state of the geoloc sharing.

```
State getState()
```

- Method: returns the reason code of the geoloc sharing.

```
ReasonCode getReasonCode()
```

- Method: returns the direction of the sharing:

```
com.gsma.services.rcs.RcsService.Direction getDirection()
```

- Method: accepts geoloc sharing invitation.

```
void acceptInvitation()
```

- Method: rejects geoloc sharing invitation.

```
void rejectInvitation()
```

- Method: aborts the sharing.

```
void abortSharing()
```

Class **GeolocSharingListener:**

This class offers callback methods on geoloc sharing events.

- Method: callback called when the geoloc sharing state is changed.

```
void onStateChanged(ContactId contact, String sharingId,
GeolocationSharing.State state, GeolocSharing.ReasonCode reasonCode)
```

- Method: callback called during the sharing progress.

```
void onProgressUpdate(ContactId contact, String sharingId, long
currentSize, long totalSize)
```

- Method: callback called when a delete operation completed that resulted in that one or several geoloc sharings was deleted specified by the sharingIds parameter corresponding to a specific contact.

```
void onDeleted(ContactId contact, Set<String> sharingIds)
```

4.4.9.3 Intents

Intent broadcasted when a new geoloc sharing invitation has been received. This Intent contains the following extras:

- “sharingId”: unique ID of the geoloc sharing.

```
com.gsma.services.rcs.gsh.action.NEW_GEOLOC_SHARING
```

4.4.9.4 Content Providers

A content provider is used to store the geolocation sharing history persistently. There is one entry per geolocation sharing.
Class **GeolocSharingLog**:

Event log provider member id used when merging the data from this provider with other registered event log provider members data into a common cursor:

```
static final int HISTORYLOG_MEMBER_ID = 5
```

URI constant to be able to query the provider data (Note that only read operations are supported since exposing write operations would conflict with the fact that the stack is performing write operations internally to keep the data matching the current situation):

```
static final Uri CONTENT_URI =
"content://com.gsma.services.rcs.provider.geolocshare/geolocshare"
```

The “SHARING_ID” column below is defined as the unique primary key and can be references with adding a path segment to the CONTENT_URI + “/” + <primary key>

Column name definition constants to be used when accessing this provider:

```
static final String SHARING_ID = "sharing_id"
static final String CONTACT = "contact"
static final String GEOLOC = "geoloc"
static final String MIME_TYPE = "mime_type"
static final String DIRECTION = "direction"
static final String TIMESTAMP = "timestamp"
static final String STATE = "state"
static final String REASON_CODE = "reason_code"
```

The content provider has the following columns:

GEOLOCSHARE

Data	Data type	Comment
SHARING_ID	String (primary key)	Unique sharing identifier
CONTACT	String (not null)	ContactId formatted number of the remote contact
GEOLOC	String (not null)	The geolocation stored as a String parseable with the ChatLog.Message.getGeoloc() method.
MIME_TYPE	String (not null)	Multipurpose Internet Mail Extensions (MIME) type of the geoloc
DIRECTION	Integer	Direction of sharing. See enum Direction.
TIMESTAMP	Long	Date of the sharing
STATE	Integer	See enum GeolocSharing.State for valid states
REASON_CODE	Integer	See enum GeolocSharing.ReasonCode for valid reason codes

4.4.9.5 Permissions

Geoloc Share is a convenience mechanism to allow geolocation information to be delivered in a chat message. From the perspective of a client receiving such events, the permissions are no different from those relating to any other chat message. On the sending side, permissions are defined that govern the ability of a client to access geolocation information, and to send that information via the Geoloc Share mechanism.

Access to the Geoloc API is requires the following permissions:

- android.permission.ACCESS_FINE_LOCATION: this is the standard Android permission that governs whether or not the app is entitled to access fine-grained geolocation information such as might be available from GPS.

- android.permission.ACCESS_COARSE_LOCATION: this is the standard Android permission that governs whether or not the app is entitled to access coarse-grained geolocation information such as might be available from CellID or WiFi sources.

- com.gsma.services.rcs.RCS_LOCATION_SEND: this is a new permission that is required to send Geolocation data over an RCS chat session.

- com.gsma.services.rcs.RCS_USE_CHAT: this is the permission that governs access to the chat API which is a prerequisite to being able to use the Geoloc Share API.

- com.gsma.services.rcs.RCS_GEOLOCSHARE_READ: this is a new permission that is required by a client in order to read the geolocation share history from the content provider.

4.4.10 Contacts API

There is already an Android API to manage contacts of the local address book, see Android package **android.provider.ContactsContract.** This API offers additional methods to:

- Add RCS info in the local address book,
- Extract RCS info from the local address book.

4.4.10.1 Package

Package name **com.gsma.services.rcs.contacts**

4.4.10.2 Methods and Callbacks

Class **ContactsService**:

This class offers methods to extract RCS info associated to contacts from the local address book.

- Method: connects to the API.

```
void connect()
```

- Method: disconnects from the API.

```
void disconnect()
```

- Method: returns the list of RCS contacts.

```
Set<RcsContact> getRcsContacts()
```

- Method: Returns the RCS contact infos from its contact ID (i.e. MSISDN)

```
RcsContact getRcsContact(ContactId contact)
```

- Method: returns the list of contacts online (i.e. registered).

```
Set<RcsContact> getRcsContactsOnline()
```

- Method: returns the list of contacts supporting a given extension or service ID (i.e. capability).

```
Set<RcsContact> getRcsContactsSupporting(String serviceId)
```

- Method: get the vCard of a contact. The parameter contact contains the database URI of the contact in the native address book. The method returns the complete filename including the path of the visit card. The filename has the file extension ".vcf" and is generated from the native address book vCard URI (see Android SDK attribute `ContactsContract.Contacts.CONTENT_VCARD_URI` which returns the referenced contact formatted as a vCard when opened through `openAssetFileDescriptor(Uri, String)`).

```
static String getVCard(Context ctx, Uri contact)
```

Class **ContactUtils**:

This class offers utility methods to verify and format contact identifier.

- Method: get an instance of ContactUtils. May be null if country code cannot be read from provider.

```
static ContactUtils getInstance(Context context)
```

- Method: returns true if the given contactId have the syntax of valid RCS contactId.If the string is too short (1 digit at least), too long (more than 15 digits) or contains illegal characters (valid characters are digits, space, '-', leading '+') then it returns false.

```
boolean isValidContact(String contact)
```

- Method: formats the given contact to uniquely represent a RCS contact. If the input string is not valid - as can be tested with the method isValidContact() - then RcsContactFormatException is thrown else a valid ContactId is returned.

```
ContactId formatContact (String contact)
```

Class **ContactId:**

This class represents a formatted and valid contact number. All normal java object methods are supported for this class like toString(), equals(), hashCode()…

NOTE : the contact format is the international representation of the phone number "<CC><NDC><SN>" with:

CC : the country code with a leading '+' character

NDC : the national destination code

SN: the subscriber number

All these codes CC, NDC, SN are digits. If this number needs to be displayed in the UI with some particular UI formatting, it is in the scope of UI code to format that. This class will never hold specific UI formatted numbers since they need to be unique.

Class **RcsContact**:

This class maintains the information related to a RCS contact.

- Method: returns the canonical contact ID (i.e. MSISDN).

```
ContactId getContactId()
```

- Method: returns the displayname associated to the contact.

```
String getDisplayName()
```

- Method: returns the capabilities associated to the contact.

```
Capabilities getCapabilities()
```

- Method: is contact online (i.e. registered to the service platform).

```
boolean isRegistered()
```

4.4.10.3 Content Providers

In addition to the methods, the RCS information are stored in the local address book thanks to the Contacts Contract interface of the Android Software Development Kit (SDK). This permits to have a native integration of RCS in the address book.

See the following MIME-type to be supported:

MIME type	Comment
vnd.android.cursor.item/com.gsma.services.rcs.number	RCS phone number
vnd.android.cursor.item/com.gsma.services.rcs.registration-state	Registration state (online | offline)
vnd.android.cursor.item/com.gsma.services.rcs.image-share	Image share capability supported
vnd.android.cursor.item/com.gsma.services.rcs.video-share	Video share capability supported
vnd.android.cursor.item/com.gsma.services.rcs.im-session	IM/Chat capability supported
vnd.android.cursor.item/com.gsma.services.rcs.file-transfer	File transfer capability supported
vnd.android.cursor.item/com.gsma.services.rcs.extensions	RCS extensions supported
vnd.android.cursor.item/com.gsma.services.rcs.geoloc-push	Geolocation push capability supported

Implementation notes:

- To store the MIME-type see the following tutorial http://developer.android.com/reference/android/provider/ContactsContract.RawContacts.html
- A raw contact is created to store the RCS info associated to a contact. A RCS account is created to manage raw contacts.

- When a contact becomes enriched with RCS information, we associate a corresponding raw contact with MIME type vnd.android.cursor.item/vnd.rcs.
- The number associated to the contact is put into the field Data.DATA1.
- The supported MIME type is put into the field Data.MIMETYPE.
- The description associated to the supported MIME type is always put into the field Data.DATA2. This label is displayed at UI level (i.e. menu item of the local native address book).
- If a MIME type is not set for a contact, this means the associated capability is not supported.

4.4.10.4 Permissions

Access to the Contacts API requires the following permissions:

- android.permission.READ_CONTACTS: this permission is required by any client using the capabilities service, since use of the API implicitly reveals information about past and current contacts for the device.
- Additionally, methods that reveal contact capabilities (getRcsContactsSupporting() and getCapabilities()) require:
- com.gsma.services.rcs.RCS_READ_CAPABILITIES: this is a new permission that governs access to capability information.

4.4.11 API Versioning

This API maintains information about the current version of the RCS terminal API.

A build is identified by:

- GSMA version: hotfixes, Blackbird, .etc.
- Implementer name: entity name who has implemented the API.
- Release number of the API.
- Incremental number to identify the build into a release number.

A software release of the API is identified uniquely by its release number and the incremental number.

4.4.11.1 Package

Package name **com.gsma.services.rcs**

4.4.11.2 Methods and Callbacks

Class **Build**:

This class offers information related to the build version of the API.

- Constant: API release implementer name.

```
public final static String API_CODENAME
```

- Constant: API version number from class Build.VERSION_CODES.

```
public final static int API_VERSION.
```

- Constant: Internal number used by the underlying source control to represent this build.

```
public final static int API_INCREMENTAL
```

Class **Build.VERSION_CODES**:

This class contains the list of API versions.

- Constant: The original first version of RCS API or hotfixes version.

```
public final static int BASE
```

- Constant: Blackbird version

```
public final static int BLACKBIRD
```

- Constant: Crane version

```
public final static int CRANE
```

4.4.11.3 Content Providers

4.4.12 Multimedia Session API

This service API exposes all functionality related to initiate a multimedia session between two clients in order to implement a new IMS service based session. The new service is identified by a unique service ID which corresponds to an IARI feature tag and ICSI tag in the signalling flows, the same service ID is used as an extension in the Capability service API.

There are 2 types of services offered by the API:

- Real time messaging session based on the MSRP protocol for media. A session is established between contacts and multimedia messages or data are exchanged in real time while the session exists. A session exists from its initiation to its termination.
- Real time streaming session based on the RTP protocol for media. A session is established between contacts and multimedia payloads are streamed in real time while the session exists. A session exists from its initiation to its termination.

The session may be accepted or rejected by the remote contact. Any type of message may be exchanged between end points.

The API allows:

- Initiate a multimedia session for messaging or streaming.
- Accept/reject an incoming session invitation.
- Retrieve the list of sessions using a given feature tag.
- Terminate a session.

For a given service, several sessions may coexist in parallel.

This service API hides:

- the SIP signalling complexity and SDP (Session Description Protocol) answer/offer mechanism to negotiate the media exchanged between the two end points.

- The media protocol (MSRP for messaging and RTP for streaming).

Thanks to this API, any application can implement a new RCS/IMS service on top of the RCS background service which maintains a single attachment to the RCS/IMS platform and utilizes common IMS procedures (e.g. authentication) for services implemented on top of it.

Each new RCS/IMS service is associated to a service ID:

- The service ID is used to define a new capability (see Capability API) and to share it with others remote contacts.
- The service ID is used to identify the service in the device (for incoming and outgoing request), but also on the platform side (e.g. to trigger an Application server).

4.4.12.1 Package

Package name **com.gsma.services.rcs.extension**

4.4.12.2 Methods and Callbacks

Class **MultimediaSessionService:**

This class offers the main entry point to initiate and manage new and existing multimedia sessions. Several applications may connect/disconnect to the API.

- Method: connects to the API.

```
void connect()
```

- Method: disconnects from the API.

```
void disconnect()
```

- Method: Returns the configuration of the multimedia session service.

```
MultimediaSessionServiceConfiguration getConfiguration()
```

- Method: returns the list of messaging sessions in progress associated to a given service ID.

```
Set<MultimediaMessagingSession> getMessagingSessions(String
serviceId)
```

- Method: returns a messaging session in progress from its unique session ID.

```
MultimediaMessagingSession getMessagingSession(String sessionId)
```

- Method: initiate a new multimedia session for real time messaging with a remote contact and for a given service. The messages exchanged in real time during the session may be from any type.

```
MultimediaSession initiateMessagingSession(String serviceId,
ContactId contact)
```

- Method: returns the list of streaming sessions in progress associated to a given service ID.

```
Set<MultimediaStreamingSession> getStreamingSessions(String
serviceId)
```

- Method: returns a streaming session in progress from its unique session ID.

```
MultimediaStreamingSession getStreamingSession(String sessionId)
```

- Method: initiate a new multimedia session for real time streaming with a remote contact and for a given service. The payloads exchanged in real time during the session may be from any type.

```
MultimediaSession initiateStreamingSession(String serviceId,
ContactId contact)
```

- Method: deletes multimedia sessions from its session ID.

```
Void addEventListener(MultimediaMessagingSessionListener listener)
```

- Method: deletes multimedia sessions from its session ID.

```
void removeEventListener(MultimediaMessagingSessionListener listener)
```

- Method: deletes multimedia sessions from its session ID.

```
void addEventListener(MultimediaMessagingSessionListener listener)
```

- Method: deletes multimedia sessions from its session ID.

```
void removeEventListener(MultimediaMessagingSessionListener listener)
```

Class **MultimediaSession:**

This class maintains the information related to a multimedia session and offers methods to manage it. This is an abstract class between a messaging session and a streaming session.

- Enum: the state of the multimedia session.

```
enum State { INVITED(0), INITIATED(1), STARTED(2), ABORTED(3),
FAILED(4), REJECTED(5), RINGING(6), ACCEPTING(7) }
```

- Enum: the reason code for the multimedia session.

```
enum ReasonCode  { UNSPECIFIED(0), REJECTED_TIME_OUT(1),
REJECTED_BY_USER(2), REJECTED_BY_REMOTE(3), FAILED_SESSION(4),
FAILED_MEDIA(5) }
```

- Method: returns the session ID of the multimedia session.

```
String getSessionId()
```

- Method: returns the remote contact.

```
ContactId getRemoteContact()
```

- Method: returns the service ID.

```
String getServiceId()
```

- Method: returns the state of the session.

```
State getState()
```

- Method: returns the reason code of the session.

```
ReasonCode getReasonCode()
```

- Method: returns the direction of the session:

```
com.gsma.services.rcs.RcsService.Direction getDirection()
```

- Method: accepts session invitation.

```
void acceptInvitation()
```

- Method: rejects session invitation.

```
void rejectInvitation()
```

- Method: aborts the session.

```
void abortSession()
```

Class **MultimediaMessagingSession:**

This class inherits from the class MultimediaSession and is related to a messaging session.

- Method: send a multimedia message or data in real time.

```
void sendMessage(byte[] content)
```

Class **MultimediaMessagingSessionListener:**

This class offers callback methods on multimedia session events.

- Method: Callback called when the multimedia messaging session state/reasonCode is changed

```
void onStateChanged(ContactId contact, String sessionId,
MultimediaMessagingSession.State state,
MultimediaMessagingSession.ReasonCode reasonCode)
```

- Method: callback called when a multimedia message or data is received.

```
void onMessageReceived(ContactId contact, String sessionId, byte[]
content)
```

Class **MultimediaStreamingSession:**

This class inherits from the class MultimediaSession and is related to a streaming session.

- Method: send a multimedia payload or data in real time.

```
void sendPayload(byte[] content)
```

Class **MultimediaStreamingSessionListener:**

This class offers callback methods on multimedia session events.

- Method: Callback called when the multimedia messaging session state/reasoncode is changed

```
void onStateChanged(ContactId contact, String sessionId,
MultimediaStreamingSession.State state,
MultimediaStreamingSession.ReasonCode reasonCode)
```

- Method: callback called when a multimedia message or data is received.

```
void onPayloadReceived(ContactId contact, String sessionId, byte[]
content)
```

Class **MultimediaSessionServiceConfiguration:**

This class represents the particular configuration of Multimedia Service.

- Method: Return maximum length of a multimedia message

```
int getMessageMaxLength()
```

4.4.12.3 Intents

Intent broadcasted when a new messaging session invitation has been received. This Intent contains the following extra:

- “sessionId”: (String) unique ID of the multimedia session.

```
com.gsma.services.rcs.extension.action.NEW_MESSAGING_SESSION
```

Intent broadcasted when a new streaming session invitation has been received. This Intent contains the following extras:

- “sessionId”: (String) unique ID of the multimedia session.

```
com.gsma.services.rcs.extension.action.NEW_STREAMING_SESSION
```

A service is identified by its service ID which permits to route the incoming request (i.e. SIP INVITE) to the corresponding Android application on the device. This mechanism is implemented by declaring an Intent filter in the Manifest file of the application. See the following syntax to be used, where “xxx” corresponds to the service ID:

```
<intent-filter>

  <action
android:name="com.gsma.services.rcs.extension.action.NEW_MESSAGING_SE
SSION"/>

  <data android:mimeType="com.gsma.services.rcs/xxx"/>
 <category android:name="android.intent.category.LAUNCHER"/>
 <category android:name="android.intent.category.DEFAULT"/>

</intent-filter>


<intent-filter>

  <action
android:name="com.gsma.services.rcs.extension.NEW_STREAMING_SESSION"/
>

  <data android:mimeType="com.gsma.services.rcs/xxx"/>
 <category android:name="android.intent.category.LAUNCHER"/>
 <category android:name="android.intent.category.DEFAULT"/>

</intent-filter>
```

See the Capability API for the service ID syntax.

So when an incoming SIP request arrives in the RCS background service, the feature tag of the request is read and analyzed in order to broadcast an Intent containing the feature tag in its MIME type. Then the Intent is captured by the corresponding application.

4.4.13 File Upload API

This API exposes all functionality related to upload a file to the RCS Content Server. It allows:

- Upload a file to the Content Server over HTTP.
- Get info on the uploaded file in order to share the file link via any solution (SMS, Chat, Multimedia Session, .etc).
- Monitor the upload progress.
- Abort the upload.

4.4.13.1 Package

Package name **com.gsma.services.rcs.upload**

4.4.13.2 Methods and Callbacks

Class **FileUploadService**:

This class offers the main entry point to upload files to the Content Server. Several files may be uploaded at a time. Several applications may connect/disconnect to the API.

- Method: connects to the API.

```
void connect()
```

- Method: disconnects from the API.

```
void disconnect()
```

- Method: returns the list of file uploads in progress.

```
Set<FileUpload> get FileUploads()
```

- Method: returns a file upload in progress from its unique ID.

```
FileUpload getFileUpload(String uploadId)
```

- Method: Uploads a file to the RCS content server. The parameter file contains the URI of the file to be uploaded (for a local or a remote file). The parameter fileicon defines if the stack shall try to generate a thumbnail. If the max number of simultaneous uploads is achieved an exception is thrown. If the max size of a file upload is achieved an exception is thrown.

```
FileUpload uploadFile(Uri file, boolean fileicon)
```

- Method: returns true if a file can be uploaded right now using the uploadFile method.

```
boolean canUploadFile()
```

- Method: adds an event listener on file upload events.

```
void addEventListener(FileUploadListener listener)
```

- Method: removes an event listener from file upload.

```
void removeEventListener(FileUploadListener listener)
```

- Method: returns the configuration for the File Upload service.

```
FileUploadServiceConfiguration getConfiguration()
```

Class **FileUploadServiceConfiguration**:

This class represents the particular configuration of the File Upload Service (this the same parameter values as for FT Service).

- Method: returns the max file size of a file upload. It can return 0 if there is no limitation.

```
long getMaxSize()
```

Class **FileUpload**:

This class maintains the information related to a file upload and offers methods to monitor the upload.

- Enum: the FileUpload state.

```
enum State { INACTIVE(0), STARTED(1), ABORTED(2), FAILED(3),
TRANSFERRED(4) }
```

- Method: returns the upload ID of the upload.

```
String getUploadId()
```

- Method: returns the URI of the file to be uploaded.

```
Uri getFile()
```

- Method: returns the state of the upload.

```
State getState()
```

- Method: returns info related to the uploaded file on the content server.

```
FileUploadInfo getUploadInfo()
```

- Method: aborts the upload.

```
void abortUpload()
```

Class **FileUploadInfo**:

This class contains information related to the file uploaded on the content server.

- Method: returns URI of the file on the content server.

```
Uri getFile()
```

- Method: returns the validity of the file on the content server.

```
long getValidity()
```

- Method: returns the size of the file.

```
long getSize()
```

- Method: returns the original filename.

```
String getFilename()
```

- Method: returns the mime type of the file.

```
String getMimeType()
```

- Method: returns URI of the file icon on the content server.

```
Uri getFileicon()
```

- Method: returns the validity of the file icon on the content server.

```
long getFileiconValidity()
```

- Method: returns the size of the file icon.

```
long getFileiconSize()
```

- Method: returns the mime type of the file icon.

```
String getFileiconMimeType()
```

Class **FileUploadListener**:

This class offers callback methods on file upload events.

- Method: callback called when the file upload state has been changed.

```
void onStateChanged(String uploadId, FileUpload.State state)
```

- Method: callback called during the upload progress.

```
void onProgressUpdate(String uploadId, long currentSize, long
totalSize)
```

4.4.13.3 Permissions

Access to the File Upload API requires the following permissions:

- com.gsma.services.rcs.RCS_FILEUPLOAD_SEND: this is a new permission that is required by a client in order to upload a file to the content server.

4.4.14 Convergent historylog API

4.4.14.1 Package

Package name **com.gsma.services.rcs.history**

4.4.14.2 Methods and Callbacks

Class **HistoryService**:
This class offers the possibility to register/unregister additional history log provider members on top of those that the terminal API already added by default and which the history log provider supports data from to be presented as a merged cursor. The history log provider members that are added by default by the stack and thus need no registration by any application to be used are currently ChatLog.Message, FileTransferLog, ImageShareLog,

VideoShareLog and GeolocShareLog. Several applications may connect/disconnect to the API.

- Method: connects to the API.

  ```
  void connect()
  ```

- Method: disconnects from the API.

  ```
  void disconnect()
  ```

- Method: register an extra event log member so that the event log provider can merge data from that database together with other event log member's data. The column mapping parameter allows for mapping exactly how the columns in the registered provider should be mapped to the event log provider columns in the resulting cursor.

  ```
  void registerExtraHistoryLogMember(int providerId, Uri database,
  String table, Map<String, String> columnMapping)
  ```

- Method: unregister an external history log member so that it can no longer be used to join together the data from this member together with the other evenloh members.

  ```
  void unRegisterExtraHistoryLogMember(int providerId)
  ```

4.4.14.3 Content Providers

The content provider in this package is a virtual content provider in that it does not store any data itself but it allows for a client to make queries dynamically combining entries from several other specified providers per query returning a merged cursor containing all entries that match the selection query in those specified providers. Any normal query should be possible to make against the event log provider including specifying sort order, selection arguments as well as any projection of choice matching the data columns specified below. Operations of insert/update and delete has naturally been blocked in this provider as such operations are handled by other use cases and in each individual other provider that stores the actual data. Note that only read operations are supported.
Class **HistoryLog**:

Base URI constant to be able to query the provider data. Specific history log members ids needs to be appended to this base uri as query parameters to specify which members data should be merged (See HistoryLogUriBuilder):

```
static final Uri CONTENT_URI = "content://com.gsma.services.rcs.provider.
history/history"
```

Alternative URI constant to be able to make a query of the provider data that does not support parameters (like "?") but is more optimized for speed on larger datasets. Specific history log members ids needs to be appended to this uri as query parameters to specify which members data should be merged (See HistoryLogUriBuilder):

```
static final Uri CONTENT_URI_PARAMLESS =
"content://com.gsma.services.rcs.provider. history/history_paramless"
```

Column name definition constants to be used when accessing this provider:

```
static final String PROVIDER_ID = "provider_id"
```

```
static final String ID = "id"
static final String MIME_TYPE = "mime_type"
static final String DIRECTION = "direction"
static final String CONTACT = "contact”
static final String TIMESTAMP = "timestamp"
static final String TIMESTAMP_SENT = "timestamp_sent"
static final String TIMESTAMP_DELIVERED = "timestamp_delivered"
static final String TIMESTAMP_DISPLAYED = "timestamp_displayed"
static final String MIME_TYPE = "mime_type"
static final String STATUS = "status"
static final String REASON_CODE = "reason_code"
static final String READ_STATUS = "read_status"
static final String CHAT_ID = "chat_id"
static final String DIRECTION = "direction"
static final String CONTENT = "content"
static final String FILEICON = "fileicon"
static final String FILEICON_MIME_TYPE = "fileicon_mime_tyoe"
static final String FILENAME = "filename"
static final String FILESIZE = "filesize"
static final String TRANSFERRED = "transferred"
static final String DURATION = "duration"
```

The content provider exposes the following virtual table and virtual columns:

HISTORYLOG

Data	Data Type	Description
PROVIDER_ID	Integer	The id of the provider of the entry matching the id declared as a constant in that history log provider member (ex Chat.Message..HISTORYLOG_MEMBER_ID or FileTransfer.HISTORYLOG_MEMBER_ID)
ID	String	Identifier of the entry (“msg_id”, “ft_id” or “sharing_id” etc depending on the corresponding provider of the entry)
MIME_TYPE	String	Multipurpose Internet Mail Extensions (MIME) type of the entry
DIRECTION	Integer	See enum Direction
CONTACT	String	ContactId formatted number associated with the entry status. See corresponding provider for the list of reason codes
TIMESTAMP	Long	Time when entry was inserted

Data	Data Type	Description
TIMESTAMP_SENT	Long	Time when this entry was sent. 0 means not sent.
TIMESTAMP_DELIVERED	Long	Time when this entry was delivered. 0 means not delivered.
TIMESTAMP_DISPLAYED	Long	Time when this entry was displayed. 0 means not displayed.
STATUS	Integer	Status (or State) of the entry. See corresponding provider for available statuses and/or states
REASON_CODE	Integer	Reason code associated with the entry status. See corresponding provider for the list of reason codes
READ_STATUS	Integer	Read status (UNREAD or READ) matching the read status of the corresponding provider of the entry.
CHAT_ID	String	Id for chat room
CONTENT	String	Content of the message if this entry corresponds to a content message or the file uri if this entry is a file transfer, image share, geoloc share or video share etc.
FILEICON	String	File Icon Uri if the entry corresponds to a file transfer and it has a file icon attached to it
FILEICON_MIME_TYPE	String	Multipurpose Internet Mail Extensions (MIME) type of the file icon
FILENAME	String	File name if this entry corresponds to a file transfer
FILESIZE	Long	File size in bytes if this entry corresponds to a file transfer
TRANSFERRED	Long	Size transferred in bytes if this entry corresponds to a file transfer
DURATION	Long	Duration of the sharing or call in seconds if this entry corresponds to a sharing or a call. The value is only set at the end of the sharing or call.

4.4.14.4 Methods and Callbacks

Class **HistoryLogUriBuilder:**
This class offers methods to build an Uri that can be used to query the history log provider. The uri format is constructed by adding each provider id as standard uri query parameters to either the CONTENT_URI or the CONTENT_URI_PARAMLESS exposed in the HistoryLog class. Note order of added history log provider members in the uri is of no significance as sort order can be specified on the data in the returned cursor from the history log provider anyway when querying it.

- Constructor: Instantiates a new HistoryLogUriBuilder with a Uri of choice.

```
HistoryLogUriBuilder(Uri historyLogUri)
```

- Method: adds a registered history log provider member id to the builder instance. A maximum of ten members can be added in total.

```
HistoryLogUriBuilder appendProvider(int providerId)
```

- Method: returns an Uri containing the added providers.

```
Uri build()
```

Annex A Document Management

A.1 Document History

Version	Date	Brief Description of Change	Approval Authority	Editor / Company
0.1	26 Nov 2013	Joyn Blackbird release are incorporated	RCS TSG JTA	Kelvin Qin and Tom Van Pelt / GSMA
1.0	31 Jan 2014	Approved by PSMC	PSMC	Kelvin Qin / GSMA
1.5	10 Oct 2014	Multimeida API is added and some other API improvement	RCSJTA	Kelvin Qin / GSMA

A.2 Other Information

Type	Description
Document Owner	RCS TSG JTA
Editor / Company	Kelvin Qin / GSMA

It is our intention to provide a quality product for your use. If you find any errors or omissions, please contact us with your comments. You may notify us at prd@gsma.com

Your comments or suggestions & questions are always welcome.