joyn Device API Specification

Version 0.90

31 October 2013

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

Security Classification: Non-confidential

Copyright Notice

Disclaimer

Antitrust Notice

Table of Contents

1 Introduction

1.1 Overview

This document defines the architecture and a set of standardized Application Programming Interfaces (API) to develop 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 defined in [PRD RCC.60].

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.
joyn 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
CS	Circuit Switched
FT	File Transfer
ID	Identifier
IM	Instant Messaging
IMS	IP Multimedia Sub-system
IS	Image Share
MIME	Multipurpose Internet Mail Extensions

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

1.5 References

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

1.6 Conventions

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

2 API Architecture

2.1 Architecture Overview

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

The fundamental enabling component is the **IMS Stack** which contains the protocol suite (Session Initiation Protocol [SIP], Message Session Relay Protocol [MSRP], Real-Time Protocol [RTP]/Real-Time Control Protocol [RTCP], Hyper-Text Transfer Protocol [HTTP], etc.) and core services (IMS Session Management, Media 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 the **Client logic**, which provides the client API and the functionality to mediate access to that API by multiple clients. Client applications and services access the underlying functionality exclusively through this interface. The client logic exposes two levels of client API:

- the **Service API**, which is the principal API providing 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 **Privileged Client API**, which provides access to particularly sensitive parts of the stack.

The joyn **Core Applications** are the (typically embedded) applications that provide the end-user's access to RCS services. The Core Applications also expose a **UX API** (not shown in Figure 1) whereby other applications can programmatically invoke operations that are interactively fulfilled by the Core Applications. Core Applications make use of both the Service API and the Privileged Client API.

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

A special distinction is made for independent applications, labelled here as **Trusted Applications** that provide services directly on top of the IMS Stack (and which may themselves define their own Service APIs).

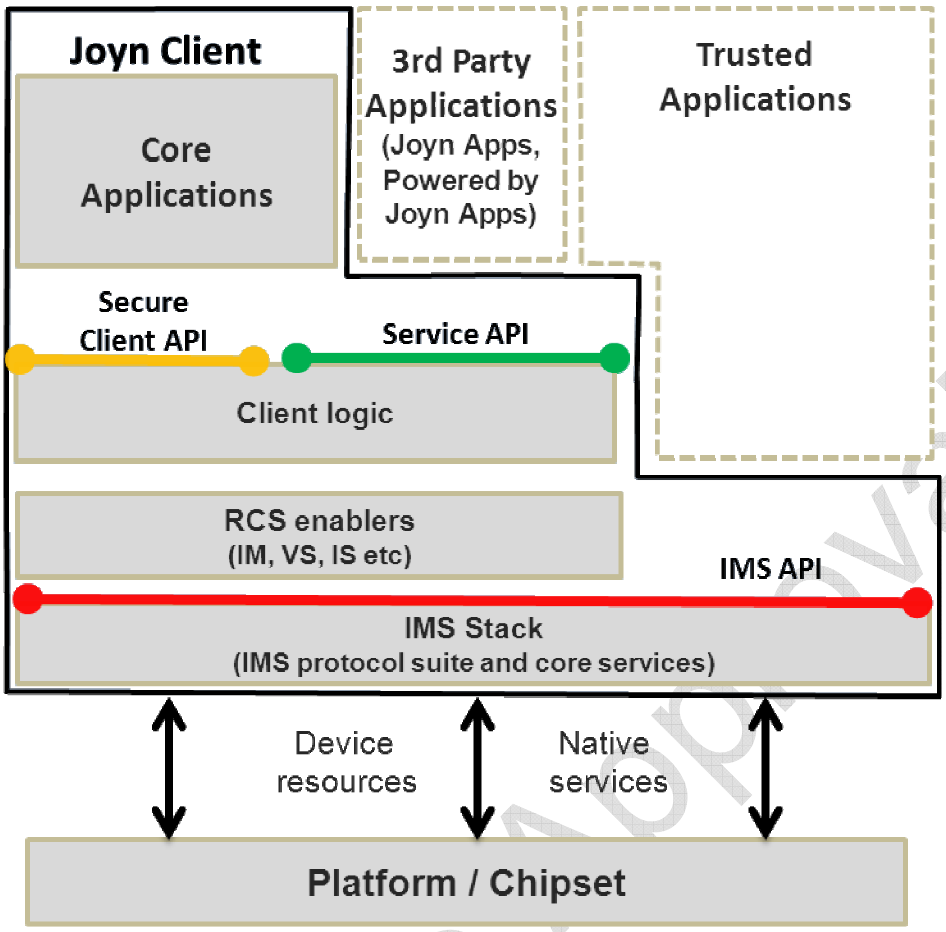


Figure 1: General Architecture Overview

2.1.1 API Descriptions

Architecturally, the four API types can be classed as exposing joyn functionality (UX API, Service API and Privileged Client API) and APIs exposing IMS stack functionality (IMS API).

2.1.1.1 UX APIs

UX APIs enable other installed applications to interface or link the native RCS service.

2.1.1.2 Service APIs

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.

Those functionalities are:

- IM/Chat service
- Image Share
- Video Share
- File Transfer
- Client Connector
- Capabilities

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

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.

2.1.1.3 Privileged Client APIs

The APIs on this level expose sensitive internal functionality of the joyn stack and, accordingly, access to these APIs cannot be granted by a user to any application.

The Privileged APIs will be defined in the future version.

2.1.1.4 IMS APIs

These APIs allow using a subset of IMS functionality to create new services that work along Standard Services.

The IMS APIs will be defined in the future version.

2.1.2 Applications Types

Applications types can be divided into three broad categories shown in Figure 2:

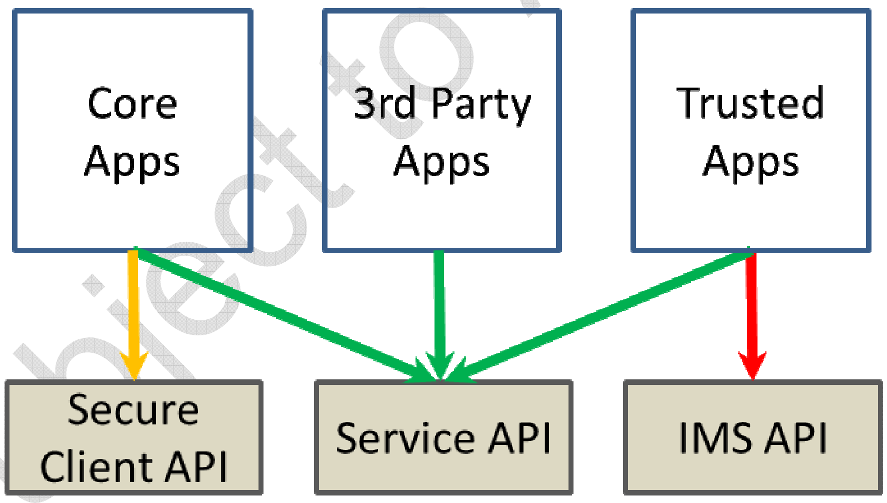


Figure 2: Mapping of target component types to APIs based on security

- Third Party Applications - All Third Party Apps can use only Service APIs. Some Third Party Application may provide service-over-service functionality. These applications use additional parameterization defined in the Service APIs, but have their own feature tags not defined by the joyn Specification.
- Core Applications - Components that are included in the joyn Client. These components must undergo GSMA accreditation as part of the joyn Client. Core Applications are the only ones that access Privileged Client APIs. Core Applications

may use Service APIs (such as IM) and can have overlapping functionality with Third Party Applications.

- Trusted Applications - These applications can use both IMS APIs and Service APIs. They can also define new services that are not part of the joyn Specification and need have their own feature tag.

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

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 joyn Client can be active on a device. This is defined by ID_3_1_1 in **Error! Reference source not found.**. This constraint limits the possibilities for deployment of additional joyn Clients with the Terminal API, as they cannot replace the package of a previously installed client 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 Client 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 joyn 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 joyn services.

4.2.1.1 Service API Access Control

The Service API comprises a number of specific services (IM, File Transfer, etc.) and access to each of these is mediated by one or more permissions. The Service APIs are designed so that they may be used by Third Party Applications. In each case, the API exposes privacy-sensitive information or may trigger service charges, and the user must grant permission for any application to use that API. The permissions are defined for each service API in the sections that follow.

4.2.1.2 Privileged Client API Access Control

The Privileged Client APIs are sensitive and their abuse could compromise the integrity of the stack or the joyn services, Access is therefore restricted so that they may only be used by authorised applications such as the Core Applications that are part of the joyn Client. Any component requiring access to the Privileged Client API must undergo review and approval in the same manner as the other elements of the joyn Client. Procedures for obtaining access to these APIs are beyond the scope of this document.

4.2.1.3 IMS API Access Control

Since the IMS API provides low-level access to the IMS stack, there is significant scope for abuse including generation of excessive network traffic, interception of requests intended for other applications, and other abuses resulting in denial of service.

Access to the IMS API is limited only to trusted components provided by MNO and/or OEMs. Procedures for obtaining access to these APIs are beyond the scope of this document.

4.3 UX API

This API offers:

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

4.3.1 Package

Package name **org.gsma.joyn**

4.3.2 Methods and Callbacks

Class **joynUtils**:

- Method: returns the list of joyn clients installed on the device (except myself). An application is identified as a joyn client by including an intent filter with the ACTION_CLIENT_SETTINGS action in the Manifest.xml of the application.

```
  <intent-filter>
    <category android:name="android.intent.category.LAUNCHER"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <action
android:name="org.gsma.joyn.client.action.VIEW_SETTINGS"/>
  </intent-filter>

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

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

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

The action should start with the specific client package name and should terminate with ACTION_CLIENT_GET_STATUS.

```
static void isJoynClientActivated(Context context, ResolveInfo
appInfo, BroadcastReceiver receiverResult)
```

- Method: Load the settings activity of a particular joyn client. The joyn client activity is identified by the object ResolveInfo recovered via the method getJoynClients.

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

4.3.3 Intents

Class **Intents.Chat**:

This class offers Intents to link applications to joyn 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_CHAT = "org.gsma.joyn.
action.VIEW_CHAT"
```

- Intent: load the chat application to start a new conversation with 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_INITIATE_CHAT =
"org.gsma.joyn.action.INITIATE_CHAT"
```

- 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 found the main entry of the group chat application is displayed.

```
static final String ACTION_VIEW_CHAT_GROUP =
"org.gsma.joyn.action.VIEW_GROUP_CHAT"
```

- 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. If no parameter the main entry of the group chat application is displayed.

```
static final String ACTION_INITIATE_CHAT_GROUP =
"org.gsma.joyn.action.INITIATE_CHAT_GROUP"
```

Class **Intents.Client**:

This class offers intents to link client applications.

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

```
static final String ACTION_CLIENT_SETTINGS =
“org.gsma.joyn.client.action.SETTINGS”;
```

- Intent: get the client status.

```
static final String ACTION_CLIENT_GET_STATUS =
“.client.action.GET_STATUS”;
```

Class **Intents.FileTransfer**:

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

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

```
static final String ACTION_VIEW_FT = "org.gsma.joyn.action.VIEW_FT"
```

- 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_FT =
"org.gsma.joyn.action.INITIATE_FT"
```

Class **Intents.IPCall:**

This class offers Intents to link applications to joyn applications for IP Call services.

- Intent: load the IP call application to view a call. This Intent takes into parameter an URI on the call (i.e. content://ipcalls/ipcall_ID). If no parameter found the main entry of the IP call application is displayed.

```
static final String ACTION_VIEW_IPCALL =
"org.gsma.joyn.action.VIEW_IPCALL"
```

- Intent: load the IP call application to start a new call with 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 IP call application is displayed.

```
static final String ACTION_INITIATE_IPCALL =
"org.gsma.joyn.action.INITIATE_IPCALL"
```

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

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

4.4 Services API

4.4.1 Overview

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

4.4.2 Access Control

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

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

4.4.3 Common architecture

The joyn terminal API contains the following service API:

- Capability service API,
- Chat API,
- File Transfer API,
- Video Share service API,
- Image Share service 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 **org.gsma.joyn**

4.4.3.2 Methods and Callbacks

Class **joynService**:

Each service API should extends the abstract class joynService.

- 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.

```
joynService(Context ctx, joynServiceListener listener)
```

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

  ```
  void connect()
  ```

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

  ```
  void disconnect()
  ```

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

  ```
  boolean isServiceConnected()
  ```

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

  ```
  boolean isServiceRegistered()
  ```

- Method: adds a listener on service registration event.

  ```
  void addServiceRegistrationListener(JoynServiceRegistrationListener
  listener)
  ```

- Method: removes a listener on service registration event.

  ```
  void
  removeServiceRegistrationListener(JoynServiceRegistrationListener
  listener)
  ```

Class **joynServiceListener**:

- 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). The reason code may have the following values: CONNECTION_LOST, SERVICE_DISABLED, INTERNAL_ERROR.

  ```
  void onServiceDisconnected(int reason)
  ```

Class **JoynServiceRegistrationListener**:

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

```
void onServiceUnregistered()
```

Class joynServiceConfiguration:

This class represents the particular configuration of joyn Service.

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

```
static boolean isServiceActivated()
```

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

```
static String getUserDisplayName()
```

4.4.3.3 Exceptions

Class **joynServiceException**:
This class is used to propagate generic service API exception.

Class **JoynServiceNotAvailableException**:
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. joyn service not yet started or API not yet connected).

Class **joynServiceNotRegisteredException**:
This class is thrown when a method of the service API using the RCS service platform is called and the terminal is not registered to the RCS service platform (e.g. not yet registered).

4.4.3.4 Permissions

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

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

4.4.4 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: the feature Store&Forward is managed internally by the RCS service and not appears at API level. So if a remote contact is offline and S&F is supported then the corresponding capabilities are activated.

Note: there is the same API between File transfer and File Tranfser over HTTP. So from API point of view 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 contain basic call flows of the CD service API.

Figure 3 is an example that shows the retrieval of the capabilities of a list of remote contacts.

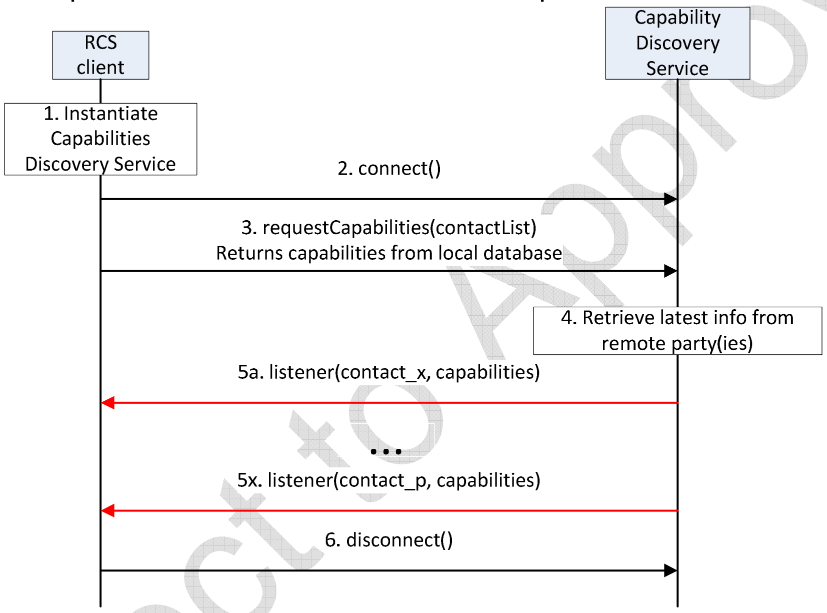

RCS client
Capability Discovery Service
1. Instantiate Capabilities Discovery Service
2. connect()
3. requestCapabilities(contactList)
Returns capabilities from local database
4. Retrieve latest info from remote party(ies)
5a. listener(contact_x, capabilities)
...
5x. listener(contact_p, capabilities)
6. disconnect()

Figure 3: 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 argument. 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 **org.gsma.joyn.capability**

4.4.4.3 Methods and Callbacks

Class **CapabilityService**:

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

- Method: connects to the API.

```
void connect()
```

- Method: disconnects from the API.

```
void disconnect()
```

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

```
Capabilities getMyCapabilities()
```

- Method: returns the capabilities of a given contact from the local database. This method doesn't request any network update to the remote contact. The parameter contact supports the following formats: MSISDN in national or international format, SIP address, SIP-URI or Tel-URI.

```
Capabilities getContactCapabilities(String contact);
```

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

```
void requestContactCapabilities(String contact)
```

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

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

- Method: requests capabilities for all contacts existing in the local address book. This method initiates in 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(ICapabilitiesListener listener)
```

- Method: unregisters a capabilities listener.

```
void removeCapabilitiesListener(ICapabilitiesListener listener)
```

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

```
void addCapabilitiesListener(Set<String> contacts,
ICapabilitiesListener listener)
```

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

```
void removeCapabilitiesListener(Set<String> contacts,
ICapabilitiesListener listener)
```

Class **ICapabilitiesListener**:
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(String contact, Capabilities
capabilities)
```

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

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

  ```
  boolean isFileTransferSupported()
  ```

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

  ```
  boolean isImSessionSupported()
  ```

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

  ```
  boolean isImageSharingSupported()
  ```

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

  ```
  boolean isVideoSharingSupported()
  ```

- Method: returns true if 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()
  ```

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 Mobile Subscriber Integrated Services Digital Network Number (MSISDN).

The content provider has the following columns:

Data	Data type	Comment
CONTACT_NUMBER	TEXT	Contains the MSISDN of the contact associated to the capabilities
CAPABILITY_IMAGE_SHARING	Integer	Image sharing capability. Values: 1 (true), 0 (false)
CAPABILITY_VIDEO_SHARING	Integer	Video sharing capability. Values: 1 (true), 0 (false)
CAPABILITY_IM_SESSION	Integer	IM/Chat capability. Values: 1 (true), 0 (false)
CAPABILITY_FILE_TRANSFER	Integer	File transfer capability. Values: 1 (true), 0 (false)
CAPABILITY_GEOLOC_PUSH	Integer	Geolocation push capability. Values: 1 (true), 0 (false)
CAPABILITY_IP_VOICE_CALL	Integer	IP voice call capability. Values: 1 (true), 0 (false)
CAPABILITY_IP_VIDEO_CALL	Integer	IP video call capability. Values: 1 (true), 0 (false)

Data	Data type	Comment
CAPABILITY_EXTENSIONS	TEXT	Supported RCS extensions. List of features tags semicolon separated (e.g. <TAG1>;<TAG2>;…;TAGn)

4.4.4.5 RCS extensions

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

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

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

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

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

An extension starts with the prefix « `+g.3gpp.iari-ref=”urn%3Aurn-7%3A3gpp-application.ims.iari.rcse.xxx.yyy”`, where “xxx“ is the MNO/OEM ID and “yyy” is an ID associated to the application implementing the RCS extension.

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

```
<intent-filter>
        <action android:name="org.gsma.joyn.capability.EXTENSION"/>
        <data android:mimeType="+g.3gpp.iari-ref/urn%3Aurn-7%3A3gpp-application.ims.iari.rcse.xxx.yyy"/>
</intent-filter>
```

Sample:

Tic Tac Toe game:

```
<application android:label="Tic Tac Toe">
        <activity android:name=".Main">
             <intent-filter>
                    <action android:name="org.gsma.joyn.capability.EXTENSION"/>
                    <data android:mimeType="+g.3gpp.iari-ref/urn%3Aurn-7%3A3gpp-
                    application.ims.iari.rcse.orange.tictactoe"/>
              </intent-filter>
        </activity>
```

4.4.4.6 Permissions

Access to the Capabilities API is requires the following permissions:

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

4.4.5 IM/Chat API

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

- Sending messages to a contact.
- Starting group chats with an ad-hoc list of participants and an optional subject.
- Joining existing group chats.
- Re-joining existing group chats.
- Restarting a previous group chat.
- Extends a 1-1 chat to a group chat.
- Sending messages in a group chat.
- Leaving a group chat.
- Adding participants to a group chat.
- Retrieving information about a group chat (status, participants and their status)
- Receiving notifications about incoming messages, “is-composing” events, group chat invitations and group chat events.
- Accept/reject an incoming chat invitation.
- Displaying chat history (messages and group chats).
- Erasing chat history by user, by group chat, or by single messages.
- Marking messages as displayed.
- Receiving message delivery reports.
- Read configuration elements affecting IM.

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

4.4.5.1 IM/Chat API calling flow

The figures in this section contain basic call flows of the IM/Chat service API.

4.4.5.1.1 Session establishment

Figure 4 is an example that shows the flow for a RCS client establishing a IM/Chat session with a remote contact.

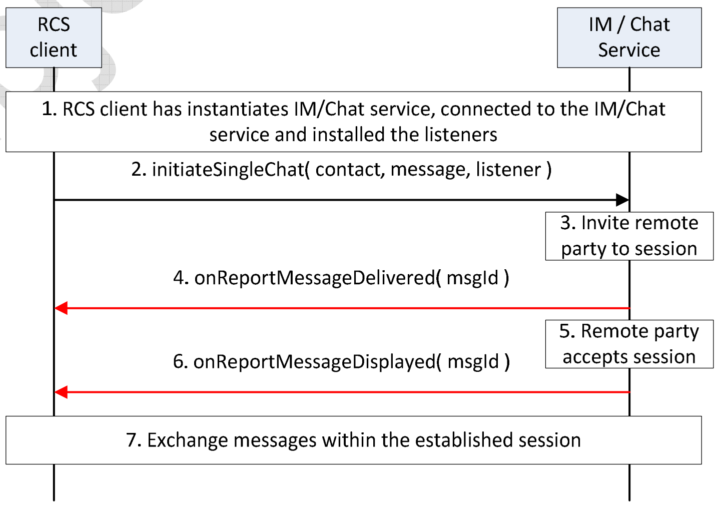

RCS client
IM / Chat Service
1. RCS client has instantiates IM/Chat service, connected to the IM/Chat service and installed the listeners
2. initiateSingleChat(contact, message, listener)
3. Invite remote party to session
4. onReportMessageDelivered(msgId)
5. Remote party accepts session
6. onReportMessageDisplayed(msgId)
7. Exchange messages within the established session

Figure 4: Establish a one-to-one chat session with remote party

1. The RCS client instantiates a service instance of the IM/Chat Service, establishes a connection with the IM/Chat Service and associates the listener with this RCS client.
2. By selecting a user, opening the IM window and typing a message, the user initiates a IM/Chat session request to the remote party
3. RCS service sends the session request to remote party
4. The session request is delivered to the remote party and a confirmation of delivery of the message is received by the RCS service. This delivery confirmation is provided to the RCS client.
5. By opening the chat window and viewing the incoming message, the remote party accepts the chat session and sends a confirmation of message display
6. RCS service provides the message displayed notification to the RCS client
7. Using procedures described in Figure 6, the two users exchange IM / Chat messages

4.4.5.1.2 Incoming session request

Figure 5 is an example that shows the flow for a RCS client receiving an IM/Chat session request from a remote contact.

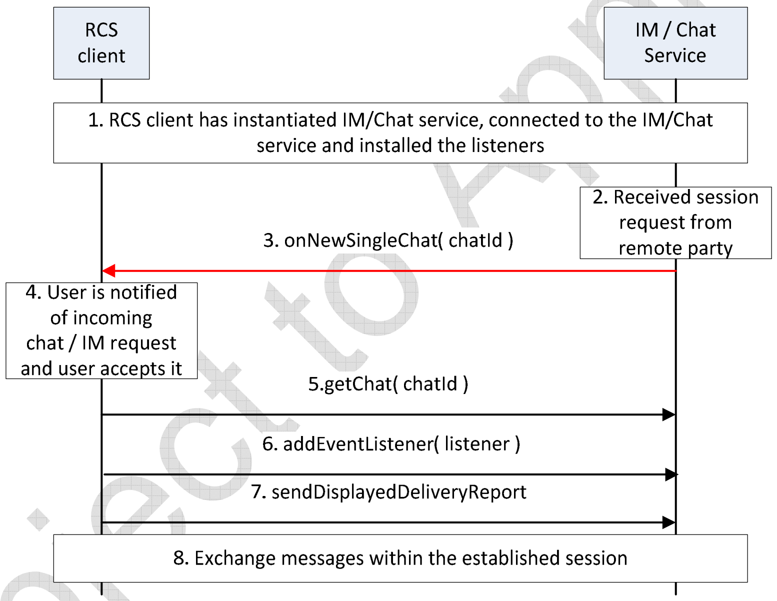

RCS client
IM / Chat Service
1. RCS client has instantiated IM/Chat service, connected to the IM/Chat service and installed the listeners
2. Received session request from remote party
3. onNewSingleChat(chatId)
4. User is notified of incoming chat / IM request and user accepts it
5.getChat(chatId)
6. addEventListener(listener)
7. sendDisplayedDeliveryReport
8. Exchange messages within the established session

Figure 5: Establish a one-to-one chat session with remote party

1. The RCS client instantiates a service instance of the IM/Chat Service, establishes a connection with the IM/Chat Service and associates the listener with this RCS client.
2. IM/Chat Service receives a request for a session from a remote party
3. IM/Chat Service notifies the RCS client through the invocation of the appropriate listener function
4. RCS client notifies the user by displaying the message received in the incoming session invitation to the user. The user accepts the session request
5. RCS client retrieves the chat object associated with this session
6. RCS client installs the listener function for this chat session
7. RCS client requests the IM/Chat service to send to the remote party, an indication of display of the message to the user
8. Using procedures described in Figure 6, the two users exchange IM / Chat messages

4.4.5.1.3 Message exchange after chat session establishment

Figure 6 is an example that shows the flow for a RCS client receiving a IM/Chat session request from a remote contact.

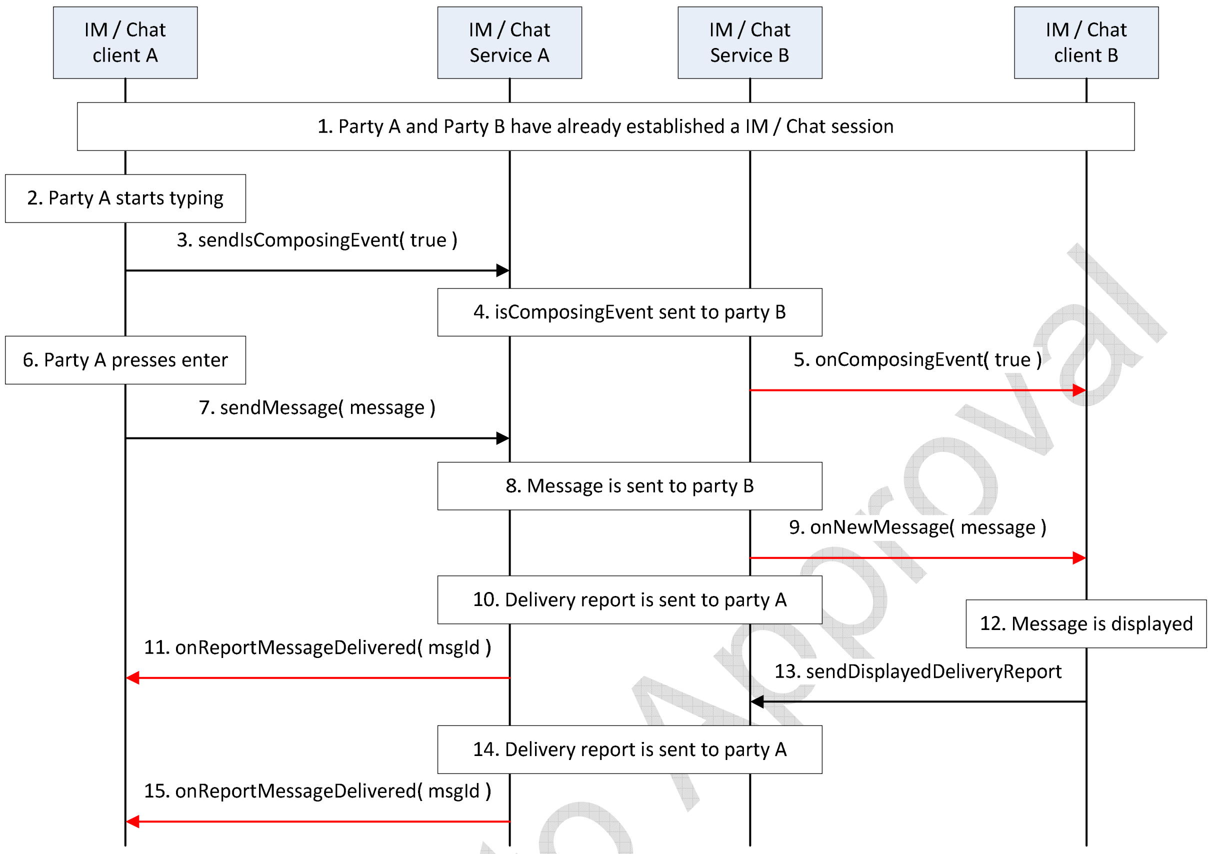

IM / Chat client A
IM / Chat Service A
IM / Chat Service B
IM / Chat client B
1. Party A and Party B have already established a IM / Chat session
2. Party A starts typing
3. sendIsComposingEvent(true)
4. isComposingEvent sent to party B
5. onComposingEvent(true)
6. Party A presses enter
7. sendMessage(message)
8. Message is sent to party B
9. onNewMessage(message)
10. Delivery report is sent to party A
11. onReportMessageDelivered(msgId)
12. Message is displayed
13. sendDisplayedDeliveryReport
14. Delivery report is sent to party A
15. onReportMessageDelivered(msgId)

Figure 6: Establish a one-to-one chat session with remote party

1. A party and B party have already established a IM/Chat session
2. Party A starts to type a new chat / IM message
3. Client A provides an indication of composing event to the IM/Chat service
4. IM/Chat service of party A provides indication of composing toward party B.
5. IM/Chat service of party B provides the indication to the IM client of party B
6. Party A completes typing the message and presses enter
7. Client A provides the message to the IM/Chat service A
8. IM/Chat service A sends the message to IM/Chat service B
9. IM/Chat service B provides an indication to the client B
10. IM/Chat service B provides delivery indication to IM/Chat service A
11. IM/Chat service A provides the delivery indication to IM client A
12. IM Client B displays the message to user
13. IM client B provides the displayed indication to the IM/Chat service B
14. IM/Chat service B provides the displayed indication to IM/Chat service A
15. IM/Chat service A provides the indication to the client A

4.4.5.2 Package

Package name **org.gsma.joyn.chat**

4.4.5.3 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: opens a single chat with a given contact and returns a Chat instance. The parameter contact supports the following formats: MSISDN in national or international format, SIP address, SIP-URI or Tel-URI.

  ```
  Chat openSingleChat(String contact, ChatListener listener)
  ```

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

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

- Method: rejoins an existing group chat from its unique chat ID.

  ```
  GroupChat rejoinGroupChat(String chatId)
  ```

- Method: restarts a previous group chat from its unique chat ID.

  ```
  GroupChat restartGroupChat(String chatId)
  ```

- Method: adds a listener on new chat invitation event.

  ```
  void addEventListener(NewChatListener listener)
  ```

- Method: removes a listener on new chat invitation event.

  ```
  void removeEventListener(NewChatListener listener)
  ```

- Method: returns the list of single chats in progress.

  ```
  Set<Chat> getChats()
  ```

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

  ```
  Chat getChat(String chatId)
  ```

- Method: returns a single chat from its invitation Intent.

  ```
  Chat getChatFor(Intent intent)
  ```

- Method: returns the list of group chats in progress.

  `Set<GroupChat> getGroupChats()`

- Method: returns a group chat in progress from its unique ID.

  `GroupChat getGroupChat(String chatId)`

- Method: returns a group chat from its invitation Intent.

  `GroupChat getGroupChatFor(Intentintent)`

- Method: returns the configuration for chat service.

  `ChatServiceConfiguration getConfiguration()`

Class **NewChatListener:**

This class offers callback methods on new chat invitation event.

- Method: callback called when a new chat invitation has been received

  `void onNewSingleChat(String chatId, ChatMessage message)`

- Method: callback called when a new group chat invitation has been received

  `void onNewGroupChat(String chatId)`

Class **ChatMessage**:
This class contains chat message information for single and group chat.

- Method: returns the contact who sends the message.

  `String getContact()`

- Method: returns the message ID.

  `String getId()`

- Method: returns the message content.

  `String getMessage()`

- Method: returns the receipt date.

  `Date getReceiptDate()`

- Method: is displayed delivery report requested.

  `boolean isDisplayedReportRequested()`

Class **Chat**:

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

- Method: returns the remote contact.

  ```
  String getRemoteContact()
  ```

- Method: sends a chat message. The method returns a unique message ID.

  ```
  String sendMessage(String message)
  ```

- Method: sends a displayed delivery report for a given message ID.

  ```
  void sendDisplayedDeliveryReport(String msgId)
  ```

- Method: sends an “is-composing” event. The status is set to true when typing a message, else it is set to false.

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

- Method: adds a listener on chat events.

  ```
  void addEventListener(ChatListener listener)
  ```

- Method: removes a listener on chat events.

  ```
  void removeEventListener(ChatListener listener)
  ```

Class **ChatListener**:
This class offers callback methods on 1-1 chat events.

- Method: callback called when a new message has been received

  ```
  void onNewMessage(ChatMessage message)
  ```

- Method: callback called when a message has been delivered to the remote.

  ```
  void onReportMessageDelivered(String msgId)
  ```

- Method: callback called when a message has been displayed by the remote.

  ```
  void onReportMessageDisplayed(String msgId)
  ```

- Method: callback called when a message has failed to be delivered to the remote.

  ```
  void onReportMessageFailed(String msgId)
  ```

- 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(boolean status)
  ```

Class **GroupChat**:

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

- Method: returns the chat ID.

  ```
  String getChatId()
  ```

- Method: returns the subject of the group chat.

  ```
  String getSubject()
  ```

- Method: returns the list of connected participants. A participant is identified by its MSISDN in national or international format, SIP address, SIP-URI or Tel-URI.

  ```
  Set<String> getParticipants()
  ```

- Method: returns the direction of the group chat: INCOMING, OUTGOING.

  ```
  int getDirection()
  ```

- Method: returns the state of the group chat. The state may have the following values: INVITED, INITIATED, STARTED, TERMINATED, ABORTED, FAILED.

  ```
  int getState()
  ```

- Method: accepts chat invitation.

  ```
  void acceptInvitation()
  ```

- Method: rejects chat invitation.

  ```
  void rejectInvitation()
  ```

- Method: sends a message to the group. This method returns a unique message ID.

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

- Method: sends a “is-composing” event. The status is set to true when typing a message, else it is set to false.

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

- Method: adds participants to a group chat.

  ```
  void addParticipants(Set<String> 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: quits a group chat conversation. The conversation will continue between other participants if there are enough participants.

  ```
  void quitConversation()
  ```

- Method: adds a listener on chat events.

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

- Method: removes a listener on chat events.

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

Class **GroupChatListener**:
This class offers callback methods on group chat events.

- Method: callback called when the session is well established and messages may be exchanged with the group of participants.

  ```
  void onSessionStarted()
  ```

- Method: callback called when a new message has been received

  ```
  void onNewMessage(ChatMessage message)
  ```

- Method: callback called when a message has failed to be delivered to the remote.

  ```
  void onReportMessageFailed(String msgId)
  ```

- Method: callback called when a new geoloc has been received

  ```
  void onNewGeoloc(GeolocMessage message)
  ```

- Method: callback called when a message has been delivered to the remote.

  ```
  void onReportMessageDelivered(String msgId)
  ```

- 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 contact, boolean status)
  ```

- Method: callback called when the session has been aborted or terminated.

  ```
  void onSessionAborted ()
  ```

- Method: callback called when the session has failed. The error code may be: CHAT_NOT_FOUND, INVITATION_DECLINED, CHAT_FAILED.

  ```
  void onSessionError(interror)
  ```

- Method: callback called when a new participant has joined the group chat.

```
void onParticipantJoined(String contact, String contactDisplayname)
```

- Method: callback called when a participant has left voluntary the group chat.

```
void onParticipantLeft(String contact)
```

- Method: callback called when a participant is disconnected from the group chat.

```
void onParticipantDisconnected(String contact)
```

Class **ChatServiceConfiguration**:
This class represents the particular configuration of IM Service.

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

```
boolean isImCapAlwaysOn()
```

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

```
boolean isImWarnSf()
```

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

```
int getChatSessionTimeout()
```

- Method: returns maximum participants in group chat session.

```
int getGroupChatMaxParticipantsNumber()
```

- Method: return maximum single chat message’s length can have

```
long getSingleChatMessageMaxLength()
```

- Method: returns the maximum single group chat message’s length can have

```
long getGroupChatMessageMaxLength()
```

- Method: returns the max number of simultaneous single chats.

```
int getMaxSingleChats()
```

- Method: returns the max number of simultaneous group chats.

```
int getMaxGroupChats()
```

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

```
boolean isSmsFallback()
```

- Method: returns True if auto accept mode activated for chat, else returns False.

  ```
  boolean isChatAutoAcceptMode()
  ```

- Method: returns True if auto accept mode activated for group chat, else returns False.

  ```
  boolean isGroupChatAutoAcceptMode()
  ```
- Method: activates or deactivates the displayed delivery report on received chat messages.

  ```
  void setDisplayedDeliveryReport(boolean state)
  ```

4.4.5.4 Intents

Intent broadcasted when a new chat conversation has been received. This Intent contains the following extras:

- "contact": MSISDN of the contact sending the invitation.
- "contactDisplayname": display name of the contact sending the invitation (extracted from the SIP address).
- "firstMessage": first chat message in the new conversation (parceable object). This may be a ChatMessage or a GeolocMessage.

```
org.gsma.joyn.chat.action.NEW_CHAT
```

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

- "contact": MSISDN of the contact sending the invitation.
- "contactDisplayname": display name of the contact sending the invitation (extracted from the SIP address).
- "chatId": unique ID of the group chat conversation.
- "subject": subject of the group chat (optional).

```
org.gsma.joyn.chat.action.NEW_GROUP_CHAT
```

4.4.5.5 Content Providers

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

The content provider has the following tables and columns:

- GROUPCHAT

No.	Data	Data Type	Description
1.	CHAT_ID	String	Id for chat room
2.	STATE	Integer	State of chat room (INACTIVE = 0, ACTIVE = 1, CLOSED_BY_USER = 2, FAILED = 3, PENDING = 4, DECLINED = 5)
3.	SUBJECT	String	Subject of the group chat room

No.	Data	Data Type	Description
4	DIRECTION	Integer	INCOMING, OUTGOING
5	TIMESTAMP	Long	Date of the invitation

- Message

No.	Data	Data Type	Description
1.	MESSAGE_ID	Long	Id of the message
2.	CHAT_ID	String	Id of chat room
3.	CONTACT_NUMBER	String	Contact number of sender
4.	BODY	String	Body of the message
5.	TIMESTAMP	Long	Time when message inserted
6.	TIMESTAMP_SENT	Long	Time when message sent. If 0 means not sent.
7.	TIMESTAMP_DELIVERED	Long	Time when message delivered. If 0 means not delivered.
8.	TIMESTAMP_ DISPLAYED	Long	Time when message displayed. If 0 means not displayed.
9.	MIME_TYPE	String	Multipurpose Internet Mail Extensions (MIME) type of message
10.	MESSAGE_STATUS	Integer	Status of chat message (UNREAD = 0, UNREAD_REPORT = 1, READ = 2, SENDING = 3, SENT = 4, FAILED = 5, TO_SEND = 6, BLOCKED = 7) Status of system message (INVITED = 0, ACCEPTED = 1, DECLINED= 2, TIMEOUT = 3, JOINED = 4, GONE = 5, DISCONNECT = 6, BUSY = 7, FAILED = 8)
11.	DIRECTION	Integer	Status direction of message (INCOMING = 0, OUTGOING = 1)
12.	MESSAGE_TYPE	Integer	Type of message (MESSAGE= 0, SYSTEM = 1)

Note:

Status values for chat session management in GROUPCHAT->STATE:

- INVITED : incoming chat.
- INITIATED : outgoing chat.
- STARTED : invitation has been accepted and chat is started.
- ABORTED: chat has been aborted.
- TERMINATED: chat has been terminated.
- CLOSED_BY_USER: ended session closed by user.
- FAILED: chat session has failed.

Status values for a message in MESSAGE->MESSAGE_STATUS:

- In case of a chat message (MESSAGE->MESSAGE_TYPE = MESSAGE):

  - UNREAD: the message is delivered.
  - UNREAD_REPORT: the message is delivered and a displayed delivery report is requested.
  - READ: message is read (i.e. DISPLAYED).
  - SENDING: message is in progress of sending.
  - SENT: message is sent.
  - FAILED: message is failed to be sent.
  - TO_SEND: message is queued to be sent by stack.
  - BLOCKED: message is a spam message.

- In case of a system message (MESSAGE->MESSAGE_TYPE = SYSTEM):

  - INVITED: invitation pending.
  - ACCEPTED: invitation has been accepted (i.e; JOINED).
  - DECLINED: the invitation has been declined on remote participant.
  - TIMEOUT: the invitation has been timeout.
  - JOINED: participant has joined the group.
  - GONE: participant has left the group (i.e. DEPARTED).
  - DISCONNECTED (i.e. BOOTED).
  - BUSY.
  - FAILED.

Type values of a message direction in MESSAGE->DIRECTION:
- INCOMING: a message from remote user (incoming message)
- OUTGOING: a message sent for a remote user / conference

Type values for a system message (conference subscription) MESSAGE->MESSAGE_TYPE:
- MESSAGE: the message represents a chat message.
- SYSTEM: the message is considered a system (events) message.

4.4.5.6 Permissions

Access to the Chat API is requires the following permissions:
- org.gsma.joyn.RCS_USE_CHAT: this is a new permission that governs access to the chat API, and is required both to receive and to send over an RCS chat session.
- org.gsma.joyn.RCS_READ_CHAT: this is a new permission that that is required by a client in order to read the chat history from the content provider.

4.4.6 File Transfer API

This API exposes all functionality related to transferring files via the File Transfer Service. It allows:
- Send a file transfer request
- Send a file transfer request with thumbnail (file icon)
- Receive notifications about incoming file transfer and file transfer events.
- Receive notifications upon file delivery
- Retrieve the list of all file transfers and their statuses for a specific contact
- Clean all file transfer history or single file transfers (including the transferred files if possible)

- Monitor a file transfer's progress.
- Cancel a file transfer in progress.
- Accept/reject an incoming file transfer request.
- Read configuration elements affecting file transfer
- Resume a file transfer

4.4.6.1 File Transfer API calling flow

The figures in this section contain basic call flows of the File Transfer service API.

4.4.6.1.1 Session establishment

Figure 7 is an example that shows the flow for a File Transfer (FT) client establishing a FT session with a remote contact.

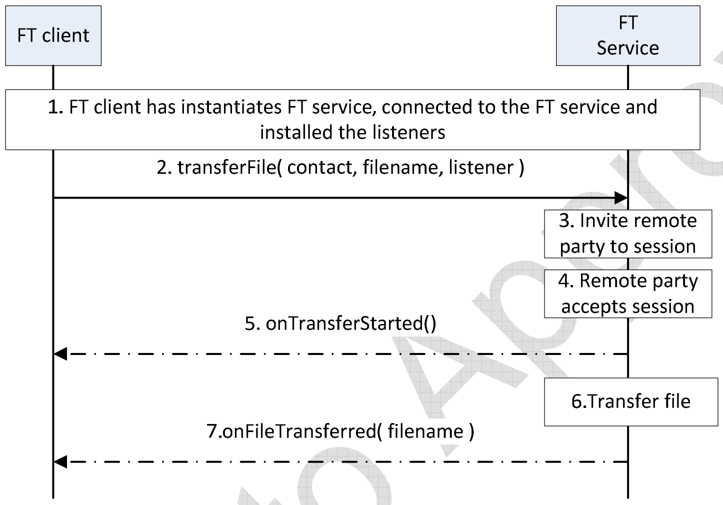

FT client
FT Service
1. FT client has instantiates FT service, connected to the FT service and installed the listeners
2. transferFile(contact, filename, listener)
3. Invite remote party to session
4. Remote party accepts session
5. onTransferStarted()
6.Transfer file
7.onFileTransferred(filename)

Figure 7: Successful sending of a file to remote party

1. The FT client instantiates a service instance of the File Transfer Service, establishes a connection with the FT Service and associates the listener with this RCS client.
2. By selecting a user and the file to be transferred, the user initiates a FT session request to the remote party
3. The FT service sends the session request to remote party
4. The session request is delivered to the remote party and indication is received that the remote user has accepted the file transfer request.
5. The FT client is notified that the file transfer has started.
6. The FT service transfers the file to the remote party.
7. The FT client is notified that the file transfer is completed.

4.4.6.1.2 Incoming session request

Figure 8 is an example that shows the flow for a FT client receiving a File Transfer request from a remote contact.

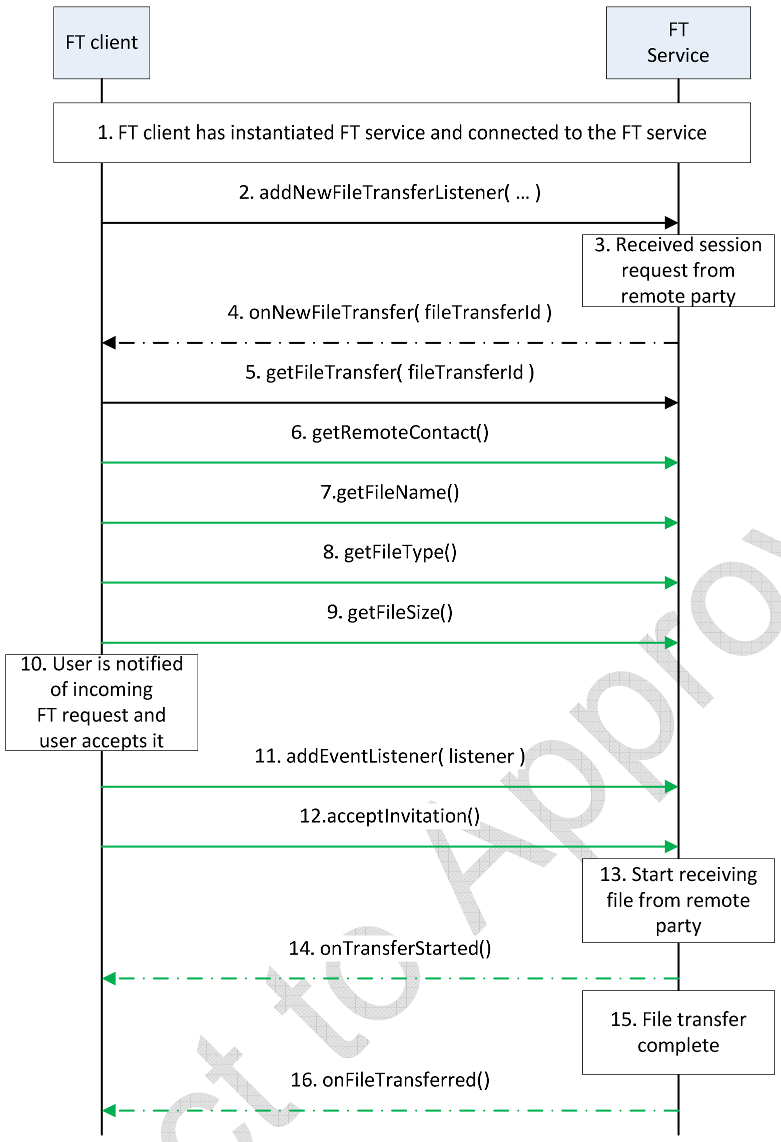

FT client
FT Service
1. FT client has instantiated FT service and connected to the FT service
2. addNewFileTransferListener(…)
3. Received session request from remote party
4. onNewFileTransfer(fileTransferId)
5. getFileTransfer(fileTransferId)
6. getRemoteContact()
7.getFileName()
8. getFileType()
9. getFileSize()
10. User is notified of incoming FT request and user accepts it
11. addEventListener(listener)
12.acceptInvitation()
13. Start receiving file from remote party
14. onTransferStarted()
15. File transfer complete
16. onFileTransferred()

Figure 8: Successful reception of a file from remote party

1. The FT client instantiates a service instance of the FT Service and establishes a connection with the FT Service.
2. The FT client associates the listener with this FT client.
3. The FT Service receives a request for a session from a remote party
4. The FT Service notifies the FT client through the invocation of the appropriate listener function
5. The FT client retrieves the file transfer session object from the FT service
6. The FT client retrieves the identity of the remote party that is sending the file.
7. The FT client retrieves the name of the file that is being sent by the remote party.
8. The FT client retrieves the type of the file that is being sent by the remote party
9. The FT client retrieves the size of the file that is being sent by the remote party.
10. The FT client notifies the user by displaying the details of the file that is being sent to the user. The user accepts the request,
11. The FT client installs the listener function for this FT session.
12. RCS client notifies the FT service that the user has accepted to receive the file
13. The FT service informs the remote end that it is ready to receive the file

14. The FT client is notified that the file transfer has started.
15. The FT service interacts with the remote end to receive the file
16. The FT client is notified that the file transfer is complete.

4.4.6.2 Package

Package name **org.gsma.joyn.ft**

4.4.6.3 Methods and Callbacks

Class **FileTransferService**:

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

- Method: connects to the API.

```
void connect()
```

- Method: disconnects from the API.

```
void disconnect()
```

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

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

- Method: returns a current file transfer from its unique ID.

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

- Method: returns a current file transfer from its invitation Intent.

```
FileTransfer getFileTransferFor(Intent intent)
```

- Method: transfers a file to a contact. The parameter contact supports the following formats: MSISDN in national or international format, SIP address, SIP-URI or Tel-URI. The parameter file contains the complete filename including the path to be transferred.

```
FileTransfer transferFile(String contact, String filename,
FileTransferListener listener)
```

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

```
FileTransfer transferFile(String contact, String filename, String
fileicon, FileTransferListener listener)
```

- Method: transfers a file to a group of contacts outside of a current group chat. The parameter file contains the complete filename including the path to be transferred. See also the method GroupChat.sendFile() of the Chat API to send a file from an existing group chat conversation.

```
FileTransfer transferFileToGroup(Set<String> contacts, String
filename, FileTransferListener listener)
```

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

  ```
  FileTransfer transferFileToGroup(Set<String> contacts, String
  filename, String fileicon, FileTransferListener listener)
  ```

- Method: returns the configuration for File Transfer service.

  ```
  FileTransferServiceConfiguration getConfiguration()
  ```

- Method: adds a new file transfer invitation listener.

  ```
  void addNewFileTransferListener(NewFileTransferListener listener)
  ```

- Method: removes a new file transfer invitation listener.

  ```
  void removeNewFileTransferListener(NewFileTransferListener listener)
  ```

Class **FileTransfer**:
This class maintains the information related to a file transfer and offers methods to manage the transfer.

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

  ```
  String getTransferId()
  ```

- Method: returns the remote contact.

  ```
  String getRemoteContact()
  ```

- Method: returns the direction of the transfer: INCOMING, OUTGOING.

  ```
  int getDirection()
  ```

- Method: returns the complete filename of the icon or thumbnail.

  ```
  String getFileIconName()
  ```

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

  ```
  String getFileName()
  ```

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

  ```
  String getFileSize()
  ```

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

  ```
  String getFileType()
  ```

- Method: returns the state of the file transfer. The state may have the following values: INVITED, INITIATED, STARTED, TRANSFERRED, ABORTED, FAILED, and PAUSED.

```
int getState()
```

- Method: accepts file transfer invitation.

```
void acceptInvitation()
```

- Method: rejects file transfer invitation.

```
void rejectInvitation()
```

- Method: aborts the file transfer.

```
void abortTransfer()
```

- Method: resume the file transfer.

  void resumeTransfer()

- Method: adds a listener on file transfer events.

```
void addEventListener(IFileTransferListener listener)
```

- Method: removes a listener from file transfer.

```
void removeEventListener(IFileTransferListener listener)
```

Class **FileTransferListener**:
This class offers callback methods on file transfer events.

- Method: callback called when the file transfer is started.

```
void onTransferStarted()
```

- Method: callback called when the file transfer has been aborted.

```
void onTransferAborted()
```

- Method: callback called when the transfer has failed. The error code may be: INVITATION_DECLINED, SAVING_FILE, and TRANSFER_FAILED.

```
void onTransferError(int error)
```

- Method: callback called during the transfer progress.

```
void onTransferProgress(long currentSize, long totalSize)
```

- Method: callback called when the file has been transferred. The parameter filename contains the complete filename including the path.

```
void onFileTransferred(String filename)
```

Class **NewFileTransferListener**:
This class offers callback method to receive new file transfer invitation.

- Method: callback called when a new file transfer invitation has been received.

  ```
  void onNewFileTransfer(String transferId)
  ```

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

  ```
  void onReportFileDelivered(String transferId)
  ```

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

  ```
  void onReportFileDisplayed(String transferId)
  ```

Class **FileTransferServiceConfiguration**:
This class represents the particular configuration of FT Service.

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

  ```
  long getWarnSize()
  ```

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

  ```
  long getMaxSize()
  ```

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

  ```
  boolean getAutoAcceptMode()
  ```

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

  ```
  int getMaxFileTransfers()
  ```

- Method: returns True if file icon (i.e. thumbnail) is supported, else returns False.

  ```
  boolean isFileIconSupported()
  ```

- **Method: returns the max size for a file icon (i.e. thumbnail).**

  ```
  long getMaxFileIconSize()
  ```

4.4.6.4 Intents

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

  - “contact”: MSISDN of the contact sending the invitation.
  - “contactDisplayname”: display name of the contact sending the invitation (extracted from the SIP address).
  - “transferId”: unique ID of the file transfer.

- “fileicon”: filename of the file icon (complete path).
- “filename”: filename of the file to be transferred.
- “filesize”: size of the file in bytes.
- ”filetype”: MIME type of the file.

```
org.gsma.joyn.ft.action.NEW_FILE_TRANSFER
```

4.4.6.5 Content Providers

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

The content provider has the following columns:

Data	Data type	Comment
FT_ID	TEXT	Unique file transfer identifier
CONTACT_NUMBER	TEXT	Contains the MSISDN of the remote contact
FILENAME	TEXT	Filename
TYPE	TEXT	MIME type of the file
FILEICON	TEXT	Filename
DIRECTION	Integer	Incoming transfer or outgoing transfer
FILE_SIZE	Long	File size in bytes
TRANSFERRED	Long	Size transferred in bytes
TIMESTAMP	Long	Date of the transfer
STATE	Integer	See note below for the list of states

Note :

State values for file transfer session management:

- INVITED: incoming session.
- INITIATED: outgoing session.
- STARTED: invitation has been accepted and transfer is started.
- TRANSFERRED: file has been transferred with success.
- ABORTED: session has been aborted.
- FAILED: session has failed.
- PAUSED : transfer has been paused and may be resumed

4.4.6.6 Permissions

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

- org.gsma.joyn.RCS_FILETRANSFER_RECEIVE: this is a new permission that is required by a client in order to handle the receipt of a file transferred from a remote party.
- org.gsma.joyn.RCS_FILETRANSFER_SEND: this is a new permission that is required by a client in order to initiate the transfer of a file transferred to a remote party.
- org.gsma.joyn.RCS_FILETRANSFER_READ: this is a new permission that is required by a client in order to read the file transfer history from the content provider.

4.4.7 Image Share API

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

- Send an image share request
- Receive notifications about 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 Image Share API calling flow

The figures in this section contain basic call flows of the Image Share service API.

4.4.7.1.1 Session establishment

Figure 9 is an example that shows the flow for an image share client establishing an image share session with a remote contact.

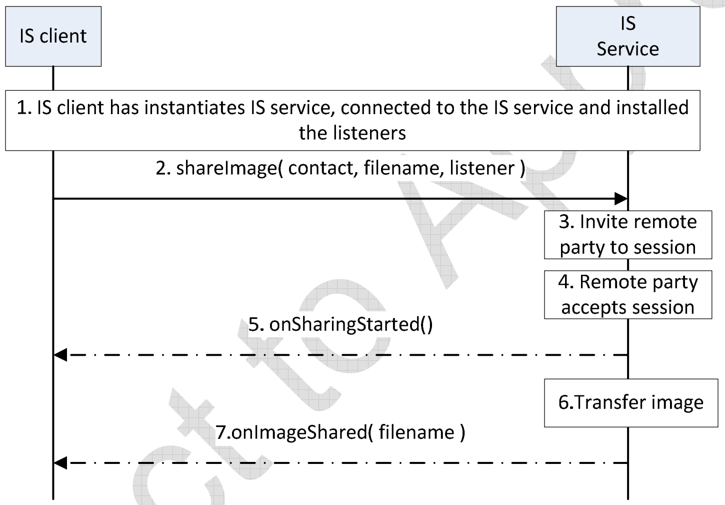

IS client
IS Service
1. IS client has instantiates IS service, connected to the IS service and installed the listeners
2. shareImage(contact, filename, listener)
3. Invite remote party to session
4. Remote party accepts session
5. onSharingStarted()
6.Transfer image
7.onImageShared(filename)

Figure 9: Successful sharing an image with remote party

1. The Image Share (IS) client instantiates a service instance of the Image Share Service, establishes a connection with the IS Service and associates the listener with this IS client.
2. By selecting a user and the image file to be transferred, the user initiates a IS session request to the remote party
3. IS service sends the session request to remote party
4. The session request is delivered to the remote party and indication is received that the remote user has accepted the image sharing request.
5. The IS client is notified that the image sharing has started.
6. The IS service transfers the image to the remote party.
7. The IS client is notified that the image sharing is completed.

4.4.7.1.2 Incoming session request

Figure 10 is an example that shows the flow for a IS client receiving a Image Share request from a remote contact.

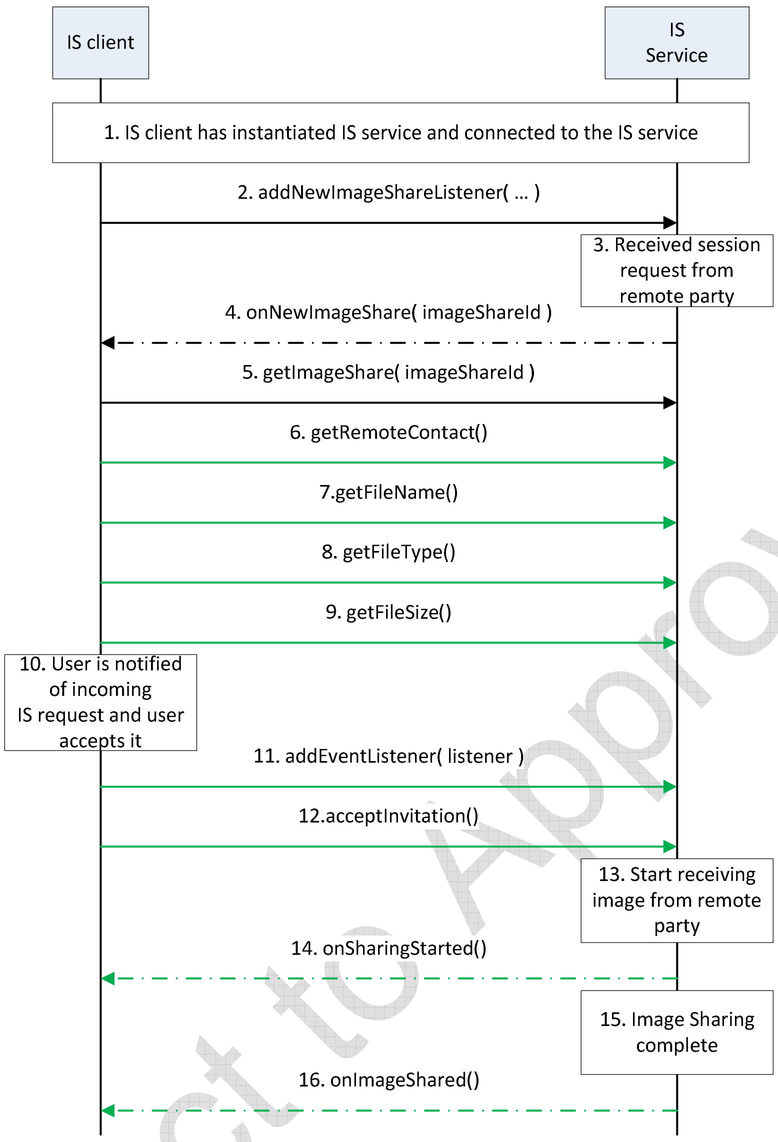

IS client
IS Service
1. IS client has instantiated IS service and connected to the IS service
2. addNewImageShareListener(…)
3. Received session request from remote party
4. onNewImageShare(imageShareId)
5. getImageShare(imageShareId)
6. getRemoteContact()
7.getFileName()
8. getFileType()
9. getFileSize()
10. User is notified of incoming IS request and user accepts it
11. addEventListener(listener)
12.acceptInvitation()
13. Start receiving image from remote party
14. onSharingStarted()
15. Image Sharing complete
16. onImageShared()

Figure 10: Successful sharing of an image from remote party

1. The IS client instantiates a service instance of the IS Service and establishes a connection with the IS Service.
2. The IS client associates the listener with this IS client.
3. IS Service receives a request for a session from a remote party
4. IS Service notifies the IS client through the invocation of the appropriate listener function
5. The IS client retrieves the image share session object from the FT service
6. The IS client retrieves the identity of the remote party that is sharing the image.
7. The IS client retrieves the name of the image file that is being shared by the remote party.
8. The IS client retrieves the type of the image file that is being shared by the remote party
9. The IS client retrieves the size of the image file that is being shared by the remote party.
10. The IS client notifies the user by displaying the details of the image that is being shared to the user. The user accepts the request,

11. The IS client installs the listener function for this IS session.
12. The IS client notifies the IS service that the user has accepted to receive the image sharing request
13. The IS service informs the remote end that it is ready to receive the image to be shared
14. The IS client is notified that the image share has started.
15. The IS service interacts with the remote end to receive the image being shared
16. The IS client is notified that the image sharing is complete.

4.4.7.2 Package

Package name **org.gsma.joyn.ish**

4.4.7.3 Methods and Callbacks

Class **ImageSharingService**:

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

- Method: connects to the API.

```
void connect()
```

- Method: disconnects from the API.

```
void disconnect()
```

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

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

- Method: returns a current image sharing from its unique ID.

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

- Method: returns a current image sharing from its invitation Intent.

```
ImageSharing getImageSharingFor(Intent intent)
```

- Method: shares an image with a contact. The parameter contact supports the following formats: MSISDN in national or international format, SIP address, SIP-URI or Tel-URI. The parameter file contains the complete filename including the path of the image to be shared. An exception if thrown if there is no on-going CS call.

```
ImageSharing shareImage(String contact, String filename,
ImageSharingListener listener)
```

- Method: returns the configuration for image share service.

```
ImageSharingServiceConfiguration getConfiguration()
```

- Method: adds a new image share invitation listener.

```
void addNewImageSharingListener(NewImageSharingListener listener)
```

- Method: removes a new image share invitation listener.

```
void removeNewImageSharingListener(NewImageSharingListener listener)
```

Class **ImageSharing**:

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

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

```
String getSharingId()
```

- Method: returns the remote contact.

```
String getRemoteContact()
```

- Method: returns the complete filename including the path 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 the file to be shared.

```
String getFileType()
```

- Method: returns the state of the image share. The state may have the following values: INVITED, INITIATED, STARTED, TRANSFERRED, ABORTED and FAILED.

```
int getState()
```

- Method: returns the direction of the sharing: INCOMING, OUTGOING.

```
int getDirection()
```

- Method: accepts image share invitation.

```
void acceptInvitation()
```

- Method: rejects image share invitation.

```
void rejectInvitation()
```

- Method: aborts the sharing.

```
void abortSharing()
```

- Method: adds a listener on image sharing events.

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

- Method: removes a listener from image sharing.

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

Class **ImageSharingListener**:
This class offers callback methods on image sharing events.

- Method: callback called when the sharing is started.

```
void onSharingStarted()
```

- Method: callback called when the sharing has been aborted.

```
void onSharingAborted()
```

- Method: callback called when the sharing has failed. **The error code may be: INVITATION_DECLINED, SAVING_FAILED and SHARING_FAILED.**

```
void onSharingError(int error)
```

- Method: callback called during the sharing progress.

```
void onSharingProgress(long currentSize, long totalSize)
```

- Method: callback called when the sharing has been terminated. The parameter filename contains the complete filename including the path.

```
void onImageShared(String filename)
```

Class **NewImageSharingListener**:
This class offers callback method to receive new image share invitation.

- Method: callback called when a new image share invitation has been received.

```
void onNewImageSharing(String sharingId)
```

Class **ImageSharingServiceConfiguration**:
This class represents the particular configuration of Image Sharing Service.

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

```
long getWarnSize()
```

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

```
long getMaxSize()
```

4.4.7.4 Intents

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

- "contact": MSISDN of the contact sending the invitation.
- "contactDisplayname": display name of the contact sending the invitation (extracted from the SIP address).
- "sharingId": unique ID of the image sharing.
- "filename": filename of the image to be transferred
- "filesize": size of the image file in bytes.
- "filetype": MIME_type of the image.

```
org.gsma.joyn.ish.action.NEW_IMAGE_SHARING
```

4.4.7.5 Content Providers

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

The content provider has the following columns:

Data	Data type	Comment
SHARING_ID	TEXT	Unique sharing identifier
CONTACT_NUMBER	TEXT	Contains the MSISDN of the remote contact
FILENAME	TEXT	Filename
TYPE	TEXT	MIME type of the file
DIRECTION	Integer	Incoming sharing or outgoing sharing
FILE_SIZE	Long	File size in bytes
TRANSFERRED	Long	Size transferred in bytes
TIMESTAMP	Long	Date of the sharing
STATE	Integer	See note below for the list of states

Note :

State values for file transfer session management :

- INVITED: incoming session.
- INITIATED: outgoing session.
- STARTED: invitation has been accepted and sharing is started.
- TRANSFERRED: session has been terminated and sharing successful.
- ABORTED: session has been aborted.
- FAILED: session has failed.

4.4.7.6 Permissions

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

- org.gsma.joyn.RCS_IMAGESHARE_RECEIVE: this is a new permission that is required by a client in order to handle the receipt of a image shared by a remote party.

- org.gsma.joyn.RCS_IMAGESHARE_SEND: this is a new permission that is required by a client in order to initiate the sharing of an image with a remote party.
- org.gsma.joyn.RCS_IMAGESHARE_READ: this is a new permission that is required by a client in order to read the image share history from the content provider.

4.4.8 Video Share API

This API exposes all functionality related to sharing 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.

4.4.8.1 Package

Package name **org.gsma.joyn.vsh**

4.4.8.2 Methods and Callbacks

Class **VideoSharingService**:

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

- Method: connects to the API.

```
void connect()
```

- Method: disconnects from the API.

```
void disconnect()
```

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

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

- Method: returns a current video sharing from its unique ID.

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

- Method: returns a current video share from its invitation Intent.

```
VideoSharing getVideoSharingFor(Intent intent)
```

- Method: shares a live video stream with a contact. The parameter contact supports the following formats: MSISDN in national or international format, SIP address, SIP-URI or Tel-URI. The parameter player contains a media player which streams over RTP the live video from the camera. The media player is just an interface which permits to have a player implementation independent from the joyn API. An exception if thrown if there is no on-going CS call.

```
VideoSharing shareVideo(String contact, VideoPlayer player,
VideoSharingListener listener)
```

- Method: returns the configuration for video share service.

```
VideoSharingServiceConfiguration getConfiguration()
```

- Method: adds a new video share invitation listener.

```
void addNewVideoSharingListener(NewVideoSharingListener listener)
```

- Method: removes a new video share invitation listener.

```
void removeNewVideoSharingListener(NewVideoSharingListener listener)
```

Class **VideoSharing**:

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

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

```
String getSharingId()
```

- Method: returns the remote contact.

```
String getRemoteContact()
```

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

```
String getVideoEncoding()
```

- Method: returns the video format (e.g. QCIF).

```
String getVideoFormat()
```

- Method: returns the state of the video share. The state may have the following values: INVITED, INITIATED, STARTED, TERMINATED, ABORTED, and FAILED.

```
int getState()
```

- Method: returns the direction of the sharing: INCOMING, OUTGOING.

```
int getDirection()
```

- Method: accepts video share invitation with a given renderer.

```
void acceptInvitation(VideoRenderer renderer)
```

- Method: rejects video share invitation.

```
void rejectInvitation()
```

- Method: aborts the sharing.

```
void abortSharing()
```

- Method: adds a listener on video sharing events.

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

- Method: removes a listener from video sharing.

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

Class **VideoSharingListener**:

This class offers callback methods on video sharing events.

- Method: callback called when the sharing is started.

```
void onSharingStarted()
```

- Method: callback called when the sharing has been aborted.

```
void onSharingAborted()
```

- Method: callback called when the sharing has failed. The error code may be: INVITATION_DECLINED, and SHARING_FAILED.

```
void onSharingError(int error)
```

Class **VideoPlayer**:

This class offers an interface to manage the video player instance independently of the joyn service. The video player is implemented in the application side.

- Method: opens the player and prepares resources (e.g. encoder, camera).

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

- Method: closes the player and de-allocates resources.

```
void close()
```

- Method: starts the player.

```
void start()
```

- Method: stops the player.

```
void stop()
```

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

```
int getLocalRtpPort()
```

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

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

- Method: adds a listener on video player events.

  ```
  void addEventListener(VideoPlayerListener listener)
  ```

- Method: removes a listener from video player.

  ```
  void removeEventListener(VideoPlayerListener listener)
  ```

Class **VideoPlayerListener:**
This class offers callback methods on video player events.

- Method: callback called when the player is opened.

  ```
  void onPlayerOpened()
  ```

- Method: callback called when the player is started.

  ```
  void onPlayerStarted()
  ```

- Method: callback called when the player is stopped.

  ```
  void onPlayerStopped()
  ```

- Method: callback called when the player is closed.

  ```
  void onPlayerClosed()
  ```

- Method: callback called when the player has failed.

  ```
  void onPlayerFailed()
  ```

Class **VideoRenderer**:
This class offers an interface to manage the video renderer instance independently of the joyn service. The video renderer is implemented in the application side.

- Method: opens the renderer and prepares resources (e.g. decoder, display).

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

- Method: closes the renderer and de-allocates resources.

  ```
  void close()
  ```

- Method: starts the renderer.

  ```
  void start()
  ```

- Method: stops the renderer.

  ```
  void stop()
  ```

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

```
int getLocalRtpPort()
```

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

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

- Method: adds a listener on video renderer events.

```
void addEventListener(VideoRendererListener listener)
```

- Method: removes a listener from video renderer.

```
void removeEventListener(VideoRendererListener listener)
```

Class **VideoRendererListener:**

This class offers callback methods on video renderer events.

- Method: callback called when the renderer is opened.

```
void onRendererOpened()
```

- Method: callback called when the renderer is started.

```
void onRendererStarted()
```

- Method: callback called when the renderer is stopped.

```
void onRendererStopped()
```

- Method: callback called when the renderer is closed.

```
void onRendererClosed()
```

- Method: callback called when the renderer has failed.

```
void onRendererFailed()
```

Class **VideoCodec**:

This class maintains the information related to a video codec.

- 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 width (e.g. 176).

```
int getVideoWidth()
```

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

```
int getVideoHeight()
```

- Method: returns a codec parameter from its key name (e.g. profile-level-id, packetization-mode).

```
String getParameter(String key)
```

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

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

```
int getMaxTime()
```

4.4.8.3 Intents

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

- “contact”: MSISDN of the contact sending the invitation.
- “contactDisplayname”: display name of the contact sending the invitation (extracted from the SIP address).
- “sharingId”: unique ID of the sharing.
- ”encoding”: video encoding (e.g. H264).
- “format”: video format (e.g. QCIF).

```
org.gsma.joyn.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.

The content provider has the following columns:

Data	Data type	Comment
SHARING_ID	TEXT	Unique sharing identifier
CONTACT_NUMBER	TEXT	Contains the MSISDN of the remote contact

Data	Data type	Comment
DIRECTION	Integer	Incoming sharing or outgoing sharing
TIMESTAMP	Long	Date of the sharing
STATE	integer	See note below for the list of states
DURATION	Long	Duration of the sharing in seconds. The value is only set at the end of the sharing.

Note :

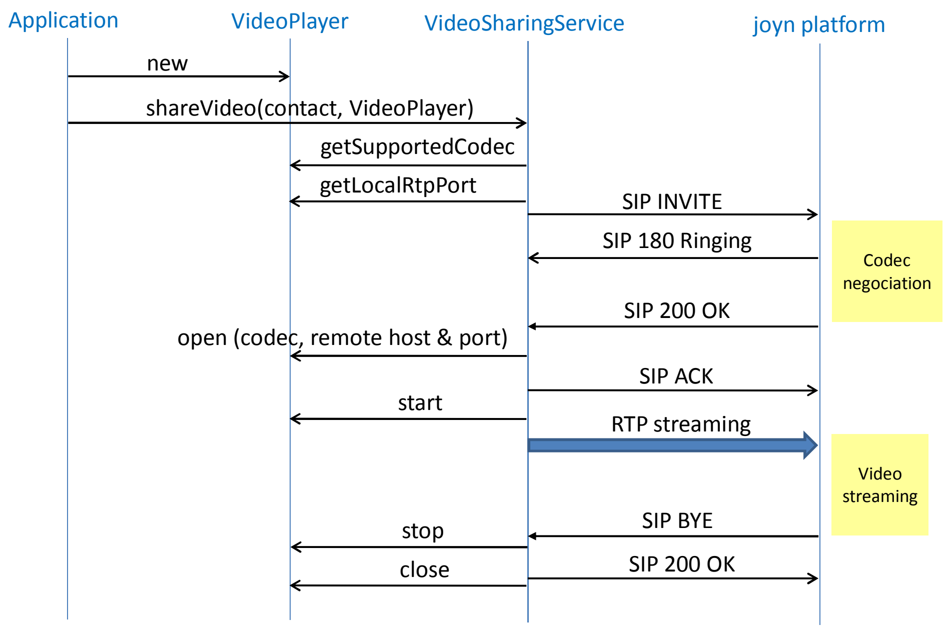
State values for video sharing session management:

- INVITED: incoming session.
- INITIATED: outgoing session.
- STARTED: invitation has been accepted and sharing is started.
- ABORTED: session has been aborted.
- TERMINATED: session has been terminated.
- FAILED: session has failed.

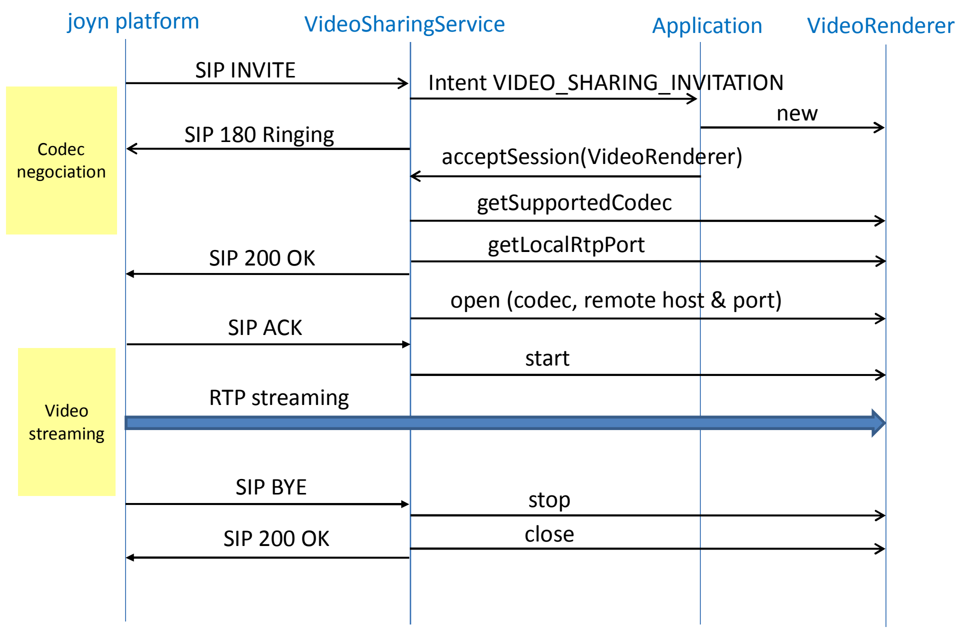
4.4.8.5 Sequence diagrams

The following diagrams explain the interactions between the video player or video renderer instantiated by the application and the joyn API.

Originating side:



Terminating side:



4.4.8.6 Permissions

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

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

4.4.9 Geoloc Share API

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

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

A geoloc contains the following information:

- a label associated to the geoloc info,
- latitude,
- longitude,
- altitude,
- 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 the Chat log in order to be displayed afterwards from the “Show us in a map” service.

4.4.9.1 Package

Package name org.gsma.joyn.gsh

4.4.9.2 Methods and Callbacks

Class **GeolocSharingService:**

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

- Method: connects to the API.

  ```
  void connect()
  ```

- Method: disconnects from the API.

  ```
  void disconnect()
  ```

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

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

- Method: returns a current geoloc sharing from its unique ID.

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

- Method: returns a current geoloc sharing from its invitation Intent.

  ```
  GeolocSharing getGeolocSharingFor(Intent intent)
  ```

- Method: shares a geoloc with a contact. The parameter contact supports the following formats: MSISDN in national or international format, SIP address, SIP-URI or Tel-URI. An exception is thrown if there is no ongoing CS call.

  ```
  GeolocSharing    shareGeoloc(String    contact,    Geoloc    geoloc,
    GeolocSharingListener listener)
  ```

- Method: adds a new geoloc sharing invitation listener.

  ```
  void addNewGeolocSharingListener(NewGeolocSharingListener listener)
  ```

- Method: removes a new geoloc sharing invitation listener.

  ```
  void          removeNewGeolocSharingListener(NewGeolocSharingListener
    listener)
  ```

Class **GeolocSharing:**

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

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

```
String getSharingId()
```

- Method: returns the remote contact.

```
String getRemoteContact()
```

- Method: returns the geoloc info to be shared.

```
Geoloc getGeoloc()
```

- Method: returns the state of the geoloc sharing. The state may have the following values: INVITED, INITIATED, STARTED, TRANSFERRED, ABORTED, and FAILED.

```
int getState()
```

- Method: returns the direction of the sharing: INCOMING, OUTGOING.

```
int getDirection()
```

- Method: accepts geoloc sharing invitation.

```
void acceptInvitation()
```

- Method: rejects geoloc sharing invitation.

```
void rejectInvitation()
```

- Method: aborts the sharing.

```
void abortSharing()
```

- Method: adds a listener on geoloc sharing events.

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

- Method: removes a listener from geoloc sharing.

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

Class **GeolocSharingListener:**

This class offers callback methods on geoloc sharing events.

- Method: callback called when the sharing is started.

```
void onSharingStarted()
```

- Method: callback called when the sharing has been aborted.

```
void onSharingAborted()
```

- Method: callback called when the sharing has failed. The error code may be: INVITATION_DECLINED, SAVING_FAILED, SHARING_FAILED.

```
void onSharingError(int error)
```

- Method: callback called during the sharing progress.

```
void onSharingProgress(long currentSize, long totalSize)
```

- Method: callback called when the sharing has been terminated. The parameter geoloc contains the geoloc info.

```
void onGeolocShared(Geoloc geoloc)
```

Class NewGeolocSharingListener:

This class offers callback method to receive new geoloc sharing invitation.

- Method: callback called when a new geoloc sharing invitation has been received.

```
void onNewGeolocSharing(String sharingId)
```

Note: the object “Geoloc” is already defined in the Chat API.

4.4.9.3 Intents

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

- “contact”: MSISDN of the contact sending the invitation.
- “contactDisplayname”: display name of the contact sending the invitation (extracted from the SIP address).
- “sharingId”: unique ID of the geoloc sharing.

```
org.gsma.joyn.gsh.action.NEW_GEOLOC_SHARING
```

4.4.9.4 Content Providers

The geoloc info shared during a call is stored in the Chat Log. There is no specific call log as we have with the Image/Video Sharing services.

4.4.9.5 Permissions

Geoloc Share is a convenience mechanism to allow geolocation information to be delivered in a chat message. From the point of view of any 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.
- org.gsma.joyn.RCS_LOCATION_SEND: this is a new permission that is required to send Geolocation data over an RCS chat session.
- org.gsma.joyn.RCS_USE_CHAT: this is the permission that governs access to the chat API which is a prerequisite to being able to use the Geoloc Share API.
- org.gsma.joyn.RCS_READ_CHAT: this is the permission that that is required by a client in order to read data relating to geloocation push events in the chat history from the content provider.

4.4.10 IP Call API

This API exposes all functionality related to manage the IP Call service. It allows to:

- Initiate an IP Call: voice only, voice & video.
- Receive notifications about incoming IP Call invitation.
- Monitors an IP Call (call hold, call events, etc.).
- Ends a current IP Call.
- Accept/reject an incoming IP Call.
- Read configuration elements affecting IP call.

The media engine which process audio and video samples is implemented by the application (via software or via hardware), in this case the API stays independent from the media engine implementation and offers more openness.

4.4.10.1 Package

Package name org.gsma.joyn.ipcall

4.4.10.2 Methods and Callbacks

Class **IPCallService:**

This class offers the main entry point to manage IP Calls. 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 IP calls in progress.

```
Set<IPCall> getIPCalls()
```

- Method: returns a current IP call from its unique ID.

```
IPCall getIPCall(String callId)
```

- Method: returns a current IP call from its invitation Intent.

```
IPCall getIPCallFor(Intent intent)
```

- Method: initiates an IP Call with a contact. The parameter contact supports the following formats: MSISDN in national or international format, SIP address, SIP-URI or Tel-URI. The parameters player and renderer contain respectively a media player and a media renderer which stream over RTP the live audio and live video. The media player/renderer are just interfaces which permit to have an implementation independent from the joyn API.

```
IPCall initiateCall(String contact, IPCallPlayer player,
IPCallRenderer renderer, IPCallListener listener)
```

- Method: returns the configuration for IP call service.

```
IPCallServiceConfiguration getConfiguration()
```

- Method: adds a new IP call invitation listener.

```
void addNewIPCallListener(NewIPCallListener listener)
```

- Method: removes a new IP call invitation listener.

```
void removeNewIPCallListener(NewIPCallListener listener)
```

Class **IPCall:**

This class maintains the information related to an IP Call and offers methods to manage the call.

- Method: returns the call ID of the IP call.

```
String getCallId()
```

- Method: returns the remote contact.

```
String getRemoteContact()
```

- Method: returns the state of the IP call. The state may have the following values: INVITED, INITIATED, STARTED, ABORTED, FAILED.

```
int getState()
```

- Method: returns the direction of the sharing: INCOMING, OUTGOING.

```
    int getDirection()
```

- Method: accepts IP call invitation.

```
    void acceptInvitation(IPCallPlayer player, IPCallRenderer renderer)
```

- Method: rejects IP call invitation.

```
    void rejectInvitation()
```

- Method: aborts the call.

```
    void abortCall()
```

- Method: Checks if the call is on hold.

```
    boolean isOnHold()
```

- Method:Puts a call on hold. If su

```
    void holdCall()
```

- Method:Continues a call that hold's on.

```
    void continueCall()
```

- Method: adds a listener on IP call events.

```
    void addEventListener(IPCallListener listener)
```

- Method: removes a listener from IP call.

```
    void removeEventListener(IPCallListener listener)
```

Class **IPCallListener:**

This class offers callback methods on IP call events.

- Method: callback called when the call is started.

```
    void onCallStarted()
```

- Method: callback called when the call has been aborted.

```
    void onCallAborted()
```

- Method: callback called when the call has failed. The error code may be: INVITATION_DECLINED, CALL_FAILED.

```
    void onCallError(int error)
```

- Method: callback called when the call has been held.

  ```
  void onCallHeld()
  ```

- Method: callback called when the call continues after on hold.

  ```
  void onCallContinue()
  ```

Class **NewIPCallListener:**

This class offers callback method to receive new IP call invitation.

- Method: callback called when a new IP call invitation has been received.

  ```
  void onNewIPCall(String callId)
  ```

Class **IPCallPlayer:**

This class offers an interface to manage the IP call player instance independently of the joyn service. The IP call player is implemented in the application side.

- Method: opens the player and prepares resources (e.g. encoder, microphone, camera).

  ```
  void open(AudioCodec audiocodec, VideoCodec videocodec, String
  remoteHost, int remoteAudioPort, int remoteVideoPort)
  ```

- Method: closes the player and deallocates resources.

  ```
  void close()
  ```

- Method: starts the player.

  ```
  void start()
  ```

- Method: stops the player.

  ```
  void stop()
  ```

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

  ```
  int getLocalAudioRtpPort()
  ```

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

  ```
  int getLocalVideoRtpPort()
  ```

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

  ```
  AudioCodec[] getSupportedAudioCodecs()
  ```

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

```
VideoCodec[] getSupportedVideoCodecs()
```

- Method: adds a listener on IP call player events.

```
void addEventListener(IPCallPlayerListener listener)
```

- Method: removes a listener from IP call player.

```
void removeEventListener(IPCallPlayerListener listener)
```

Class **IPCallPlayerListener**:

This class offers callback methods on IP call player events.

- Method: callback called when the player is opened.

```
void onPlayerOpened()
```

- Method: callback called when the player is started.

```
void onPlayerStarted()
```

- Method: callback called when the player is stopped.

```
void onPlayerStopped()
```

- Method: callback called when the player is closed.

```
void onPlayerClosed()
```

- Method: callback called when the player has failed.

```
void onPlayerFailed()
```

Class **IPCallRenderer:**

This class offers an interface to manage the IP call renderer instance independently of the joyn service. The IP call renderer is implemented in the application side.

- Method: opens the renderer and prepares resources (e.g. decoder, display).

```
void open(AudioCodec audiocodec, VideoCodec videocodec, String
remoteHost, int remoteAudioPort, int remoteVideoPort)
```

- Method: closes the renderer and deallocates resources.

```
void close()
```

- Method: starts the renderer.

```
void start()
```

- Method: stops the renderer.

  ```
  void stop()
  ```

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

  ```
  int getLocalAudioRtpPort()
  ```

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

  ```
  int getLocalVideoRtpPort()
  ```

- Method: returns the list of audio codecs supported by the renderer.

  ```
  AudioCodec[] getSupportedAudioCodecs()
  ```

- Method: returns the list of video codecs supported by the renderer.

  ```
  VideoCodec[] getSupportedVideoCodecs()
  ```

- Method: adds a listener on IP call renderer events.

  ```
  void addEventListener(IPCallRendererListener listener)
  ```

- Method: removes a listener from IP call renderer.

  ```
  void removeEventListener(IPCallRendererListener listener)
  ```

Class **IPCallRendererListener:**

This class offers callback methods on IP call renderer events.

- Method: callback called when the renderer is opened.

  ```
  void onRendererOpened()
  ```

- Method: callback called when the renderer is started.

  ```
  void onRendererStarted()
  ```

- Method: callback called when the renderer is stopped.

  ```
  void onRendererStopped()
  ```

- Method: callback called when the renderer is closed.

  ```
  void onRendererClosed()
  ```

- Method: callback called when the renderer has failed.

```
void onRendererFailed()
```

Class **AudioCodec:**

This class maintains the information related to an audiocodec.

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

```
String getEncoding()
```

- Method: returns the codec sample rate (e.g. 15).

```
int getSampleRate()
```

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

```
int getPayloadType()
```

- Method: returns a codec parameter from its key name (e.g. packetization-mode).

```
String getParameter(String key)
```

Class **VideoCodec:**

This class maintains the information related to a video codec.

- 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 width (e.g. 176).

```
int getVideoWidth()
```

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

```
int getVideoHeight()
```

- Method: returns a codec parameter from its key name (e.g. profile-level-id, packetization-mode).

  ```
  String getParameter(String key)
  ```

Class **IPCallServiceConfiguration:**

This class represents the particular configuration of IP Call Service.

- Method: returns True if the IP voice call service can reach any user or returns False if only joyn users supporting the capability may be called.

  ```
  boolean isVoiceCallBreakout()
  ```

4.4.10.3 Intents

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

- “contact”: MSISDN of the contact sending the invitation.
- “contactDisplayname”: display name of the contact sending the invitation (extracted from the SIP address).
- “callId”: unique ID of the IP call.

```
org.gsma.joyn.ipcall.action.NEW_CALL
```

4.4.10.4 Content Providers

A content provider is used to store the IP call history persistently. There is one entry per call.

The content provider has the following columns:

Data	**Data type**	**Comment**
CALL_ID	TEXT	Unique call identifier
CONTACT_NUMBER	TEXT	Contains the MSISDN of the remote contact
DIRECTION	Integer	Incoming sharing or outgoing sharing
TIMESTAMP	Long	Date of the sharing
STATE	integer	See note below for the list of states
DURATION	Long	Duration of the call in seconds. The value is only set at the end of the call.

Note :

State values for IP call session management :

- INVITED : incoming session.

- INITIATED : outgoing session.
- STARTED : invitation has been accepted and call is started.
- ABORTED: session has been aborted.
- TERMINATED: session has been terminated.
- FAILED : session has failed.

4.4.10.5 Permissions

Access to the IP Call API is requires the following permissions:

- org.gsma.joyn.RCS_USE_IPCALL this is a new permission that governs access to the IP Call API, and is required to initiate, receive and to manage RCS IP Call sessions.
- org.gsma.joyn.RCS_READ_IPCALL: this is a new permission that that is required by a client in order to read the IP Call history from the content provider.

4.4.11 Contacts API

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

- To add RCS info in the local address book,
- To extract RCS info from the local address book.

4.4.11.1 Package

Package name **org.gsma.joyn.contacts**

4.4.11.2 Methods and Callbacks

Class **ContactsService**:

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

- Method: connects to the API.

```
void connect()
```

- Method: disconnects from the API.

```
void disconnect()
```

- Method: returns the list of joyn contacts.

```
Set<joynContact> getjoynContacts()
```

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

```
Set<joynContact> getjoynContactsOnline()
```

- Method: returns the list of contacts supporting a given feature tag (i.e. capability).

```
Set<joynContact> getjoynContactsSupporting(String tag)
```

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

```
String getVCard(Uri contact)
```

Class **joynContact**:
This class maintains the information related to a joyn contact.

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

```
String getContactId()
```

- Method: returns the capabilities associated to the contact.

```
org.gsma.joyn.capability.Capabilities getCapabilities()
```

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

```
boolean isRegistered()
```

4.4.11.3 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 joyn in the address book.

See the following MIME-type to be supported:

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

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

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

4.4.11.4 Permissions

Access to the Contacts API is requires the following permissions:

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

4.4.12 API Versioning

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

A build is identified by:

- GSMA version: hotfixes, Blackbird, .etc.
- Implementor 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.

4.4.12.1 Package

Package name **org.gsma.joyn**

4.4.12.2 Methods and Callbacks

Class **Build**:

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

- Constant: GSMA version number from class Build.GSMA_CODES.

```
public final static int GSMA_VERSION
```

- Constant: API release implementer name.

```
public final static String API_CODENAME
```

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

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

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

  ```
  public final static int API_INCREMENTAL
  ```

Class **Build.GSMA_CODES**:
This class contains the list of GSMA versions.

- Constant: joyn hotfixes version

  ```
  public final static int RCSE_HOTFIXES_1_2
  ```

- Constant: joyn Blackbird version

  ```
  public final static int RCSE_BLACKBIRD
  ```

Class **Build.VERSION_CODES**:
This class contains the list of API versions.

- Constant: The original first version of joyn API

  ```
  public final static int BASE
  ```

4.5 Privileged Client API

4.5.1 Overview

This section lists all APIs that are joyn Client exclusive. Each of these APIs will only be used by Core Application. Each API exposes all its functionality on a high level and does put constraints on the invoking application as to the method preconditions.

This section will be defined in future version.

4.6 IMS APIs

4.6.1 Overview

This API exposes IMS core service functionalities.

This section will be defined in future version.

Annex A Document Management

A.1 Document History

Version	Date	Brief Description of Change	Approval Authority	Editor / Company
0.85	30 May 2013	Draft Version for internal review	RCS TSG JTA	Kelvin Qin and Tom Van Pelt /

				GSMA
0.9	02 Oct 2013	Joyn Blackbird release are incorporated	RCS TSG JTA	Kelvin Qin and Tom Van Pelt / GSMA

A.2 Other Information

Type	Description
Document Owner	RCS TSG JTA
Editor / Company	Kelvin Qin and Tom Van Pelt / 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@gsm.org

Your comments or suggestions & questions are always welcome.