internal Group

Reference NSM/FT/R&D/MAPS/DVC/xx-09

Edition 1.10

Checked by:

Date:

RCS-e stack

API Specification

Approved by:

Author:

AUFFRET Jean-Marc (ASC Devices)
jeanmarc.auffret@orange-ftgroup.com

Date :

Date:
2011-09-20

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.

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:

- 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.
- 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.

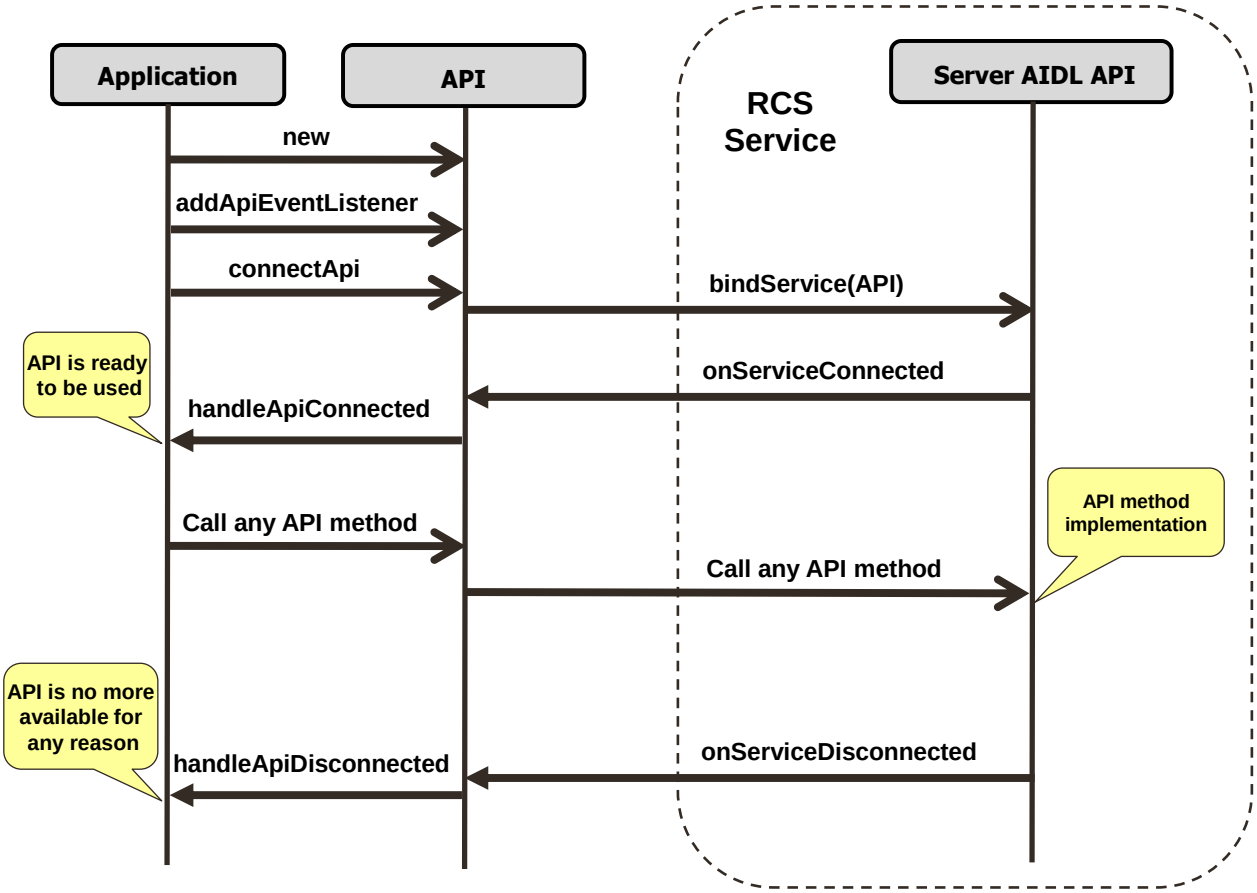

Figure 1: Common API architecture

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:

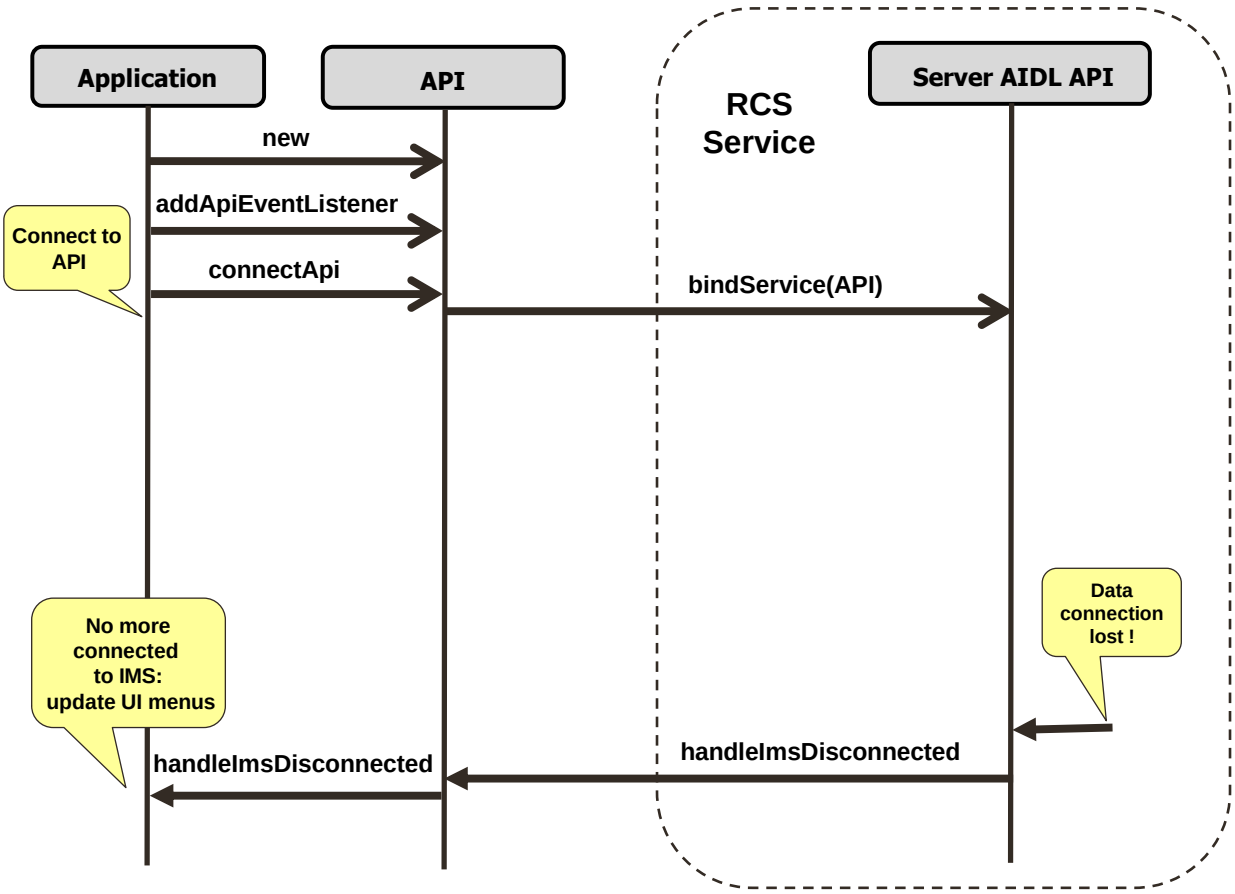

Figure 2: IMS connection monitoring

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`
- `ClientApiListener`
- `ClientApiIntents`
- `ClientApiException`
- `CoreServiceNotAvailableException`

3.2. API

```
// Is service started:
// - Parameter "ctx": application context.
// - Returns True if the service is running in background, else returns False.
boolean isServiceStarted(Context ctx);

// Start the RCS service
void startRcsService();

// Stop the RCS service
void stopRcsService();

// Add an API event listener:
// - Parameter "listener": API event listener.
void addApiEventListener(ClientApiListener listener);

// Remove an API event listener:
// - Parameter "listener": API event listener.
void removeApiEventListener(ClientApiListener listener);

// Remove all API event listeners:
void removeAllApiEventListeners();

// Client API event listener:
interface ClientApiListener {
    // API is disabled (e.g. server not started)
    public void handleApiDisabled();

    // API is connected to the server
    public void handleApiConnected();

    // API is disconnected from the server
    public void handleApiDisconnected();
}
```

3.3. Intents declaration

```
// Intent which permits to load the RCS settings application:
"com.orangelabs.rcs.SETTINGS"

// Intent broadcasted when the RCS service status has changed:
// - Parameter "status": starting, started, stopping, stopped, failed
"com.orangelabs.rcs.SERVICE_STATUS"
```

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 and messaging 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:

- `ImsApi`
- `ImsEventListener`

4.2. API

```
// Add an IMS event listener:
// - Parameter "listener": IMS event listener.
public void addImsEventListener(ImsEventListener listener)

// Remove an IMS event listener:
// - Parameter "listener": IMS event listener.
public void removeImsEventListener(ImsEventListener listener)

// IMS event listener interface:
public interface ImsEventListener {
    // Service is connected to the IMS
    public void handleImsConnected();

     // Service is disconnected from the IMS
    public void handleImsDisconnected();
}

// Is connected to IMS:
// - Returns True if Service is registered to IMS, else returns False.
public boolean isImsConnected();
```

4.3. Intents

```
// Intent broadcasted when the registration state has changed:
// - Parameter "status": connected, disconnected
"com.orangelabs.rcs.SERVICE_REGISTRATION"
```

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

```
// Connect API:
void connectApi();

// Disconnect API:
void disconnectApi();

// Get my capabilities:
// - Returns current capabilities.
Capabilities getMyCapabilities();

// Get contact capabilities:
// - Parameter "contact": remote contact.
// - Returns current capabilities of a contact.
Capabilities getContactCapabilities(String contact);

// Request capabilities for a given contact:
// - Parameter "contact": remote contact.
// - Returns current capabilities of a contact and request a new network update
in background.
Capabilities requestCapabilities(String contact);

// Refresh capabilities for all contacts:
public void refreshAllCapabilities();

// Capabilities object:
class Capabilities {
    // Image sharing support
    boolean imageSharing;

    // Video sharing support
    boolean videoSharing;

    // IM session support
    boolean imSession;

    // File transfer support
    boolean fileTransfer;

    // CS video support
    boolean csVideo;
```

```
    // Presence discovery support
    boolean presenceDiscovery = false;

    // Social presence support
    boolean socialPresence;

    // List of supported extensions
    ArrayList<String> extensions;

    // Last capabilities update
    long timestamp;
}
```

5.3. Intents

```
// Intent broadcasted when contact capabilities has changed:
// - Parameter "contact": remote contact.
// - Parameter "capabilities": object containing the capabilities.
"com.orangelabs.rcs.capability.CONTACT_CAPABILITIES"

// Intent broadcasted to discover capability extensions:
"com.orangelabs.rcs.capability.EXTENSION"

// RCS-e extension prefix
"urn%3Aurn-7%3A3gpp-application.ims.iari.rcse.orange"
```

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

```
// Get list of supported MIME types for RCS contacts
// - Returns MIME types.
String[] getRcsMimeTypes();

// Get contact info:
// - Parameter "contact": remote contact.
// - Returns contact info or null if not found.
ContactInfo getContactInfo(String contact);

// Get a list of all RCS contacts having social presence info:
// - Returns a list of contacts.
List<String> getRcsContactsWithSocialPresence();

// Get a list of all RCS contacts:
// - Returns a list of contacts.
List<String> getRcsContacts();

// Get a list of RCS contacts which are available:
// - Returns a list of contacts.
List<String> getRcsContactsAvailable();

// Get a list of all RCS blocked contacts:
// - Returns a list of contacts.
List<String> getRcsBlockedContacts();

// Get a list of all RCS invited contacts:
// - Returns a list of contacts.
List<String> getRcsInvitedContacts();

// Get a list of all RCS willing contacts:
// - Returns a list of contacts.
List<String> getRcsWillingContacts();

// Get a list of all RCS cancelled contacts:
// - Returns a list of contacts.
```



```
List<String> getRcsCancelledContacts();

// Get the IM-blocked status of a contact:
// - Parameter "contact": remote contact.
// - Returns True if the contact if blocked for IM, else returns False.
boolean isContactImBlocked(String contact);

// Is the number in the RCS blocked list:
// - Parameter "number": phone number.
// - Returns True if the number in blocked list, else returns False.
boolean isNumberBlocked(String number);

// Is the number in the RCS buddy list:
// - Parameter "number": phone number.
// - Returns True if the number in granted list, else returns False.
boolean isNumberShared(String number);

// Has the number been invited to RCS:
// - Parameter "number": phone number.
// - Returns True if the number in invited list, else returns False.
boolean isNumberInvited(String number);

// Has the number invited us to RCS:
// - Parameter "number": phone number.
// - Returns True if the number in willing list, else returns False.
boolean isNumberWilling(String number);

// Has the number invited us to RCS then be cancelled:
// - Parameter "number": phone number.
// - Returns True if the number has been cancelled, else returns False.
boolean isNumberCancelled(String number);

// Set the IM-blocked status of a contact:
// - Parameter "contact": remote contact.
// - Parameter "status": blocked state.
void setImBlockedForContact(String contact, boolean status);

// Get list of blocked contacts for IM sessions:
// - Returns a list of contacts.
List<String> getBlockedContactsForIm();

// Get list of contacts that can use IM sessions:
// - Returns a list of contacts.
List<String> getImSessionCapableContacts();

// Get list of contacts that can do use rich call features:
// - Returns a list of contacts.
List<String> getRichcallCapableContacts();

// Set the weblink visited flag to true for given contact:
// - "contact": remote contact.
void setWeblinkVisitedForContact(String contact);

// Get the weblink visited flag:
// - "contact": remote contact.
boolean hasWeblinkBeenUpdatedForContact(String contact);

// Remove a cancelled presence invitation:
// - "contact": remote contact.
void removeCancelledPresenceInvitation(String contact);

// Contact info object
class ContactInfo {
```

```
    // Capabilities
    Capabilities capabilities;

    // Presence info
    PresenceInfo presenceInfo;

    // Contact URI
    String contact;

    // Registration state
    boolean isRegistered;

    // RCS status (not RCS | RCS capable)
    String rcsStatus;

    // RCS status timestamp
    long rcsStatusTimestamp;
}
```

Note: `Capabilities` object is defined in the Capability API.

Note: `PresenceInfo` object is defined in the Presence API.

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

```
// Connect API:
void connectApi();

// Disconnect API:
void disconnectApi();

// Get my presence info:
// - Returns social presence info.
PresenceInfo getMyPresenceInfo();

// Set my presence info:
// - Parameter "info": social presence info.
// - Returns True if successful, else returns False.
boolean setMyPresenceInfo(PresenceInfo info);

// Invite a contact to share its presence:
// - "contact": remote contact.
// - Returns True if successful, else returns False.
boolean inviteContact(String contact);

// Accept the sharing invitation from a contact:
// - "contact": remote contact.
// - Returns True if successful, else returns False.
boolean acceptSharingInvitation(String contact);

// Reject the sharing invitation from a contact:
// - "contact": remote contact.
// - Returns True if successful, else returns False.
boolean rejectSharingInvitation(String contact);

// Ignore the sharing invitation from a contact:
// - "contact": remote contact.
void ignoreSharingInvitation(String contact);

// Revoke a shared contact:
// - "contact": remote contact.
// - Returns True if successful, else returns False.
boolean revokeContact(String contact);

// Unrevoke a revoked contact:
```

```
// - "contact": remote contact.
// - Returns True if successful, else returns False.
boolean unrevokeContact(String contact);

// Unblock a blocked contact:
// - "contact": remote contact.
// - Returns True if successful, else returns False.
boolean unblockContact(String contact);

// Get the list of granted contacts:
// - Returns a list of contacts.
List<String> getGrantedContacts();

// Get the list of revoked contacts:
// - Returns a list of contacts.
List<String> getRevokedContacts();

// Get the list of blocked contacts:
// - Returns a list of contacts.
List<String> getBlockedContacts();

// Presence info object
class PresenceInfo {
    // Presence timestamp
    long timestamp;

    // Presence status (online, offline)
    String status;

    // Free text
    String freetext;

    // Favorite link
    FavoriteLink favoriteLink;

    // Photo icon
    PhotoIcon photo;

    // Geoloc
    Geoloc geoloc;
}

// Photo-icon object
class PhotoIcon {
    // Photo content
    byte[] content;

    // Image type
    String type;

    // Width
    int width;

    // Height
    int height;

    // Etag
    String etag;
}

// Favorite link object
class FavoriteLink {
    // Link
```

```
    String link;

    // Name
    String name;
}

// Geoloc object
class Geoloc {
    // Latitude
    double latitude;

    // Longitude
    double longitude;

    // Altitude
    double altitude;
}
```

7.3. Intents

```
// Intent broadcasted when a presence sharing invitation has been received:
// - Parameter "contact": remote contact.
"com.orangelabs.rcs.presence.PRESENCE_SHARING_INVITATION"

// Intent broadcasted when user presence info has changed:
"com.orangelabs.rcs.presence.MY_PRESENCE_INFO_CHANGED"

// Intent broadcasted when a contact info has changed:
// - Parameter "contact": remote contact.
"com.orangelabs.rcs.presence.CONTACT_INFO_CHANGED"

// Intent broadcasted when a contact photo-icon has changed:
// - Parameter "contact": remote contact.
"com.orangelabs.rcs.presence.CONTACT_PHOTO_CHANGED"

// Intent broadcasted when a presence sharing info has changed:
// - Parameter "contact": remote contact.
// - Parameter "status": sharing status.
// - Parameter "reason": reason associated to the status.
"com.orangelabs.rcs.presence.PRESENCE_SHARING_CHANGED"
```

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

```
// Connect API:
void connectApi();

// Disconnect API:
void disconnectApi();

// Initiate a live video sharing session:
// - Parameter "contact": remote contact.
// - Parameter "player": media player.
// - Returns a video sharing session.
IVideoSharingSession initiateLiveVideoSharing(String contact, IMediaPlayer
player);

// Initiate a pre-recorded video sharing session:
// - Parameter "contact": remote contact.
// - Parameter "file": file to be streamed.
// - Parameter "player": media player.
// - Returns a video sharing session.
IVideoSharingSession initiateVideoSharing(String contact, String file,
IMediaPlayer player);

// Get a video sharing session from its session ID:
// - Parameter "id": session ID.
// - Returns a video sharing session or null if not found.
IVideoSharingSession getVideoSharingSession(String id);

// Initiate an image sharing session with a contact:
// - Parameter "contact": remote contact.
// - Parameter "file": file to be transferred.
// - Returns an image sharing session.
IImageSharingSession initiateImageSharing(String contact, String file);

// Get an image sharing session from its session ID:
// - Parameter "id": session ID.
// - Returns an image sharing session or null if not found.
IImageSharingSession getImageSharingSession(String id);
```



```
// Set multiparty call:
// - Parameter "flag": Boolean flag.
void setMultiPartyCall(boolean flag);

// Set call hold:
// - Parameter "flag": Boolean flag.
void setCallHold(boolean flag);

// Image sharing session:
interface IImageSharingSession {
    // Get session ID
    String getSessionID();

    // Get remote contact
    String getRemoteContact();

    // Get session state
    int getSessionState();

    // Get filename
    String getFilename();

    // Get filesize
    long getFilesize();

    // Accept the session invitation
    void acceptSession();

    // Reject the session invitation
    void rejectSession();

    // Cancel the session
    void cancelSession();

    // Add session listener
    void addSessionListener(IImageSharingEventListener listener);

    // Remove session listener
    void removeSessionListener(IImageSharingEventListener listener);
}

// Image sharing event listener:
interface IImageSharingEventListener {
    // Session is started
    void handleSessionStarted();

    // Session has been aborted
    void handleSessionAborted();

    // Session has been terminated by remote
    void handleSessionTerminatedByRemote();

    // Content sharing progress
    void handleSharingProgress(long currentSize, long totalSize);

    // Content sharing error
    void handleSharingError(int error);

    // Image has been transferred
    void handleImageTransfered(String filename);
}

// Video sharing session:
```

```
interface IVideoSharingSession {
    // Get session ID
    String getSessionID();

    // Get remote contact
    String getRemoteContact();

    // Get session state
    int getSessionState();

    // Accept the session invitation
    void acceptSession();

    // Reject the session invitation
    void rejectSession();

    // Cancel the session
    void cancelSession();

    // Set the media renderer (only used for incoming session)
    void setMediaRenderer(IMediaRenderer renderer);

    // Add session listener
    void addSessionListener(IVideoSharingEventListener listener);

    // Remove session listener
    void removeSessionListener(IVideoSharingEventListener listener);
}
```

// Video sharing event listener:

```
interface IVideoSharingEventListener {

    // Session is started
    void handleSessionStarted();

    // Session has been aborted
    void handleSessionAborted();

    // Session has been terminated by remote
    void handleSessionTerminatedByRemote();

    // Content sharing error
    void handleSharingError(int error);
}
```

// Session state

```
interface SessionState {
    // Session state is unknown (i.e. session dialog path does not exist)
    int UNKNOWN;

    // Session has been cancelled (i.e. SIP CANCEL exchanged)
    int CANCELLED;

    // Session has been established (i.e. 200 OK/ACK exchanged)
    int ESTABLISHED ;

    // Session has been terminated (i.e. SIP BYE exchanged)
    int TERMINATED ;

    // Session is pending (not yet accepted by a final response by the remote)
    int PENDING ;
}
```

8.3. Intents

```
// Intent broadcasted when a new image sharing invitation has been received:
// - Parameter "contact": remote contact.
// - Parameter "contactDisplayname": display name of remote contact.
// - Parameter "sessionId": session ID of the incoming session.
// - Parameter "filename": file name to be transferred.
// - Parameter "filesize": file size to be transferred.
// - Parameter "filetype": MIME-type of the file to be transferred.
 "com.orangelabs.rcs.richcall.IMAGE_SHARING_INVITATION"

// Intent broadcasted when a new video sharing invitation has been received:
// - Parameter "contact": remote contact.
// - Parameter "contactDisplayname": display name of remote contact.
// - Parameter "sessionId": session ID of the incoming session.
// - Parameter "videotype": MIME-type of the video to be streamed.
 "com.orangelabs.rcs.richcall.VIDEO_SHARING_INVITATION"
```

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”).

9.2. API

```
// Connect API:
void connectApi();

// Disconnect API:
void disconnectApi();

// Transfer a file to a contact:
// - Parameter "contact": remote contact.
// - Parameter "file": file to be transferred.
// - Returns a file transfer session.
IFileTransferSession transferFile(String contact, String file);

// Get the file transfer session from its session ID:
// - Parameter "id": session ID.
// - Returns a file transfer session or null if not found.
IFileTransferSession getFileTransferSession(String id);

// Get list of file transfer sessions with a contact:
// - Parameter "contact": remote contact.
// - Returns a file transfer session or null if not found.
List<IBinder> getFileTransferSessionWith(String contact);

// Get list of file transfer sessions:
// - Returns a list of sessions.
List<IBinder> getFileTransferSessions();

// File transfer session:
interface IFileTransferSession {
    // Get session ID
    String getSessionID();

    // Get remote contact
    String getRemoteContact();

    // Get session state
    int getSessionState();

    // Get filename
    String getFilename();

    // Get filesize
    long getFilesize();

    // Accept the session invitation
    void acceptSession();
```

```
    // Reject the session invitation
    void rejectSession();

    // Cancel the session
    void cancelSession();

    // Add session listener
    void addSessionListener(IFileTransferEventListener listener);

    // Remove session listener
    void removeSessionListener(IFileTransferEventListener listener);
}

// File transfer event listener:
interface IFileTransferEventListener {
    // Session is started
    void handleSessionStarted();

    // Session has been aborted
    void handleSessionAborted();

    // Session has been terminated by remote
    void handleSessionTerminatedByRemote();

    // Data transfer progress
    void handleTransferProgress(long currentSize, long totalSize);

    // Transfer error
    void handleTransferError(int error);

    // File has been transfered
    void handleFileTransfered(String filename);
}

// Initiate a one-to-one chat session with a contact:
// - Parameter "contact": remote contact.
// - Parameter "firstMsg": first message exchanged during the session.
// - Returns a chat session.
IChatSession initiateOne2OneChatSession(String contact, String firstMsg);

// Initiate an ad-hoc group chat session with a list of participants:
// - Parameter "participants": list of participants.
// - Parameter "firstMsg": first message exchanged during the session.
// - Returns a chat session.
IChatSession initiateAdhocGroupChatSession(List<String> participants, String
firstMsg);

// Get a chat session from its session ID:
// - Parameter "id": session ID.
// - Returns a chat session or null if not found.
IChatSession getChatSession(String id);

// Get list of chat sessions with a contact:
// - Parameter "contact": remote contact.
// - Returns a list of sessions.
List<IBinder> getChatSessionsWith(String contact);

// Get list of chat sessions:
// - Returns a list of sessions.
List<IBinder> getChatSessions();

// Set message delivery status:
```

```
// - Parameter "contact": remote contact.
// - Parameter "msgId": message ID to be acknowledged.
// - Parameter "status": status of the message.
void setMessageDeliveryStatus(String contact, String msgId, String status);

// Add message delivery listener:
// - Parameter "listener": Listener.
void addMessageDeliveryListener(IMessageDeliveryListener listener);

// Remove message delivery listener:
// - Parameter "listener": Listener.
void removeMessageDeliveryListener(IMessageDeliveryListener listener);


// Chat session:
interface IChatSession {
    // Get session ID
    String getSessionID();

    // Get remote contact
    String getRemoteContact();

    // Get session state
    int getSessionState();

    // Is chat group
    boolean isChatGroup();

    // Is Store & Forward session
    boolean isStoreAndForward();

    // Get first message exchanged during the session
    String getFirstMessage();

    // Accept the session invitation
    void acceptSession();

    // Reject the session invitation
    void rejectSession();

    // Cancel the session
    void cancelSession();

    // Get list of participants in the session
    List<String> getParticipants();

    // Add a participant to the session
    void addParticipant(String participant);

    // Add a list of participants to the session
    void addParticipants(List<String> participants);

    // Send a text message
    String sendMessage(String text);

    // Set the is composing status
    void setIsComposingStatus(boolean status);

    // Set message delivery status
    void setMessageDeliveryStatus(String msgId, String status);

    // Add session listener
    void addSessionListener(IChatEventListener listener);
```

```
    // Remove session listener
    void removeSessionListener(IChatEventListener listener);
}

// Chat event listener:
interface IChatEventListener {
    // Session is started
    void handleSessionStarted();

    // Session has been aborted
    void handleSessionAborted();

    // Session has been terminated by remote
    void handleSessionTerminatedByRemote();

    // New text message received
    void handleReceiveMessage(InstantMessage msg);

    // Chat error
    void handleImError(int error);

    // Conference event
    void handleConferenceEvent(String contact, String state);

    // Message delivery status
    void handleMessageDeliveryStatus(String msgId, String status);

    // Request to add participant is successful
    void handleAddParticipantSuccessful();

    // Request to add participant has failed
    void handleAddParticipantFailed(String reason);

    // Is composing event
    void handleIsComposingEvent(String contact, boolean status);
}

// Instant message object:
class InstantMessage {
    // Remote user
    String remote;

    // Text message
    String message;

    // Date of message
    Date date;

    // Message Id
    String msgId;
}

// Message delivery listener
interface IMessageDeliveryListener {
    // Message delivery status
    void handleMessageDeliveryStatus(String contact, String msgId, String
    status);
}

// Session state
interface SessionState {
    // Session state is unknown (i.e. session dialog path does not exist)
```

```
    int UNKNOWN;

    // Session has been cancelled (i.e. SIP CANCEL exchanged)
    int CANCELLED;

    // Session has been established (i.e. 200 OK/ACK exchanged)
    int ESTABLISHED ;

    // Session has been terminated (i.e. SIP BYE exchanged)
    int TERMINATED ;

    // Session is pending (not yet accepted by a final response by the remote)
    int PENDING ;
}
```

9.3. Intents

```
// Intent broadcasted when a new file transfer invitation has been received:
// - Parameter "contact": remote contact.
// - Parameter "contactDisplayname": display name of remote contact.
// - Parameter "sessionId": session ID of the incoming session.
// - Parameter "chatSessionId": session ID of a chat session if it exists.
// - Parameter "filename": file name to be transferred.
// - Parameter "filesize": file size to be transferred.
// - Parameter "filetype": MIME-type of the file to be transferred.
"com.orangelabs.rcs.messaging.FILE_TRANSFER_INVITATION"

// Intent broadcasted when a new chat invitation has been received:
// - Parameter "contact": remote contact.
// - Parameter "contactDisplayname": display name of remote contact.
// - Parameter "firstMessage": first message exchanged during the session.
// - Parameter "sessionId": session ID of the incoming session.
// - Parameter "isChatGroup": type of chat session.
// - Parameter "replacedSessionId": session ID of the extended 1-1 session.
"com.orangelabs.rcs.messaging.CHAT_INVITATION"

// Intent broadcasted when a 1-1 chat session has been extended to a group chat:
// - Parameter "sessionId": session ID of the new session.
// - Parameter "replacedSessionId": session ID of the extended 1-1 session.
"com.orangelabs.rcs.messaging.CHAT_SESSION_REPLACED"
```

10. SIP API

10.1. 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.

10.2. API

```
// Connect API:
void connectApi();

// Disconnect API:
void disconnectApi();

// Initiate a session with a contact:
// - Parameter "contact": remote contact.
// - Parameter "featureTag": service identifier.
// - Parameter "sdp": SDP part.
// - Returns a SIP session.
ISipSession initiateSession (String contact, String featureTag, String sdp);

// Get the session from its session ID:
// - Parameter "id": session ID.
// - Returns a SIP session or null if not found.
ISipSession getSession(String id);

// Get list of sessions with a contact:
// - Parameter "contact": remote contact.
// - Returns a SIP session or null if not found.
List<IBinder> getSessionWith(String contact);

// Get list of current established sessions:
// - Returns a list of sessions.
List<IBinder> getSipSessions();

// SIP session:
interface ISipSession {
```

```
    // Get session ID
    String getSessionID();

    // Get remote contact
    String getRemoteContact();

    // Get session state
    int getSessionState();

    // Get feature tag
    String getFeatureTag();

    // Get local SDP
    String getLocalSdp();

    // Get remote SDP
    String getRemoteSdp();

    // Accept the session invitation
    void acceptSession(String sdp);

    // Reject the session invitation
    void rejectSession();

    // Cancel the session
    void cancelSession();

    // Add session listener
    void addSessionListener(ISipSessionEventListener listener);

    // Remove session listener
    void removeSessionListener(ISipSessionEventListener listener);
}

// SIP session event listener:
interface ISipSessionEventListener {
    // Session is started
    void handleSessionStarted();

    // Session has been aborted
    void handleSessionAborted();

    // Session has been terminated by remote
    void handleSessionTerminatedByRemote();

    // SIP session error
    void handleSipSessionError(int error);
}
```

10.3. Intents

```
// Intent broadcasted when a new invitation has been received:
// - Parameter "contact": remote contact.
// - Parameter "featureTag": service identifier.
// - Parameter "sessionId": session ID of the incoming session.
"com.orangelabs.rcs.sip.SESSION_INVITATION"
```

11. Event log API

11.1. 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 (Call, SMS, and MMS).

See class:

- `EventsLogApi`

11.2. API

```
// Clear history associated to a contact contact:
// - Parameter "contact": selected contact.
void clearHistoryForContact(String contact);

// Delete a given log entry:
// - Parameter "id": ID of the entry to be deleted.
void deleteLogEntry(long id);

// Delete a SMS entry:
// - Parameter "id": ID of the entry to be deleted.
void deleteSmsEntry(long id);

// Delete a MMS entry:
// - Parameter "id": ID of the entry to be deleted.
void deleteMmsEntry(long id);

// Delete a call entry:
// - Parameter "id": ID of the entry to be deleted.
void deleteCallEntry(long id);

// Delete rich call history associated to a contact:
// - Parameter "contact": selected contact.
void deleteRichCallEntry(String contact);

// Delete an IM entry:
// - Parameter "id": ID of the entry to be deleted.
void deleteImEntry(long id);

// Delete chat and file transfer history associated to a contact:
// - Parameter "contact": selected contact.
void deleteMessagingLogForContact(String contact);

// Delete an IM session:
// - Parameter "sessionId": session ID to be deleted.
void deleteImSessionEntry(String sessionId);

// Get a cursor on the given chat session:
// - Parameter "sessionId": session ID.
// - Returns a database cursor.
Cursor getChatSessionCursor(String sessionId);

// Get cursor on the given chat contact:
```

```
// - Parameter "contact": selected contact.
// - Returns a database cursor.
Cursor getChatContactCursor(String contact);

// Get the events log URI:
// - Parameter "mode": filter mode (Call, Chat, FT, Richcall).
// - Returns content provider URI.
Uri getEventLogContentProviderUri(int mode);

// Get one to one chat history URI:
// - Returns content provider URI.
Uri getOneToOneChatLogContentProviderUri();

// Get group chat history URI:
// - Returns content provider URI.
Uri getGroupChatLogContentProviderUri();

// Get spam box history URI:
// - Returns content provider URI.
Uri getSpamBoxLogContentProviderUri();

// Mark a message as a spam:
// - Parameter "msgId": ID of the message.
// - Parameter "isSpam": whether to mark it as a spam or not.
void markChatMessageAsSpam(String msgId, boolean isSpam);

// Delete all spams
void deleteAllSpams();

// Mark a message as read:
// - Parameter "msgId": ID of the message.
// - Parameter "isRead": whether to mark it as read or unread.
void markChatMessageAsRead(String msgId, boolean isRead);

// Get number of unread messages for a given chat session:
// - Parameter "sessionId": session ID.
// - Returns the number of unread messages.
int getNumberOfUnreadChatMessages(String sessionId);

// Get the last message of a given chat session:
// - Parameter "sessionId": session ID.
// - Returns the last message of the session.
InstantMessage getLastChatMessage(String sessionId);
```

12.Media API

12.1. 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 pre-recorder video player.
- 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`
- `PrerecordedVideoPlayer`
- `VideoPlayerEventListener`

12.2. API

```
// Media RTP player:
interface IMediaPlayer {
    // Open the player
    void open(String remoteHost, int remotePort);

    // Close the player
    void close();

    // Start the player
    void start();

    // Stop the player
    void stop();

    // Returns the local RTP port
    int getLocalRtpPort();

    // Add a media listener
    void addListener(IMediaEventListener listener);

    // Remove media listeners
    void removeAllListeners();
```

```
    // Get supported media codecs
    MediaCodec[] getSupportedMediaCodecs();

    // Get media Codec
    MediaCodec getMediaCodec();

    // Set media codec
    void setMediaCodec(MediaCodec mediaCodec);
}
```

// Media RTP renderer:

```
interface IMediaRenderer {
    // Open the renderer
    void open(String remoteHost, int remotePort);

    // Close the renderer
    void close();

    // Start the renderer
    void start();

    // Stop the renderer
    void stop();

    // Returns the local RTP port
    int getLocalRtpPort();

    // Add a media listener
    void addListener(IMediaEventListener listener);

    // Remove media listeners
    void removeAllListeners();

    // Get supported media codecs
    MediaCodec[] getSupportedMediaCodecs();

    // Get media Codec
    MediaCodec getMediaCodec();

    // Set media codec
    void setMediaCodec(MediaCodec mediaCodec);
}
```

// Media event listener:

```
interface IMediaEventListener {
    // Media is opened
    void mediaOpened();

    // Media is closed
    void mediaClosed();

    // Media is started
    void mediaStarted();

    // Media is stopped
    void mediaStopped();

    // Media has failed
    void mediaError(String error);
}
```

// Media codec:

```
class MediaCodec {
    // Codec name
    String codecName;

    // Codec parameters
    Hashtable<String, String> parameters;
}
```

13.RCS settings

13.1. 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`

13.2. Parameters

UI settings:

- Service activation parameter which indicates if the RCS service may be started or not
- Roaming authorization parameter which indicates if the RCS service may be used or not in roaming
- Ringtone which is played when a social presence sharing invitation is received
- Vibrate or not when a social presence sharing invitation is received
- Ringtone which is played when a content sharing invitation is received
- Vibrate or not when a content sharing invitation is received
- Make a beep or not when content sharing is available during a call
- Video format for video share
- Video size for video share
- Ringtone which is played when a file transfer invitation is received
- Vibrate or not when a file transfer invitation is received
- Ringtone which is played when a chat invitation is received
- Vibrate or not when a chat invitation is received
- Auto-accept mode for chat invitation
- Predefined freetexts

Service settings:

- Max photo-icon size
- Max length of the freetext
- Max number of participants in a group chat
- Max length of a chat message
- Idle duration of a chat session
- Max size of a file transfer
- Warning threshold for file transfer size
- Max size of an image share
- Max duration of a video share
- Max number of simultaneous chat sessions
- Max number of simultaneous file transfer sessions
- Activate or not SMS fallback service
- Display a warning if Store & Forward service is activated
- IM start session mode
- Max entries for chat history.
- Max entries for richcall history.

User profile settings:

- IMS username or username part of the IMPU (for HTTP Digest only)
- IMS display name
- IMS private URI or IMPI (for HTTP Digest only)
- IMS password (for HTTP Digest only)

- IMS home domain (for HTTP Digest only)
- P-CSCF or outbound proxy address & port for mobile access
- P-CSCF or outbound proxy address & port for Wi-Fi access
- XDM server address & port
- XDM server login (for HTTP Digest only)
- XDM server password (for HTTP Digest only)
- IM conference URI for group chat session
- Country code

Stack settings:

- Registration retry base time
- Registration retry max time
- Polling period used before each IMS service check (e.g. test subscription state for presence service)
- Default SIP port for Mobile access
- Default SIP port for Wi-Fi access
- Default SIP protocol
- SIP transaction timeout used to wait a SIP response
- Default TCP port for MSRP session
- Default UDP port for RTP session
- MSRP transaction timeout used to wait MSRP response
- Registration expire period
- Publish expire period
- Revoke timeout
- IMS authentication procedure for mobile access
- IMS authentication procedure for Wi-Fi access
- Activate or not Tel-URI format
- Ringing session period. At the end of the period the session is cancelled
- Subscribe expiration timeout
- "Is-composing" timeout for chat service
- SIP session refresh expire period
- Activate or not permanent state mode
- Activate or not the logger
- Logger trace level
- Activate or not the SIP trace
- SIP trace filepath
- Activate or not the media trace
- Capability refresh timeout used to avoid too many requests in a short time
- Capability refresh timeout used to decide when to refresh contact capabilities
- Polling period used to decide when to refresh contacts capabilities
- CS video capability
- Image sharing capability
- Video sharing capability
- Instant Messaging session capability
- File transfer capability
- Presence discovery capability
- Social presence capability
- RCS extensions capability
- Instant messaging is always on (Store & Forward server)
- Instant messaging use report
- Network access
- SIP stack timer T1
- SIP stack timer T2
- SIP stack timer T4
- Enable SIP keep-alive
- SIP keep-alive period
- RCS APN
- SIM operator
- GRUU support
- Always-on CPU flag

13.3. API

There is one “get” method per parameter.
There is one “set” method per parameter which may be updated from UI application (e.g. ringtone).

14.RCS extensions

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

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 exchanged with other RCS contacts via SIP OPTIONS.

An application which wants to be an RCS extension detected by the RCS-e stack has just to add a specific Intent and MIME type in its Android manifest file. See the following syntax :

```
<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.orange.xxx"/>
</intent-filter>
```

where « xxx » is an id associated to the application implementing the RCS extension. See the following samples :

For a game :

```
<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.orange.game"/>
</intent-filter>
```

For a whiteboard service :

```
<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.orange.whiteboard"/>
</intent-filter>
```