RCS Device API 1.6 Specification
Version 4.0
01 March 2016

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 4
1.1 Overview 4
1.2 Scope 4
1.3 Definitions 4
1.4 Abbreviations 4
1.5 References 5
1.6 Conventions 5
2 API Architecture 5
2.1 Architecture Overview 5
2.1.1 API Descriptions 7
2.1.2 Applications Types 7
3 API concepts 8
3.1 Servers and Listeners 8
3.1.1 Service 8
3.1.2 Service Session 8
3.2 Service Version/Available/Unavailable 9
4 Android API 9
4.1 Components Interaction 9
4.1.1 New service application 9
4.1.2 Constraints 9
4.2 Security 9
4.2.1 Service API Access Control 10
4.3 UX API 10
4.3.1 Package 10
4.3.2 Methods and Callbacks 10
4.4 Services API 14
4.4.1 Overview 14
4.4.2 Access Control 14
4.4.3 Common architecture 14
4.4.4 Capability API 20
4.4.5 IM/Chat API 27
4.4.6 File Transfer API 42
4.4.7 Image Share API 54
4.4.8 Video Share API 59
4.4.9 Geoloc Share API 65
4.4.10 Contact API 70
4.4.11 API Versioning 74
4.4.12 Multimedia Session API 75
4.4.13 File Upload API 82
4.4.14 Convergent historylog API 85
Annex A Usage of Multimedia Session API to Implement Enriched Calling Services (Informative) 89

Annex B Document Management **90**
A.1 Document History 90
A.2 Other Information 90

1 Introduction

1.1 Overview

This document defines the architecture and a set of standardized Application Programming Interfaces (API) to develop joyn user experience (UX), use joyn 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 of Crane Priority Release (CPR) which is defined in Annex B [PRD RCC.62].

1.3 Definitions

Term	Description
3rd Party Applications	Applications that are not part of the joyn 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 joyn 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 joyn 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 joyn client.
IMS API	APIs that are exposed by the IMS Stack.
Standard Services	Services that are identified by feature tags, as defined by joyn Specification.

1.4 Abbreviations

Term	Description
AIDL	Android Interface Definition Language
API	Application Programming Interfaces
CD	Capability Discovery
CPR	Crane Priority Release
CS	Circuit Switched
FT	File Transfer
ID	Identifier
IM	Instant Messaging
IMS	IP Multimedia Sub-system
IS	Image Share

Term	Description
MIME	Multipurpose Internet Mail Extensions
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.62]	joyn Crane Product Definition Document Version 3.0 http://www.gsma.com/network2020/wp-content/uploads/2014/02/RCC.62_v3.0.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 joyn Client architecture is composed of several sub-systems, organized 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 joyn **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.

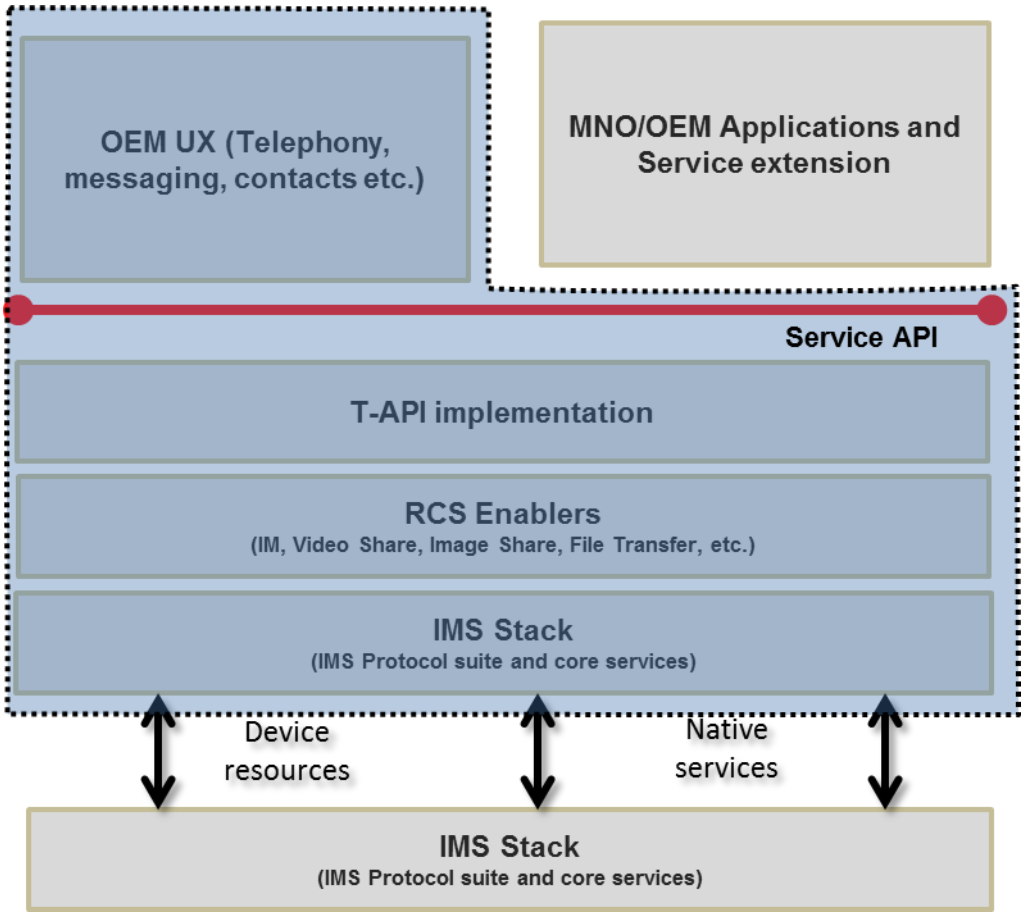

Figure 1: General Architecture Overview

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, Image Share and Geoloc Share functionality to be fully available, a call needs to be ongoing with a RCS contact possessing Video Share, Image Share and Geoloc Share capabilities respectively.

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 joyn functionality for capability fetching and retrieving parts of the joyn 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 joyn Specification.
- Core Services that are included in the joyn Client. These components must undergo GSMA accreditation as part of the joyn 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] for exact definitions of possible feature tags.

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.RCS service API exposes privacy-sensitive information or may trigger service charges, the user must grant a general permission for any application to use that API.

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 **RcsServiceControl:**
This class is a utility to control the activation of the RCS service.

- Constant: RCS stack package name

```
final static String RCS_STACK_PACKAGENAME = "com.gsma.rcs"
```

```
Method: returns a singleton instance of the RcsServiceControl
class.static RcsServiceControl getInstance(Context ctx)
```

- Method: returns true if the RCS stack is installed and not disabled on the device.

```
boolean isAvailable()
```

- Method: returns true if the RCS stack is marked as active on the device.

```
boolean isActivated()
```

- Method: returns true if the RCS stack deactivation/activation is allowed by the client.

```
boolean isActivationModeChangeable()
```

- Method: permits deactivation/activation of the RCS stack in case these operations are allowed (see isStackActivationStatusChangeable) or else throws an RcsPermissionDeniedException.

```
void setActivationMode(Boolean active)
```

- Method: returns true if the RCS service is started.

```
boolean isServiceStarted()
```

Class **Intents.Service:**

This class defines intents to handle the RCS stack activation.

- Intent: this intent is used to check if RCS stack is activated.

```
static final String ACTION_GET_ACTIVATION_MODE =
"com.gsma.services.rcs.action.GET_ACTIVATION_MODE"
```

The RCS stack sends back the intent with a boolean extra parameter indicating if the RCS stack is activated or not:

```
static final String EXTRA_GET_ACTIVATION_MODE = "get_activation_mode"
```

- Intent: this intent is used to check if RCS stack activation is changeable by the client.

```
static final String ACTION_GET_ACTIVATION_MODE_CHANGEABLE =
"com.gsma.services.rcs.action.GET_ACTIVATION_MODE_CHANGEABLE"
```

The RCS stack sends back the intent with a boolean extra parameter indicating if the RCS stack activation is changeable by the client:

```
static final String EXTRA_GET_ACTIVATION_MODE_CHANGEABLE =
"get_activation_mode_changeable"
```

- Intent: this intent is used to set the RCS stack activation mode.

```
static final String ACTION_SET_ACTIVATION_MODE =
"com.gsma.services.rcs.action.SET_ACTIVATION_MODE"
```

A boolean extra parameter is set by the client to indicate if the RCS stack shall be activated or deactivated:

```
static final String EXTRA_SET_ACTIVATION_MODE = "set_activation_mode"
```

- Intent: this intent is used to check if RCS stack is compatible with RCS client API

```
static final String ACTION_GET_COMPATIBILITY =
"com.gsma.services.rcs.action.GET_COMPATIBILITY"
```

The RCS client API sends a String extra field in ACTION_GET_COMPATIBILITY intent to convey the service class name:

```
static final String EXTRA_GET_COMPATIBILITY_SERVICE =
"get_compatibility_service"
```

The RCS client API sends a String extra field in ACTION_GET_COMPATIBILITY intent to convey the code name:

```
static final String EXTRA_GET_COMPATIBILITY_CODENAME =
"get_compatibility_codename"
```

The RCS client API sends an integer extra field in ACTION_GET_COMPATIBILITY intent to convey the version:

```
static final String EXTRA_GET_COMPATIBILITY_VERSION =
"get_compatibility_version"
```

The RCS client API sends an integer extra field in ACTION_GET_COMPATIBILITY intent to convey the increment:

```
static final String EXTRA_GET_COMPATIBILITY_INCREMENT =
"get_compatibility_increment"
```

The RCS stack sends back the intent with a boolean extra parameter indicating if the RCS client API is compatible:

```
static final String EXTRA_GET_COMPATIBILITY_RESPONSE =
"get_compatibility_response"
```

- Intent: this intent is used to check if RCS service is started

```
static final String ACTION_GET_SERVICE_STARTING_STATE =
"com.gsma.services.rcs.action.GET_SERVICE_STARTING_STATE"
```

The RCS stack sends back the intent with a boolean extra parameter indicating if the RCS service is started:

```
static final String EXTRA_GET_SERVICE_STARTING_STATE =
"get_service_starting_state"
```

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_FILE_TRANSFER =
  "com.gsma.services.rcs.action.INITIATE_ONE_TO_ONE_FILE_TRANSFER"
  ```

- 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 dialer 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.

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, geoloc, 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 explicitly bind to the service called com.gsma.rcs.service.RcsCoreService.

```
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: returns the configuration that is common for all the service APIs.

```
CommonServiceConfiguration getCommonConfiguration()
```

- 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()
```

- Method: returns the reason code for the service registration

```
RcsServiceRegistration.ReasonCode getServiceRegistrationReasonCode()
```

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 **RcsServiceRegistration**:

- Enum: the reason code for RCS service unregistration.

```
enum ReasonCode { UNSPECIFIED(0), CONNECTION_LOST(1), BATTERY_LOW(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(RcsServiceRegistration.ReasonCode
reasonCode)
```

Class **CommonServiceConfiguration:**

This class represents the particular common configuration of all RCS Services.

- Enum: the minimum battery level.

```
enum MinimumBatteryLevel { NONE(0), PERCENT_5(5), PERCENT_10(10),
PERCENT_20(20) }
```

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

```
String getMyDisplayName()
```

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

```
void setMyDisplayName(String name)
```

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

```
ContactId getMyContactId()
```

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

```
boolean isConfigValid()
```

- Method: returns the messaging client mode.

```
MessagingMode getMessagingUX()
```

- Method: returns the default messaging method.

```
MessagingMethod getDefaultMessagingMethod()
```

- Method: set the default messaging method.

```
void setDefaultMessagingMethod(MessagingMethod method)
```

- Method: returns the minimum battery level.

```
MinimumBatteryLevel getMinimumBatteryLevel()
```

- Method: sets the minimum battery level. Under the specified level, the RCS stack unregisters from the RCS platform.

```
setMinimumBatteryLevel(MinimumBatteryLevel level)
```

4.4.3.3 Common Data Classes

Class **Geoloc:**

This class allows extracting geoloc information and is used in common for both geoloc chat message content and geoloc sharings.

- Constructor: creates a Geoloc instance with the specified parameters.

```
Geoloc(String label, double latitude, double longitude, long
expiration, float accuracy)
```

- Constructor: returns a Geoloc instance as parsed from the CONTENT field in the GeolocSharingLog provider or the CONTENT field of a geoloc chat message in the ChatLog.Message provider.

```
Geoloc(String geolocContent)
```

- 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()
```

- Method: returns the String representation of a Geoloc object (same format as can be given in one of the constructors and the same format as is stored in the chat message provider).

```
String toString()
```

4.4.3.4 Exceptions

Class **RcsServiceException**:

This is the parent exception from which all of the below checked exceptions extend.

Class **RcsGenericException:**

This generic class must be thrown when 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 indented to extend this exception. The client must be able to trust that in case of any failure what so ever and none of the more specific exception below are thrown this exception will be thrown as a kind of default exception to signify that some error occurred that 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 fully could perform its scope of responsibility without having to be connected to the IMS like for instance calling getConfiguration() on a service

Class **RcsMaxAllowedSessionLimitReachedException**:

This class is thrown if the message/filetransfer/imageshare/geolocationshare etc (all the types) cannot be sent/transfered/resent or a new group chat 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 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 (UnsupportedOperationException)**:

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 pauseable file transfer that does not support that or trying to accept a file transfer on the originating side etc.

Class **RcsIllegalArgumentException (IllegalArgumentException)**:

This class is thrown when a method of the service API is called with one or multiple illegal input parameter. 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 and read access to the providers requires the com.gsma.services.permission.RCS permission. This is a new permission covering general access to the RCS service.

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 received, parsed and stored new provisioning information. This could be either a provisioning or re-provisioning success or even an unprovisioning.

```
com.gsma.services.rcs.action.SERVICE_PROVISIONING_DATA_CHANGED
```

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.

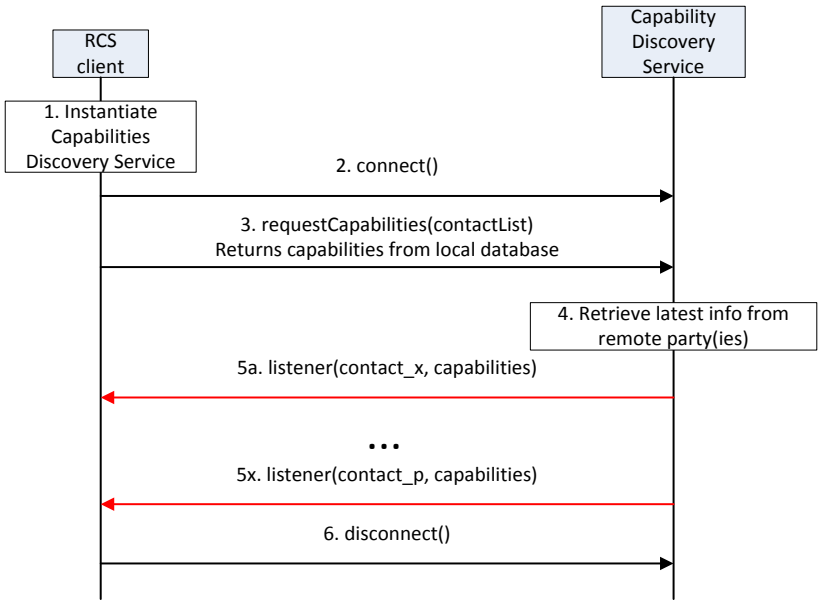

Figure 2: Get the capabilities of a list of remote contacts

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. If no matching contact capabilities is found then null is returned.

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

- Method: (deprecated) 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) and a refresh for updated capabilities was not already requested not too long ago for the same contact. 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 result of the capability refresh request is provided to all the clients that have registered the listener for this event. If the contacts parameter is null, the stack requests capabilities for all contacts existing in the local address book.

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

- Method: (deprecated) 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.

- Constants: a mask of bit to indicate if capability is supported.

```
static final int CAPABILITY_FILE_TRANSFER_SUPPORT = 0x00000001
static final int CAPABILITY_IM_SUPPORT = 0x00000002
static final int CAPABILITY_GEOLOC_PUSH_SUPPORT = 0x00000004
static final int CAPABILITY_IMAGE_SHARING_SUPPORT = 0x00000008
static final int CAPABILITY_VIDEO_SHARING_SUPPORT = 0x00000010
```

- Method: returns true if capabilities are supported, else returns false

  boolean hasCapabilities(int capabilties)

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

```
boolean isFileTransferSupported()
```

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

```
boolean isImSessionSupported()
```

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

```
boolean isImageSharingSupported()
```

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

```
boolean isVideoSharingSupported()
```

- Method: (deprecated) 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()
```

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 BASECOLUMN_ID = “_id”
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
BASECOLUMN_ID	Long (not null)	Unique value
CONTACT	String (primary key not null)	ContactId formatted number of contact associated to the capabilities
CAPABILITY_IMAGE_SHARING	Integer (not null)	Image sharing capability. Values: 1 (true), 0 (false)
CAPABILITY_VIDEO_SHARING	Integer (not null)	Video sharing capability. Values: 1 (true), 0 (false)
CAPABILITY_IM_SESSION	Integer (not null)	IM/Chat capability. Values: 1 (true), 0 (false)
CAPABILITY_FILE_TRANSFER	Integer (not null)	File transfer capability. Values: 1 (true), 0 (false)
CAPABILITY_GEOLOC_PUSH	Integer (not null)	Geolocation push capability. Values: 1 (true), 0 (false)

Data	Data type	Comment
CAPABILITY_EXTENSIONS	String	Supported RCS extensions. List of features tags semicolon separated (e.g. <TAG1>;<TAG2>;…;TAGn)
AUTOMATA	Integer (not null)	Is an automata. Values: 1 (true), 0 (false).
TIMESTAMP	Long (not null)	Timestamp of when these capabilities was received.

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 or an ICSI 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.

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 analyzing the certificate of the application supporting the feature tag or by checking the provisioning.

There are three types of IARI extensions:

1. Extensions for service provider specific service. (IARI format only).This 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.A5TgS99bJloIUIA3P45wjp63tk" />
</application>
```

2. Extensions for third-party specific service. (IARI format only). This extension should start with the prefix « `+g.3gpp.iari-ref="urn%3Aurn-7%3A3gpp-application.ims.iari.rcs.mnc<mnc>.mcc<mcc>.xxx` », 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>
```

3. Extensions for new services defined within GSMA (ICSI format only). The extension should start with the prefix « *+g.3gpp.icsi-ref=”* `urn%3Aurn-7%3A3gpp-service.ims.icsi.gsma.xxx` », where « xxx » a unique service identifier (string) associated to the new EC service. 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="gsma."/" />
</application>
```

NOTE: This is the format to be used to implement the new Enriched Calling Services (Call Composer, Post Call, Shared Sketch and Shared Maps). Please refer to Annex A of this document for further details.

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 and read access to the capabilities provider requires the following permissions:

- com.gsma.services.permission.RCS:
  this is a general permission that governs access to RCS services.
- 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).
- 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 signaling 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: Gets a chat message from its unique ID.

```
ChatMessage getChatMessage(String msgId)
```

- Method: returns true if it's possible and allowed to initiate a new group chat right now, else returns false.

```
boolean isAllowedToInitiateGroupChat()
```

- Method: returns true if it's possible and allowed to initiate a new group chat with the specified contact right now, else returns false.

```
boolean isAllowedToInitiateGroupChat(ContactId contact)
```

- 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 (i.e. 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: disables and clears any delivery expiration for a set of chat messages regardless if the delivery of them has expired already or not.

```
void clearMessageDeliveryExpiration(Set<String> msgId)
```

Class **ChatMessage**:

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

- Method: returns the contactId of the remote contact for this message or null if this is an outgoing group chat message.

```
ContactId getRemoteContact()
```

- 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()Method: returns the local timestamp of when the chat message was delivered for outgoing messages or 0 for incoming messages or it was not yet delivered.

```
long getTimestampDelivered()
```

- Method: returns the local timestamp of when the chat message was displayed for outgoing messages or 0 for incoming messages or it was not yes displayed.

```
long getTimestampDisplayed()
```

- Method: returns true if delivery for this chat message has expired or is otherwise false.

NOTE: False means either that delivery for this chat message has not yet expired, delivery has been successful, delivery expiration has been cleared (see clearMessageDeliveryExpiration) or that this particular chat message is not eligible for delivery expiration in the first place.

```
boolean isExpiredDelivery()
```

- Method: returns the status of the chat message.
- Status getStatus()Method: returns the reason code of the chat message.

```
ReasonCode getReasonCode()
```

- Method: returns the chat ID of this chat message.

```
String getChatId()
```

- Method: returns true if this chat message has been marked as read.

```
boolean isRead()
```

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: returns true if it's possible and allowed to to send messages in the one to one chat right now, else returns false.

```
boolean isAllowedToSendMessage()
```

- Method: sends a text 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 content will be stored in the message provider encoded as such.

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

- Method: sends a geoloc 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 geoloc message content is considered as mime type "application/geoloc" and will be stored in the message provider encoded as such.

NOTE: That the geoloc content can be extracted to a Geoloc object using the Geoloc class constructor.

```
ChatMessage sendMessage(Geoloc geoloc)
```

- Method: This method should be called to notify the stack if there is ongoing composing or not in this OneToOneChat. If there is an ongoing chat session established with the remote side corresponding to this OneToOneChat this means that a call to this method will send the "is-composing" event or the "is-not-composing" event to the remote side. However, since this method can be called at any time even when there is no chat session established with the remote side or when the stack is not even connected to the IMS server then the stack implementation needs to hold the last given information (i.e. composing or not composing) in memory and then send it later when there is an established session available to relay this information on.

NOTE: If this OneToOneChat corresponds to 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 setComposingStatus(boolean ongoing)
```

- 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 mimeType,
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), INITIATING(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_INACTIVITY (3),
REJECTED_BY_SECONDARY_DEVICE(4), REJECTED_SPAM(5),
REJECTED_MAX_CHATS(6), REJECTED_BY_REMOTE(7), REJECTED_BY_TIMEOUT(8),
REJECTED_BY_SYSTEM(9), FAILED_INITIATION(10) }
```

- Enum: the status of the participant.

```
enum ParticipantStatus { INVITE_QUEUED(0), INVITING(1), INVITED(2),
CONNECTED(3), DISCONNECTED(4), DEPARTED(5), FAILED(6), DECLINED(7),
TIMEOUT(8) }
```

- Method: returns the local timestamp of when the group chat invitation was initiated for outgoing group chats or the local timestamp of when the group chat invitation was received for incoming group chat invitations.

```
long getTimestamp()
```

- 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.

```
Map<ContactId, ParticipantStatus> getParticipants()
```

- Method: returns the direction of the group chat.

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

- Method: returns the contact Id of the inviter of the group chat or null if this is a group chat initiated by the local user (ie outgoing group chat).

```
ContactId getRemoteContact()
```

- 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 and allowed 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 isAllowedToSendMessage()
```

- Method: sends a text chat 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 chat 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 message content is

considered as mime type “application/geoloc” and will be stored in the message provider as such. Note that the geoloc content can be extracted to a Geoloc object using the Geoloc class constructor.

```
ChatMessage sendMessage(Geoloc geoloc)
```

- Method: This method should be called to notify the stack if there is ongoing composing or not in this GroupChat. If there is an ongoing chat session established with the remote side corresponding to this GroupChat this means that a call to this method will send the “is-composing” event or the “is-not-composing” event to the remote side. However, since this method can be called at any time even when there is no chat session established with the remote side or when the stack is not even connected to the IMS server then the stack implementation needs to hold the last given information (i.e. composing or not composing) in memory and then send it later when there is an established session available to relay this information on.

NOTE: If this GroupChat corresponds to 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 setComposingStatus(boolean ongoing)
```

- Method: returns true if it’s possible and allowed to invite additional participants to the group chat right now, else returns false.

```
boolean isAllowedToInviteParticipants()
```

- Method: returns true if it's possible and allowed to invite the specified participant to the group chat right now, else returns false.

```
boolean isAllowedToInviteParticipant(ContactId participant)
```

- Method: invite additional participants to this group chat.

```
void inviteParticipants(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: returns true if it’s possible and allowed to leave this group chat. Typically, this is always possible unless user has already chosen to leave this group chat already.

```
boolean isAllowedToLeave()
```

- 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 mimeType, 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 mimeType, String msgId, GroupDeliveryInfo.Status
status, GroupDeliveryInfo.ReasonCode reasonCode)
```

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

```
void onParticipantStatusChanged(String chatId, ContactId contact,
ParticipantStatus status)
```

- 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 **ChatServiceConfiguration**:

This class represents the particular configuration of a IM Service.

- 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 of that. This method is internally making use of a combination of the provisioning values `imWarnSf` and `imCapAlwaysOn`.

```
boolean isChatWarnSf()
```

- 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: 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 the is-composing timeout value in milliseconds.

```
long getIsComposingTimeout()
```

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

```
int getGeolocLabelMaxLength()
```

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

```
long getGeolocExpirationTime()
```

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

```
boolean isGroupChatSupported()
```

Class **ChatLog.GroupChat**:

- Method: utility method to get a map of the participants (defined by their contactId's) and their individual corresponding ParticipantStatus from its string representation in the ChatLog provider.

```
static Map<ContactId, ParticipantStatus> getParticipants(Context ctx,
String participantInfo)
```

```
Class ChatLog.Message.MimeType: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.
- "mimeType": (String) the mime type of the message (See ChatLog.Message.MimeType)

```
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 is a message deliverty timeout 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.
- “messageId”: (String) unique message ID of the message.

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

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.
- “mimeType”: (String) the mime type of the message (See ChatLog.Message.MimeType)

```
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:**

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 = 0
```

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 BASECOLUMN_ID = “_id”
static final String CHAT_ID = “chat_id”
static final String CONTACT = “contact”
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
BASECOLUMN_ID	Long (not null)	Unique value
CHAT_ID	String (primary key not null)	Id for chat room
CONTACT	String	ContactId formatted number of the inviter of the group chat or null if this is a group chat initiated the local user (ie outgoing group chat).
STATE	Integer (not null)	State of chat room. See enum GroupChat.State the list of states.
SUBJECT	String	Subject of the group chat room
DIRECTION	Integer (not null)	Status direction of group chat. See enum Direc for the list of directions.
TIMESTAMP	Long (not null)	timestamp of the invitation
REASON_CODE	Integer (not null)	Reason code associated with the group chat st See enum GroupChat.ReasonCode for the list reason codes
PARTICIPANTS	String (not null)	Representation of participants and their individu associated status stored as a String parseable the ChatLog.GroupChat.getParticipants() meth

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>

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

```
static final String BASECOLUMN_ID = "_id"
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 EXPIRED_DELIVERY = “expired_delivery”
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

Data	Data Type	Description
BASECOLUMN_ID	Long (not null)	Unique value (even across several history log members)
MESSAGE_ID	String (primary key not null)	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 (not null)	Content of the message (as defined by one of the mimetypes in ChatLog.Message.Mimetype)
TIMESTAMP	Long (not null)	Time when message inserted
TIMESTAMP_SENT	Long (not null)	Time when message sent. If 0 means not sent.
TIMESTAMP_DELIVERED	Long (not null)	Time when message delivered. If 0 means not delivered.
TIMESTAMP_ DISPLAYED	Long (not null)	Time when message displayed. If 0 means not displayed.
EXPIRED_DELIVERY	Integer (not null)	If delivery has expired for this message. Values: 1 (true), 0 (false)
MIME_TYPE	String (not null)	Multipurpose Internet Mail Extensions (MIME) type of message
STATUS	Integer (not null)	See enum Message.Content.Status or enum Message.GroupChatEvent.Status for the list of statuses.
REASON_CODE	Integer (not null)	Reason code associated with the message status. See enum Message.Content.ReasonCode for the list of reason codes
READ_STATUS	Integer (not null)	This is set on the receiver side when the message has been marked as read. See enum ReadStatus for the list of statuses.

DIRECTION	Integer (not null)	Status direction of message. See enum Direction for the list of directions.

4.4.5.5 Permissions

Access to the Chat API and read access to the chat provider requires the following permissions:

- com.gsma.services.permission.RCS: this is a general permission that governs access to RCS services.

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 file transfer 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 a file transfer from its unique ID. If no ongoing FileTransfer matching the transferId if 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: returns true if it's allowed to initiate file transfer to the contact specified by the contact parameter, else returns false.

```
boolean isAllowedToTransferFile(ContactId contact)
```

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

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

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

```
FileTransfer transferFile(ContactId contact, Uri file, Disposition
disposition, boolean attachFileIcon)
```

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

```
boolean isAllowedToTransferFileToGroupChat(String chatId)
```

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

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

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

```
FileTransfer transferFileToGroupChat(String chatId, Uri file,
Disposition disposition, 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: Disables and clears any delivery expiration for a set of file transfers regardless if the delivery of them has expired already or not.

```
void clearFileTransferDeliveryExpiration(<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),
INITIATING(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_BY_TIMEOUT(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), REJECTED_MEDIA_FAILED(12),
REJECTED_BY_SYSTEM(13), PAUSED_BY_SYSTEM(14), PAUSED_BY_USER(15),
FAILED_INITIATION(16), FAILED_DATA_TRANSFER(17), FAILED_SAVING(18),
FAILED_DELIVERY(19), FAILED_DISPLAY(20), FAILED_
NOT_ALLOWED_TO_SEND(21) }
```

- Enum: file disposition to indicate if the file is an attachment (default mode) or to be automatically played (e.g. audio message).

```
enum Disposition { ATTACH(0), RENDER(1) }
```

NOTE: The file transfer service API was not generic in the solution which defines a dedicated MIME type “application/audiomessage” for audio messages in the history log. In that case it only applies to audio message transfer that requires immediate rendering and also it is needed to define another MIME type in case the same service is implemented for a video message. To avoid these issues, a new parameter ‘Disposition’ added to the history log and session. The value of the ‘Disposition’ attribute may be RENDER (for audio messages or others in the future) or ATTACH (for normal file transfer). The usage for an Audio message is as follows:

  - Disposition = ATTACH
  - MIME type = “audio/amr” (or any other audio format if the UI authorizes it)

- 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 or null if this is an outgoing group file transfer.

```
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 file disposition.

```
Disposition getFileDisposition()
```

- Method: returns the timestamp for when file on the content server is no longer valid to download.

```
long getFileExpiration()
```

- Method: returns the timestamp for when file icon on the content server is no longer valid to download.

```
long getFileIconExpiration()
```

- Method: returns true if delivery for this file has expired or false otherwise.

NOTE: False means either that delivery for this file has not yet expired, delivery has been successful, delivery expiration has been cleared (see clearFileTransferDeliveryExpiration) or that this particular file is not eligible for delivery expiration in the first place.

```
boolean isExpiredDelivery()
```

- 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).

```
long getFileSize()
```

- 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 isAllowedToPauseTransfer()
```

- Method: pauses the file transfer.

```
void pauseTransfer()
```

- Method: returns true if it's allowed to resume 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 canResumeTransfer()
```

- Method: resumes the file transfer.

```
void resumeTransfer()
```

- Method: returns whether you are allowed to resend the transfer.

```
boolean isAllowedToResendTransfer()
```

- Method: resend a file transfer which was previously failed. This 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()
```

- Method: returns the local timestamp of when the file transfer was delivered for outgoing file transfers or 0 for incoming file transfers or it was not yes displayed.

```
long getTimestampDelivered()
```

- Method: returns the local timestamp of when the file transfer was displayed for outgoing file transfers or 0 for incoming file transfers or it was not yes displayed.

```
long getTimestampDisplayed()
```

- Method: returns true is this file transfer has been marked as read.

```
boolean isRead()
```

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 options.

```
enum ImageResizeOption { ALWAYS_RESIZE(0), ALWAYS_ASK(1), NEVER_RESIZE(2) }
```

- Method: returns the file size warning of a File Transfer configuration. It can return zero 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 max duration in milliseconds of an audio message. It returns the default value of 600 000 milliseconds if this value has not been set by the auto-configuration server.

```
long getMaxAudioMessageDuration()
```

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

```
boolean isAutoAcceptEnabled()
```

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

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

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

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

- Method: returns the image resize option

```
ImageResizeOption getImageResizeOption()
```

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

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

- 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.filetransfer.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.filetransfer.action.RESUME_FILE_TRANSFER

Intent broadcasted when there is a file transfer delivery timeout 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.
- “transferId”: (String) unique ID of the file transfer.

```
com.gsma.services.rcs.filetransfer.action.FILE_TRANSFER_DELIVERY_EXPI
RED
```

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 BASECOLUMN_ID = “_id”
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 FILE_EXPIRATION = "file_expiration"
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 DISPOSITION = "disposition"
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 EXPIRED_DELIVERY = “expired_delivery”
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
BASECOLUMN_ID	Long (not null)	Unique value (even across several history log members)
FT_ID	String (primary key not null)	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
FILE_EXPIRATION	Integer (not null)	Time when the file on the content server is no longer valid to download.
FILEICON_EXPIRATION	Integer (not null)	Time when the file icon on the content server is no longer valid to download.
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
DISPOSITION	Integer (not null)	File disposition: attachment or render. See enum Disposition.
DIRECTION	Integer (not null)	Incoming transfer or outgoing transfer. See enum Direction.
FILESIZE	Long (not null)	File size in bytes
TRANSFERRED	Long (not null)	Size transferred in bytes
TIMESTAMP	Long (not null)	Date of the transfer
TIMESTAMP_SENT	Long (not null)	Time when file is sent. If 0 means not sent.
TIMESTAMP_DELIVERED	Long (not null)	Time when file is delivered. If 0 means not delivered.
TIMESTAMP_ DISPLAYED	Long (not null)	Time when file is displayed.
EXPIRED_DELIVERY	Integer (not null)	If delivery has expired for this file. Values: 1 (true), 0 (false)
STATE	Integer (not null)	See note below for the list of states
REASON_CODE	Integer (not null)	Reason code associated with the file transfer state. See enum FileTransfer.ReasonCode for possible reason codes.
READ_STATUS	Integer (not null)	This is set on the receiver side when the message has been marked as read. See enum ReadStatus for the list of statuses.

4.4.6.5 Permissions

Access to the File Transfer API and read acccess to the file transfer provider requires the following permissions:

- com.gsma.services.permission.RCS: this is a general permission that governs access to RCS services.

4.4.6.6 Package

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

4.4.6.7 Methods and Callbacks

Class **GroupDeliveryInfo:**

This class contains group delivery information for group chat messages and group file transfers.

- 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)}
```

4.4.6.8 Content Providers

A content provider is used to store the group delivery information persistently. There is one entry per recipient of a group chat message or a group file transfer.

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 with adding a path segment to the CONTENT_URI.

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

```
static final String BASECOLUMN_ID = "_id"
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
BASECOLUMN_ID	Long (not null)	Unique value
ID	String (part of primary key* not null)	Unique Id of the chat message (“msg_id”) or file transfer (“ft_id”)
CONTACT	String (part of primary key* not null)	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 (not null)	Time when message delivered. If 0 means not delivered.
TIMESTAMP_DISPLAYED	Long (not null)	Time when message displayed. If 0 means not displayed.
STATUS	Integer (not null)	See enum GroupDeliveryInfo.Status for the list of statuses.
REASON_CODE	Integer (not null)	See enum GroupDeliveryInfo.ReasonCode for the list of reason codes.

4.4.6.9 Permissions

Read access to the group delivery info provider requires the following permissions:

- com.gsma.services.permission.RCS:

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: 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), INITIATING(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_SPAM(5),
REJECTED_BY_TIMEOUT(6) REJECTED_LOW_SPACE(7), REJECTED_MAX_SIZE(8),
REJECTED_MAX_SHARING_SESSIONS(9), REJECTED_BY_USER(10),
REJECTED_BY_REMOTE(11), REJECTED_BY_SYSTEM(12),
FAILED_INITIATION(13), FAILED_SHARING(14), FAILED_SAVING(15) }
```

- 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 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 BASECOLUMN_ID = “_id”
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
BASECOLUMN_ID	Long (not null)	Unique value (even across several history log members)
SHARING_ID	String (primary key not null)	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

Data	Data type	Comment
DIRECTION	Integer (not null)	Incoming sharing or outgoing sharing. See enum Direction
FILESIZE	Long (not null)	File size in bytes
TRANSFERRED	Long (not null)	Size transferred in bytes
TIMESTAMP	Long (not null)	Date of the sharing
STATE	Integer (not null)	See enum ImageSharing.State for the list of states
REASON_CODE	Integer (not null)	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 and read access to the image share provider requires the following permissions:

- com.gsma.services.permission.RCS: this is a general permission that governs access to RCS services.

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 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), INITIATING(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_SPAM(5),
REJECTED_MAX_SHARING_SESSIONS(6), REJECTED_BY_USER(7)
REJECTED_BY_REMOTE(8), REJECTED_BY_TIMEOUT(9), REJECTED_BY_SYSTEM(10),
FAILED_INITIATION(11), FAILED_SHARING(12) }
```

- 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 video share 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: 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 in milliseconds.

```
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 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(int width, int height)
```

- 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. The orientation ID a value between 1 and 15 arbitrarily chosen by the sender, as defined in RFC5285.

```
void SetRemoteInfo(VideoCodec codec, String remoteHost, int
remotePort, int orientationId)
```

- 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, int width, int height, 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 video frame width (e.g. 176).

```
int getWidth()
```

- Method: returns the video frame height (e.g. 144).

```
int getHeight()
```

- 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 zero if this value was not set by the auto-configuration server.

```
long 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 BASECOLUMN_ID = “_id”
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"
```

The content provider has the following columns:

VIDEOSHARE

Data	Data type	Comment
BASECOLUMN_ID	Long (not null)	Unique value (even across several history log members)
SHARING_ID	String (primary key not null)	Unique sharing identifier
CONTACT	String (not null)	ContactId formatted number of the remote contact
DIRECTION	Integer (not null)	Incoming sharing or outgoing sharing. See enum Direction
TIMESTAMP	Long (not null)	Date of the sharing
STATE	Integer (not null)	See enum VideoSharing.State for the list of states
REASON_CODE	Integer (not null)	Reason code associated with the video sharing state. See enum VideoSharing.ReasonCode for the list of reason codes
DURATION	Long (not null)	Duration of the sharing in milliseconds. The value is only set at the end of the sharing.
VIDEO_ENCODING	String	Encoding of the shared video
WIDTH	Integer (not null)	Width of the shared video
HEIGHT	Integer (not null)	Height of the shared video

4.4.8.5 Permissions

Access to the Video Share API and read access to the video share provider requires the following permissions:

- com.gsma.services.permission.RCS:
  this is a general permission that governs access to RCS services.

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 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), INITIATING(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_SPAM(5),
REJECTED_MAX_SHARING_SESSIONS(6), REJECTED_BY_USER(7),
REJECTED_BY_REMOTE(8), REJECTED_BY_TIMEOUT(9),
REJECTED_BY_SYSTEM(10), FAILED_INITIATION(11), FAILED_SHARING(12) }
```

- 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 BASECOLUMN_ID = “_id”
static final String SHARING_ID = "sharing_id"
static final String CONTACT = "contact"
static final String CONTENT = "content"
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
BASECOLUMN_ID	Long (not null)	Unique value (even across several history log members)
SHARING_ID	String (primary key not null)	Unique sharing identifier
CONTACT	String (not null)	ContactId formatted number of the remote contact
CONTENT	String	The geolocation stored as a String parseable with the Geoloc(String) constructor .
MIME_TYPE	String (not null)	Multipurpose Internet Mail Extensions (MIME) type of the geoloc (typically “application/geoloc” as to be compatible with Chat.Message.Mimetype.GEOLOC)
DIRECTION	Integer (not null)	Direction of sharing. See enum Direction.
TIMESTAMP	Long (not null)	Date of the sharing
STATE	Integer (not null)	See enum GeolocSharing.State for valid states
REASON_CODE	Integer (not null)	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.permission.RCS:
  this is a general permission that governs access to RCS services and read access to the geoloc share provider .

4.4.10 Contact 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.contact**

4.4.10.2 Methods and Callbacks

Class **ContactService**:

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

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

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

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

- Method: block a contact.

```
void blockContact(ContactId contact)
```

- Method: unblock a contact.

```
void unblockContact(ContactId contact)
```

Class **ContactUtil**:

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

- Method: get a singleton instance of ContactUtil.

```
static ContactUtil getInstance(Context ctx)
```

- 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. An RcsPermissionDeniedException is thrown if the mobile country code failed to be read and is required to validate the contact.

```
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 IllegalArgumentException is thrown else a valid ContactId is returned. An RcsPermissionDeniedException is thrown if the mobile country code failed to be read and is required to format the contact.

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

- Method: returns the user Country Code. An RcsPermissionDeniedException is thrown if the mobile country code failed to be read.

```
String getMyCountryCode()
```

- Method: returns the user Country Area Code or null if it does not exist. An RcsPermissionDeniedException is thrown if the mobile country code failed to be read.

```
String getMyCountryAreaCode()
```

- Method: checks if the my country code is defined.

```
boolean isMyCountryCodeDefined()
```

- Method: get the vCard of a contact. The contact parameter contains the database URI of the contact in the native address book. The method returns a Uri to the visit card. The visit card 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)`).

```
Uri getVCard(Uri 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 isOnline()
  ```

- Method: is contact blocked.

  ```
  boolean isBlocked()
  ```

- Method: returns the time stamp when the blocking was activated. -1 if contact is not blocked.

  ```
  long getBlockingTimestamp()
  ```

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 and are represented by constants in the ContactsProvider class:

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
vnd.android.cursor.item/com.gsma.services.rcs.blocking-state	Blocking state (blocked | unblocked)
vnd.android.cursor.item/com.gsma.services.rcs.blocking-timestamp	Time stamp when the blocking was activated

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.
- For capability mime types, the summary description associated to the supported MIME type is always put into the field Data.DATA2. The detailed description associated to the supported capability MIME type is always put into the field Data.DATA3. Those two labels are displayed at UI level (i.e. menu item of the local native address book).
- If a capability MIME type is not set for a contact, this means that the associated capability is not supported.

4.4.10.4 Permissions

Access to the Contacts API requires the following permissions:

- com.gsma.services.permission.RCS:
  this is a general permission that governs access to RCS services.
- 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.

The below table shows data fields meaning depending on the mime type.

Data.MIMETYPE (*)	Data.DATA1 (**)	Data.DATA2	Data.DATA3
number	Phone number (String)	-	-
registration-state	Phone number (String)	0 / 1 (int)	-
image-share	Phone number (String)	Description summary (String)	Description details (String)
video-share	Phone number (String)	Description summary (String)	Description details (String)
im-session	Phone number (String)	Description summary (String)	Description details (String)
file-transfer	Phone number (String)	Description summary (String)	Description details (String)
extensions	Phone number (String)	List of supported extensions (String) separated by ';'	Description details (String)
geoloc-push	Phone number (String)	Description summary (String)	Description details (String)
blocking-state	Phone number (String)	0 / 1 (int)	-
blocking-timestamp	Phone number (String)	Timestamp (long)	-

- Note (*):for sake of clarity, only the ending part of the mime type is mentioned in this column. Each ending part is in the mime type column prefixed with “vnd.android.cursor.item/com.gsma.services.rcs.” (example for blocking-timestamp, it gives: “vnd.android.cursor.item/com.gsma.services.rcs.blocking-timestamp”).
- Note (**):phone numbers are formatted as defined by ContactId (i.e.“+<country code><national destination code><subscriber number>”).

NOTE: Shaded cells correspond to capability mime-types for which record exists only if capability is supported.

The core server autonomously updates the summary and detailed description of the raw contacts - for supported capabilities - if the local setting of the device is changed.

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, Crane Priority Release (CPR), Crane, 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 = "GSMA"
```

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

```
public final static int API_VERSION = 2
```

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

```
public final static int API_INCREMENTAL = 1
```

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 = 0
```

- Constant: Blackbird version

```
public final static int BLACKBIRD = 1
```

- Constant: Crane Priority Release version

```
public final static int CPR = 2
```

4.4.11.3 Content Providers

4.4.12 Multimedia Session API

In order to implement new IMS services, this service API exposes methods:

- to initiate a multimedia session between two clients
- to exchange instant multimedia messages between two clients

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:

1. 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.
2. 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: 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 are specified by the parameter accept-types and accept-wrapped-types.

```
MultimediaSession initiateMessagingSession(String serviceId,
ContactId contact, String[] acceptType, String[] acceptWrappedType)
```

- Method: send an instant multimedia message to a remote contact and for a given service. The content takes part of the message invitation, so any multimedia media session is needed to exchange the content here.

```
void sendInstantMultimediaMessage(String serviceId, ContactId
contact, byte[] content, String contentType)
```

- 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 (deprecated): 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: initiate a new multimedia session for real time streaming with a remote contact for a given service and with a given encoding.

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

- Method: adds a new multimedia messaging session listener.

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

- Method: removes a multimedia messaging session listener.

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

- Method: adds a new multimedia streaming session listener.

```
void addEventListener(MultimediaStreamingSessionListener listener)
```

- Method: removes a multimedia streaming session listener.

```
void removeEventListener(MultimediaStreamingSessionListener 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), INITIATING(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), ABORTED_BY_USER(1),
ABORTED_BY_REMOTE(2), ABORTED_BY_SYSTEM(3), ABORTED_BY_INACTIVITY(4),
REJECTED_BY_USER(5), REJECTED_BY_REMOTE(6), REJECTED_BY_TIMEOUT(7),
REJECTED_BY_SYSTEM(8), FAILED_INITIATION(9), FAILED_SESSION(10),
FAILED_MEDIA(11) }
```

- 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 (deprecated).

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

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

```
void sendMessage(byte[] content, String contentType)
```

- Method: flush all the data of the session.

```
void flushMessages()
```

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 (deprecated).

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

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

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

- Method: callback called when multimedia messages have been flushed.

```
void onMessagesFlushed(ContactId contact, String sessionId)
```

Class **MultimediaStreamingSession:**

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

- Method: returns the encoding used during the streaming.

```
String getEncoding()
```

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

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

NOTE: contentType parameter is not included in the sendPayload method, because there is no RTP header to send the content type (there is just a payload number which is set to 97 for undefined payloads). The way to send content type is to specify the encoding or content type in the SDP part during the initialisation of the session via initiateStreaming method. Another reason is that the current implementation uses a hardcoded and proprietary encoding parameter “X-RCS” in the current MM session SIP/SDP call flow.

Class **MultimediaStreamingSessionListener:**

This class offers callback methods on multimedia session events.

- Method: Callback called when the multimedia streaming 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 ()
```

- Method: return the inactivity timeout of a multimedia messaging session. Timeout in milliseconds.

```
long getMessagingSessionInactivityTimeout(String serviceId)
```

- Method: check if the service ID has been activated by the RCS configuration server. RcsServiceNotAvailableException is thrown if the serviceId is not configured.

```
boolean isServiceActivated(String serviceId)
```

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
```

Intent broadcasted when a new instant multimedia message is received. This Intent contains the following extras:

- “contact”: Contact ID of the remote contact
- “serviceId”: (String) Service ID
- “content”: (byte[]) Message content
- “contentType”: (String) Message content type

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

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 analysed 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> getFileUploads()
```

- 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 FileUpload service.

```
FileUploadServiceConfiguration getConfiguration()
```

Class **FileUploadServiceConfiguration**:

This class represents the particular configuration of the FileUpload 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 { INITIATING(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 timestamp when the file on the content server is no longer valid to download.

  ```
  long getFileExpiration()
  ```

- Method: returns the timestamp when the file icon on the content server is no longer valid to download.

  ```
  long getFileIconExpiration()
  ```

- 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)
```

- Method: callback called when the file has been uploaded.

```
void onUploaded(String uploadId, FileUploadInfo info)
```

4.4.13.3 Permissions

Access to the File Upload API requires the following permissions:

- com.gsma.services.permission.RCS:
  this is a general permission that governs access to RCS services.

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 **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 the CONTENT_URI 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.

```
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()
```

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 needs 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 history log provider can merge data from that database together with other history 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 providerUri,
  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 history log members.

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

- Method: Creates an id for the provider matching the specificed providerId that will be unique across all historylog member's tables.

  ```
  long createUniqueId(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 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"
```

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

```
static final String BASECOLUMN_ID = "_id"
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 EXPIRED_DELIVERY = "expired_delivery"
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"
static final String DISPOSITION = "disposition"
```

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

HISTORYLOG

Data	Data Type	Description
BASECOLUMN_ID	Long (not null)	Unique value (even across several history log members)
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

Data	Data Type	Description
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
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.
EXPIRED_DELIVERY	Integer	If delivery has expired for this entry. Values: 1 (true), 0 (false)
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 chat 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 milliseconds if this entry corresponds to a sharing or a call. The value is only set at the end of the sharing or call.
DISPOSITION	Integer	File disposition (ATTACH or RENDER) if the entry corresponds to a file transfer.

4.4.14.4 Permissions

Access to the History API is requires the following permissions:

- com.gsma.services.permission.RCS:
  this is a general permission that governs access to RCS services.

Annex A Usage of Multimedia Session API to Implement Enriched Calling Services (Informative)

NOTE: This section is informative and not normative.

This section intends to illustrate how the Multimedia Session API can be used to implement Call Composer, Post Call, Shared Sketch and Shared Maps services as defined in [PRD RCC.20].

Regarding call composer:

- The call composer session as defined in Section 2.4.3 of [PRD RCC.20] can be set up using method initiateMessagingSession as defined in Section 4.4.5.2 with the following parameters:
    - serviceId set to *gsma.callcomposer*
    - contact set to the remote contact
    - acceptType set to *application/vnd.gsma.encall+xml* and *message/cpim* MIME types (as defined in section 2.3.3 of [PRD RCC.20])
    - acceptWrappedType set to *message/imdn+xml* and *application/vnd.gsma.rcs-ft-http+xml* MIME types (as defined in section 2.3.3 of [PRD RCC.20])
- A call composer picture can be uploaded as defined in section 2.4.2 of [PRD RCC.20] using the File Upload API as defined in Section 4.4.13 of this document.
- When a delivery notification on the pre-call picture has to be sent using SIP Message as defined in Section 2.4.6 of [PRD RCC.20], the method sendInstantMultimediaMessage as defined in section 4.4.5.2 can be used with the following parameter:
    - serviceId set to gsma.callcomposer
    - contact set to the remote contact
    - content set with the IMDN.
    - contentType set to message/cpim

Similar principles than for call composer can be used to set up post call, shared sketch and shared maps sessions using:

- serviceId set to gsma.callunanswered for Post call service
- serviceId set to gsma.sharedsketch for Shared Sketch service
- serviceId set to gsma.sharedmap for Shared map service
- acceptType and acceptWrappedType with the values defined in Section 2.3.3 of [PRD RCC.20] for Post call and 2.9.9.2 of [PRD RCC.20] for Shared Sketch and Shared Map.

In particular for Post Call service, the method flushMessages defined in Section 4.4.5.2 can be used to avoid possible race condition: i.e. ensure that the post call note (or post call audio message XML element) has been properly sent to the network before closing the media session.

Annex B 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	T-API 1.0: Blackbird Profile	PSMC	Kelvin Qin / GSMA
2.0	10 Oct 2014	T-API 1.5: Multimedia Session APIs are added with some other API improvement	PSMC	Kelvin Qin / GSMA
3.0	23 Jun 2015	T-API 1.5.1: Maintenance release	PSMC	Erdem Ersoz / GSMA
4.0	01 Mar 2016	T-API 1.6: Crane Priority Release	PSMC	Erdem Ersoz / GSMA

A.2 Other Information

Type	Description
Document Owner	RCS JTA
Editor / Company	Erdem Ersoz / 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.