internal Group

Edition 3.4

Checked by:

Date:

RCS-e stack

API Specification

Author:

Orange Labs
jeanmarc.auffret@orange.com

Date:
28-08-2013

Summary: Specification of the RCS API on top of the RCS-e stack. This API offers high level interface to implement RCS applications (i.e. UI, Widget).

DOCUMENT HISTORY

Version	Date	Remarks
1.0	27/05/2011	First RCS-e edition
1.1	07/06/2011	- New methods `getFilename` and `getFilesize` for a file transfer session. - New intent parameter `contactDisplayname` for session invitation and conference event listener. - RCS permission. - Add method `getRcsMimeTypes` to get MIME types associated to RCS contacts.
1.2	20/06/2011	- Add new setting “Default SIP port for Wi-fi access”. - Add new setting “Enable SIP keep-alive”. - Add new setting “SIP keep-alive period”. - Add a constant for RCS extensions in Capability API. - RCS extensions management.
1.3	25/07/2011	- Add SIP API. - Changes to the Event log API: o New method `getLastChatMessage` to get the last message of a given chat session. o New method `markChatMessageAsRead` to mark a message as being read. o New method `getNumberOfUnreadChatMessages` to get the number of unread messages for a given chat session. o New methods to manage the spam box: ▪ `getSpamBoxLogContentProviderUri` ▪ `markChatMessageAsSpam` ▪ `deleteAllSpams` - Add Intent parameter `filetype` for file transfer and image sharing invitation. - Add Intent parameter `videotype` for video share invitation. - Capability API: to avoid confusion, method `synchronizeAll` has been renamed to `refreshAllCapabilities`.

1.4	04/08/2011	- Update SIP API. - RCS extension permission. - Add a media codec in the media API. - Add a new parameter `chatSessionId` in the file transfer invitation to correlate with an existing chat session. - Add a new parameter `replacedSessionId` in the chat invitation Intent which is used when a 1-1 session has been extended to a group chat (terminating side). - Add a new Intent `CHAT_SESSION_REPLACED` which is used when a 1-1 session has been extended to a group chat (originating side). - Method `sendMessage` returns a message ID now. - New settings to define the IM session startup mode. - Add codec attributes in Media API.
1.5	30/08/2011	- Add method `getRcsContactsAvailable` in Contacts API. - Rename methods returning the SDP content in the SIP API. - Change RCS extension declaration. - New settings for RFC5626 (Registration retry base time & Registration retry max time). - Add a method `getSessionState` in all the session. - Add new methods `setMultipartyCall` and `setCallHold` in the rich call API. - New settings to manage the max number of entries for chat history and richcall history.
1.6	15/09/2011	- Remove duplicate settings (Presence service activation & Rich call service activation) which are already ported by the corresponding capabilities. - Add a new methods `getSupportedMediaCodecs` & `setMediaCodec` in the media API to support codec negociation. - Add a class `SessionState` to define session state constants. - Intent parameter `subject` replaced by `firstMessage`. - Method `getSubject` replaced by `getFirstMessage`.
1.7	20/09/2011	- Remove parameter contact in methods `handleMessageDeliveryStatus` and `setMessageDeliveryStatus` of the class messaging API. - Add a new methods and class in the messaging API which permits to deliver reports when there is no ongoing chat session: `setMessageDeliveryStatus` `addMessageDeliveryListener`, `removeMessageDeliveryListener`, `IMessageDeliveryListener`.
1.8	07/10/2011	- Add a new method `isStoreAndForward` from a chat session and chat invitation Intent. - Add methods `startRcsService` and `stopRcsService` in the class `ClientApi`.
1.9	14/11/2011	- Modify method `getLastChatMessage` to return an `InstantMessage` object. - New settings for GRUU activation.
1.10	07/12/2011	- Add new settings: SIP trace filename, Always-on CPU flag.
1.11	17/01/2012	- New settings for auto config mode.

1.12	30/01/2012	- Add terms and conditions API. - Modify P-CSCF address settings for mobile and Wi-Fi access (split address and port).
1.13	06/02/2012	- Add `videowidth` and `videoheight` in the intent `VIDEO_SHARING_INVITATION`. - New settings for end user confirmation request URI. - Modify methods `acceptTerms` and `rejectTerms`.
1.14	21/02/2012	- New settings for country area code.
2.0	06/03/2012	- Add new methods in the messaging API to rejoin an existing group chat session.
2.1	13/04/2012	- Rename Intent `com.orangelabs.rcs.SERVICE_REGISTRATION` by `com.orangelabs.rcs.IMS_STATUS`.
2.2	03/05/2012	- Add a new method in the `InstantMessage` class to get the server receipt date of the message.
2.3	16/05/2012	- Update the method `rejoinChatGroupSession` in the messaging API: use a session ID instead of a chat ID as a parameter.
2.4	03/08/2012	- New settings for auto accept file transfer. - New intent parameter `autoAccept` for file transfer invitation. - Remove calls in Events log.
2.5	14/08/2012	- Remove first message in the group chat session. - Rename methods (`isChatGroup` by `isGroupChat`, `rejoinChatGroupsSession` by `rejoinGroupChatSession`) and intent parameter (`isChatGroup` by `isGroupChat`). - Add new method `restartGroupChatSession`. - Add new method `getSubject` to a chat session. - New intent parameter `autoAccept` for chat. - Add new method `getContactInfo` in the Capabilities API. - Add new method `getMaxAddedParticipants` for a group chat session. - Add new states (departed, booted, busy, declined) in the method `handleConferenceEvent`. - New settings for auto accept group chat. - New intent parameter `autoAccept` for group chat invitation.
2.6	26/10/2012	- Add new method `getGroupChatSessions`. - Add new method `getGroupChatSessionsWith`. - Add new settings for device ID management. - Add API for terms notification. - Add new method `deleteGroupChatConversation` - Extend MIME types for RCS extensions.
2.7	15/12/2012	- Remove `setWeblinkVisitedForContact and hasWeblinkBeenUpdatedForContact.`

2.8	21/12/2012	- Add a parameter `reason` in methods `handleSessionAborted`. - Add a new class `Build` which contains information related to the current API release. - Add reason in `handleImsDisconnected` and `SERVICE_STATUS` intent. - Add a new class `SessionTerminationReason` containing the session termination reasons. - Prepare Blackbird API definition.
2.9	19/02/2013	- Add a class `ImsDisconnectionReason`. - Add method `getLastGeoloc` in the event log API. - Support of geolocation push in the Messaging API.
3.0	20/02/2013	- Support of file transfer thumbnail for file transfer and image share service. - Add new attribute `accuracy` in the `GeolocPush` object. - Add method `getVisitCard` in the Contacts API. - Add methods `setFtBlockedForContact` and `isContactFtBlocked` in Contacts API to block file transfer capability.
3.1	21/05/2013	- Remove video pre-recorded support. - Remove Start/Stop Joyn service.
3.2	09/07/2013	- Add IP Call API. - API definitions is removed and replaced by the generated Javadoc. - Update list of settings.
3.3	01/08/2013	- Group File transfer API. - Update IP Call API description.
3.4	28/08/2013	- Rename IMediaPlayer to IVideoPlayer. - Rename IMediaRenderer to IVideoRenderer. - Rename IMediaEventListener to IVideoEventListener. - New Blackbird parameters.

CONTENTS

FIGURES

1. Introduction

1.1. Purpose of the document

This document describes the RCS API of the RCS-e stack. This API offers high level interface to implement RCS applications (i.e. UI, Widget).

1.2. Reference documents

N°	Title	Release
1	RCS-e specification release documents: http://www.gsmworld.com/our-work/mobile_lifestyle/rcs/index.htm	1.1
2	Google Android SDK: http://developer.android.com/index.html	2.x

1.3. Abbreviations

Abbreviation	Name
IMS	IP Multimedia Subsystem
CSh	Content sharing
EAB	Enhanced Address Book
SIP	Session Initiation Protocol
RTP	Real Time Protocol
MSRP	Media Session Relay Protocol
AIDL	Android Inter-process communication protocol

1.4. Terminology

Term	Description
Activity	Android UI application
Service	Android background process
Intent	Android description of an operation to be performed
Intent filter	Structured description of an intent to be matched
Broadcast receiver	An application class that listens for Intents that are broadcasted
Early IMS	GIBA authentication

2. RCS API

2.1. Overall description

The RCS-e stack is implemented into an Android background service which offers a high level API: the RCS API.

The RCS API is a client/server interface based on database providers, AIDL API & Intents. Several UI may be connected at a time to manage RCS events and to interact with the single stack instance running in background.

The RCS API permits to implement RCS application (e.g. enhanced address book, content sharing, chat, widgets) by hiding RCS protocols complexity.

The RCS API offers the following API:

- Terms and conditions API: accept/reject terms by end user.
- Capability API: contact capabilities discovery.
- Contacts API: RCS contact management and integration with the native address book.
- Presence API: social presence sharing, presence subscription & publishing, anonymous fetch.
- Rich call API: image sharing & video sharing during CS call.
- IP Call API: VoIP and Visio in best effort.
- Messaging API: 1-1 chat, group chat and file transfer.
- Media API: media player & renderer.
- Events log API: chat & file transfer history, rich call history and aggregation with classic log (calls, SMS, MMS).
- RCS settings database provider: application and stack settings.

2.2. Common API definition

The RCS API uses the following Android concepts:

- Intents mechanism to broadcast incoming events (e.g. notification) and incoming invitations to any Android activity or broadcast receiver which are declared in the device.
- AIDL interfaces to initiate and to manage session in real time (start, session monitoring, stop). Session events are managed thanks to callback mechanism.

Methods of the RCS API throw an exception if the IMS core layer is not initialized or not registered to the IMS platform.

Note: Remote application exceptions are not yet supported by the AIDL SDK, a generic AIDL exception is thrown instead.

How to use Intents?

By dynamically registering an instance of a class with `Context.registerReceiver()` or by using the `<receiver>` tag in your `AndroidManifest.xml`. Then the type of requested event is fixed in the Intent filter associated to the receiver, each RCS API has its own list of available intents.

How to use the AIDL interface?

The AIDL interface is hidden by an interface which main goal is to connect to the AIDL server side and to monitor the connection with it.

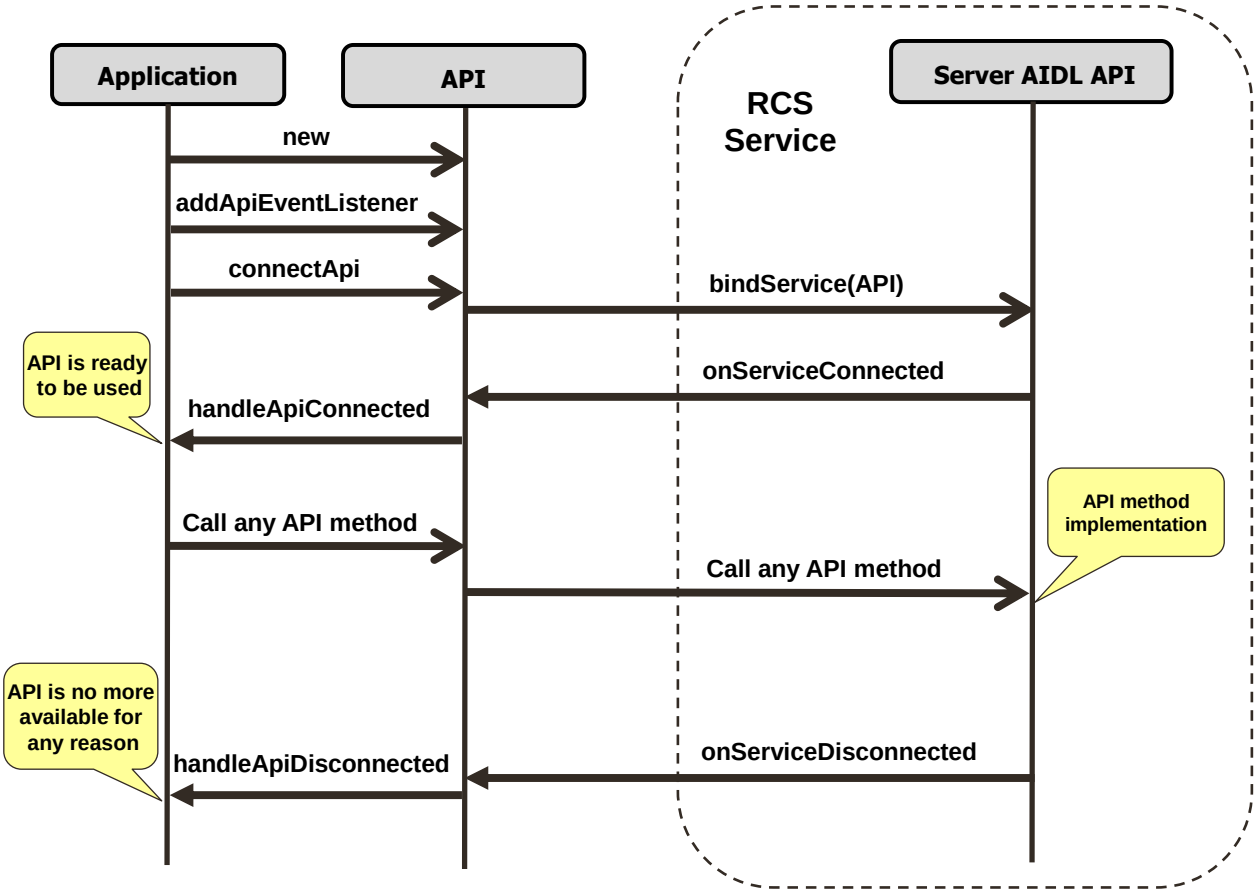

Figure 1: Common API architecture

Note: an exception is thrown if the API is not yet initialized when calling a method.

The RCS API has also API callbacks to monitor in real time the IMS connection status:

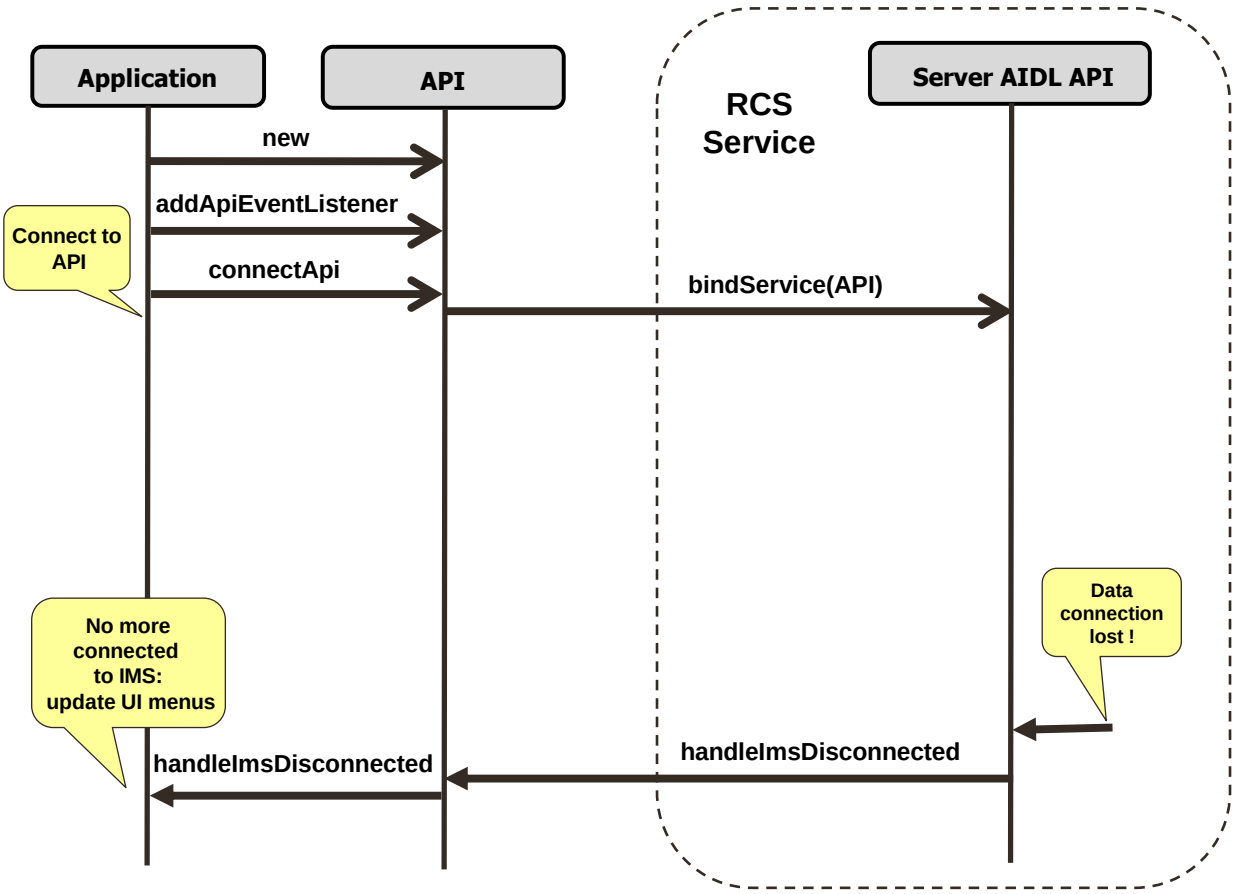

Figure 2: IMS connection monitoring

2.3. RCS permissions

2.3.1. RCS permission

An application uses the classic RCS API should declare the following permission in its manifest file:

```
<uses-permission android:name="com.orangelabs.rcs.permission.RCS"/>
```

An application using this permission should be signed (i.e. protection is “signature level”) with the same certificate as the RCS stack. This permits to avoid third party applications to call RCS API which is for native applications only.

2.3.2. RCS extension permission

An application uses the Generic SIP API should declare the following permission in its manifest file:

```
<uses-permission android:name="com.orangelabs.rcs.permission.RCS_EXTENSION"/>
```

Here there is no protection at the signature level. This permits to any third party application declaring this permission to use the Generic SIP API and to offer new services on top of the RCS stack.

2.4. Contact identity

All contacts are formatted into international format by the RCS API. So any contact format (Phone number: national or international, SIP-URI, Tel-URI) may be used as an input to the RCS API methods.

3. Common API

3.1. Description

This API is common to all other API offering an AIDL interface: Capability API, Presence API, Rich call API and messaging API.

This API is used to manage the connection to the Android service implementing the RCS-e stack (.i.e server part of the RCS API).

For example, this API may be useful:

- To detect if the Android service has been shutdown in order to disable a menu in UI part.
- To check if the API is well connected to the Android service.

See classes:

- `ClientApi`
- `ClientApiUtils`
- `ClientApiListener`
- `ClientApiIntents`
- `ClientApiException`
- `CoreServiceNotAvailableException`

3.2. API

See generated Javadoc in the SDK.

4. IMS API

4.1. Description

This API is common to all other API offering an AIDL interface: Capability API, Presence API, Rich call API, messaging API and generic SIP API.

This API is used to manage the connection with the IMS platform.

For example, this API may be used:

- To detect an IMS disconnection in order to disable a menu in UI part.
- To get the current IMS connection status in order to enable or not a RCS menu.

See classes:

- `ImsEventListener`
- `ImsApiIntents`

4.2. API

See generated Javadoc in the SDK.

5. Capability API

5.1. Description

This API permits to discover capabilities supported by contacts of the address book.

For example, this API may be used:

- To request capability update for a user when opening its contact card in the address book.
- To synchronize capabilities of all the contacts from the RCS account management menu.

See classes:

- `CapabilityApi`
- `Capabilities`
- `CapabilityApiIntents`

5.2. API

See generated Javadoc in the SDK.

6. Contacts API

6.1. Description

This API is an abstraction of the internal RCS database which contains all the RCS info associated to each contact of the address book.

A contact has the following RCS info:

- Type of contact (RCS-e compliant, Share presence, .etc).
- Supported Capabilities (Image share, Chat, .etc).
- Social presence info (fretext, photo-icon, .etc).

The additional RCS info for contacts are linked into the native address book database thanks to the ContactContract API of the Android SDK (from 2.x).

Note: this API is not based on an AIDL interface and may be used even if the RCS service is stopped.

See classes:

- `ContactsApi`
- `ContactInfo`

6.2. API

See generated Javadoc in the SDK.

7. Presence API

7.1. Description

This API permits to manage social presence info and relationships with its RCS community or RCS contacts.

This API is optional since RCS-e.

See classes:

- `PresenceApi`
- `PresenceApiIntents`
- `PresenceInfo`
- `PhotoIcon`
- `FavoriteLink`
- `Geoloc`

7.2. API

See generated Javadoc in the SDK.

8. Rich call API

8.1. Description

The API permits to share contents during a CS call (i.e. rich call service). This API should be used in coordination with the Capability API to discover if the remote contact supports “Image share” and “video share”. The capability discovering is automatically initiated by the RCS stack when the call is established, then the rich call application has just to catch the result to update the button “Share” in the dialer application.

See also the media API for video player and vide recorder.

See classes:

- `RichCallApi`
- `RichCallApiIntents`
- `IVideoSharingSession`
- `IVideoSharingEventListener`
- `IImageSharingSession`
- `IImageSharingEventListener`

8.2. API

See generated Javadoc in the SDK.

9. Messaging API

9.1. Description

This API permits to offer chat (one-to-one chat and group chat) and file transfer services.

The chat service supports:

- Delivery report (“message has been delivered”, “message has been displayed”).
- Is-composing features.
- Add one or several participants to the current conversation.
- Conference event monitoring (“someone has joined the session”, “someone has left the session”).
- Rejoin an existing group chat session.

9.2. API

See generated Javadoc in the SDK.

10.IP Call API

10.1. Description

This API permits to offer IP communications (Voice over IP and Video over IP).

The IP call service supports:

- IP audio call.
- IP audio + video call.
- Add/remove video on IP current call.
- Call hold.

10.1. API

See generated Javadoc in the SDK.

11. SIP API

11.1. Description

This API permits to implement new additional IMS services without any impact in the RCS stack. This generic SIP API manages only the signaling flow and is independent from the media part which should be supported by the application using the SIP API.

The IMS service implemented thanks to the SIP API is identified by a feature tag.

Outgoing usecase:

1. An outgoing INVITE request is sent by using the SIP API and by using a feature tag associated to the requested IMS service.
2. The session may be managed via the SIP API: cancel the session, session update, terminate the session.

Incoming usecase:

1. An incoming INVITE request is received in the RCS-e stack.
2. A feature tag is extracted from the Contact header of the incoming request.
3. A SIP Intent based on the feature tag is broadcasted on the device.
4. If an application triggers the SIP Intent, the incoming session invitation is processed (e.g. UI, sound, .etc) and may be accepted (i.e. 200 OK) or rejected (e.g. 486 Busy) by the application or by the ned user.
5. If the incoming session is accepted, the SIP API permits to manage the session: session update, terminate the session.

11.2. API

See generated Javadoc in the SDK.

12. Event log API

12.1. Description

This API permits to access to the following history:

- Rich call history.
- Chat history.
- File transfer history.
- Event history which is an aggregation of the previous history and the classic history (SMS, and MMS).

See class:

- `EventsLogApi`

12.2. API

See generated Javadoc in the SDK.

13. Media API

13.1. Description

The media API permits to connect the media player and media renderer to the RCS stack independently of the media itself. From this abstraction, the RCS stack manages (i.e. start, stop) the media stream thanks to the SIP call flow (e.g. the stack starts the media only after the SIP ACK).

The media API permits also to forward media error to the stack in order to stop the session (e.g. SIP BYE).

The supported video encodings are H.263+ and H.264 in low quality (QCIF).

This API contains by default the following media entities which may be completely replaced by another implementation (e.g. native implementation using hardware codecs) without any impact in the RCS stack:

- A live video player using the Camera.
- A video renderer.

The default video codec are software codecs implemented in C++ (source code from Android “opencore” framework) and integrated by using a JNI interface.

Optimization from last API release: the RTP transport layer is part of the media player or renderer in order to avoid to pass each RTP sample via the AIDL interface to the RCS stack.

See classes:

- `IMediaPlayer`
- `IMediaRenderer`
- `IMediaEventListener`
- `LiveVideoPlayer`
- `VideoPlayerEventListener`

13.2. API

See generated Javadoc in the SDK.

14. Terms and conditions API

14.1. Description

This API permits to accept or reject the received user terms and conditions.

14.2. API

See generated Javadoc in the SDK.

15.RCS settings

15.1. Description

This API permits to access to all the RCS stack parameters:

- Some parameters are read only and can be modified by the end user.
- Some parameters are only used by the UI part.
- Some parameters are only used by the RCS stack part.

Some parameters may be changed via the OTA interface (e.g. P-CSCF address).

See class:

- `RcsSettings`

15.2. API

See generated Javadoc in the SDK.

16. RCS extensions

RCS permits to discover in real time the capabilities supported by remote contacts by exchanging SIP OPTIONS messages between devices.

RCS-e has specified the way to extend the predefined supported capabilities (IM, File transfer, .etc) by defining an additional feature tag dedicated to RCS extensions which is also exchanged via a SIP OPTIONS message.

The RCS-e stack automatically detects when a new RCS extensions (e.g. a game, a whiteboard service, .etc) is installed or removed from the device. So the RCS-e stack adapts in real time its supported RCS extensions which are exchanged with other remote contacts via SIP OPTIONS.

There are two kinds of extensions: MNO (Mobile Network Operator) extensions which start with a prefix **+g.3gpp.iari-ref="urn%3Aurn-7%3A3gpp-application.ims.iari.rcse.xxx**" and generic extensions which don't have a predefined RCS prefix (e.g. common VoIP service).

An application which wants to implement a RCS or SIP/IMS extension detected by the RCS-e stack has just to add a specific Intent and MIME type in its Android manifest file.

16.1. MNO extensions

MNO extensions are associated to feature tags like **+g.3gpp.iari-ref="urn%3Aurn-7%3A3gpp-application.ims.iari.rcse.xxx.yyy**", where « xxx » is the MNO id and where « 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="com.orangelabs.rcs.capability.EXTENSION"/>
  <data android:mimeType="+g.3gpp.iari-ref/urn%3Aurn-7%3A3gpp-
application.ims.iari.rcse.xxx.yyy"/>
</intent-filter>
```

16.2. Generic extension sample

Generic extensions are associated to feature tags having these different syntaxes:

1) Feature tags like « +XXX »:

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

```
<intent-filter>
  <action android:name="com.orangelabs.rcs.capability.EXTENSION"/>
  <data android:mimeType="+XXX/*"/>
</intent-filter>
```

2) Feature tags with an associated value like « +XXX="YYY" »:

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

```
<intent-filter>
  <action android:name="com.orangelabs.rcs.capability.EXTENSION"/>
  <data android:mimeType="+XXX/YYY"/>
</intent-filter>
```

17.GSMA API

This API is specified by GSMA and permits to connect other applications with existing RCS applications.

There are two API:

1) GSMA UI Connector: this is an API which permits to link third party applications to the RCS application. For more info see the document "**GSMA RCS-e UI connector Guidelines 1.0**".

2) GSMA client API: this API permits to detect if a device is RCS compliant or not. For more info see the document "**GSMA Implementation guidelines 3.0, chapter 2.3**".