Rich Communication Suite RCS API Detailed Requirements

Version 5.0

16 October 2019

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

Table of Contents

1 Introduction

1.1 Overview

The GSMA Rich Communication Suite (RCS) initiatives main objective is to bring a suite of services (using enablers from Open Mobile Alliance [OMA] and other Standards Development Organisations) to market.

RCS is entering a new phase in its evolution; the introduction of Application Programming Interfaces (APIs) to bring RCS to the market has been identified in GSMA RCS as a key priority.

GSMA RCS is looking for defined APIs to reference, which includes exposing of RCS capabilities to Web and Internet based developers, offering a set of commonly supported, lightweight, Web-friendly APIs to allow mobile operators and other Service Providers to expose useful information and capabilities to application developers. It aims to reduce the effort and time needed to create applications and content that is portable across Service Providers.

This document details the functional requirements for the RCS APIs.

Each individual deployment can consist of all the APIs or a subset of them (i.e., each individual API is optional).

The requirements realisation is a subset of the latest OMA technical specifications for:

- REST_NetAPI_FileTransfer
- REST_NetAPI_NotificationChannel
- REST_NetAPI_Chat
- REST_NetAPI_ThirdPartyCall
- REST_NetAPI_CallNotification
- REST_NetAPI_ImageShare
- REST_NetAPI_VideoShare
- REST_NetAPI_ACR
- REST_NetAPI_CapabilityDiscovery
- REST_NetAPI_TerminalLocation
- REST_NetAPI_AddressBook
- REST_NetAPI_Presence
- REST_NetAPI_Messaging
- REST_NetAPI_Common
- Autho4API (mandatory OMA supporting enabler for enabling delegated authorisation)
- REST_NetAPI_WebRTCSignaling
- REST_NetAPI_NMS

1.1.1 Differences with previous version of this specification

This version of the specification introduces the following changes:

- Service Provider / Chatbot Platform (SPCP) API requirements are added as a new subject (section 5)
- Necessary changes related to the new section are reflected into the scope (section 1.2), architecture diagram (section 1.3), definition of terms table (section 1.4) and document cross-reference table (section 1.5)
- “UNI API” added to the section 2 and section 3 titles for clarification

1.2 Scope

GSMA RCS has divided the APIs into three categories based on the target application developers, business model and location of the APIs. This classification is not completely precise but has been very instrumental in the discussions:

1. Device APIs
2. Wholesale/Business-to-Business (B2B) APIs, including Service Provider / Chatbot Platform (SPCP) APIs
3. UNI/Long Tail APIs

The first category (Device APIs) characterizes APIs residing in a device meant for an application executing in that very same device. The other two latter categories access the service through an interface within the network and where the service could be executing in many different locations including the end-user devices.

When it comes to the second category, these APIs are more in line with the traditional approach taken by the industry. It is possible that many B2B scenarios are covered by current requirements, with appropriate policy and security mechanisms. Section 5 contains requirements for SPCP APIs.

The intention with the UNI/Long Tail API is to put the threshold at the lowest possible level:

1. for “anyone” or any application developer to develop a service/application that embeds one or several RCS enablers;
2. allowing the embedding of RCS enablers in very lightweight environments (such as pure web browser applications).

In this document the term RCS APIs in section 1.3 refers to both B2B and UNI/Long Tail APIs accessed through the network, whereas in sections 2, 3 and 4 the term RCS APIs only refers to UNI/Long Tail APIs.

1.3 Architecture

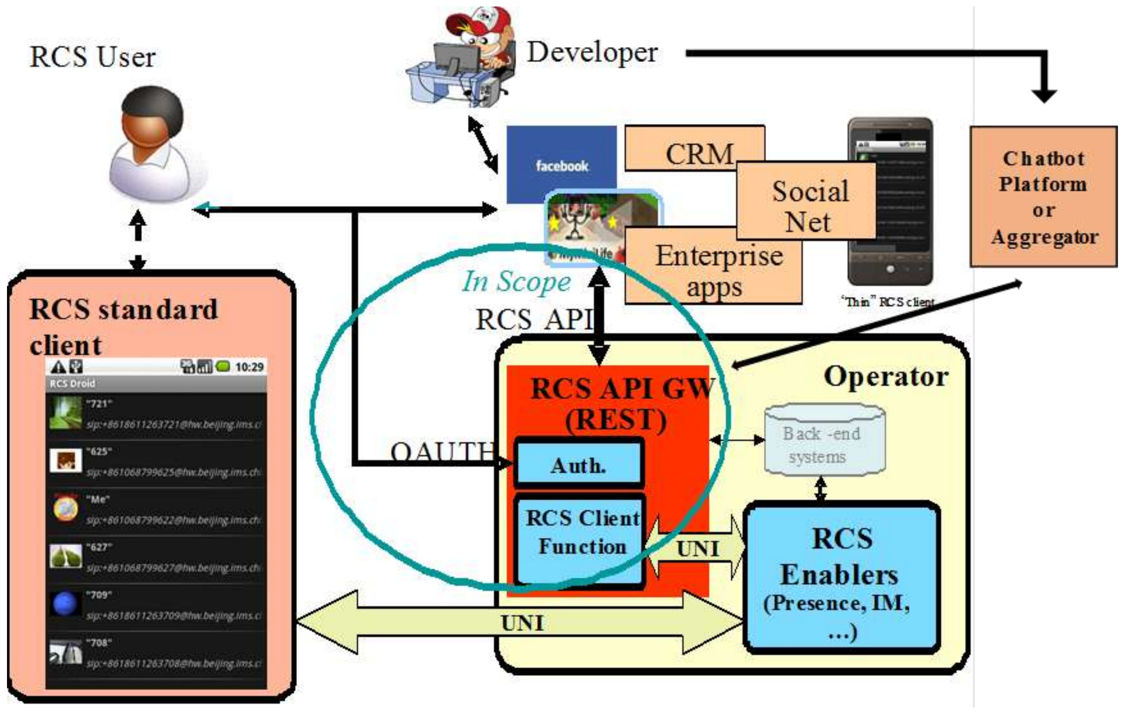


Figure 1: RCS API Architecture

The figure "RCS API Architecture" shows a sample RCS API Architecture supporting:

1. Application authorisation to access the RCS methods/functions on behalf of the RCS user.
2. End-user management of applications user has granted access to, which resource that is granted, and the possibility to revoke the access for a given application.
3. Operation of the RCS user's services via the existing RCS UNI using the defined API primitives.
4. Developer security mechanisms and engagement/registration processes aimed to individual or SME developers (out of scope of this document). Mechanisms and policies shall be defined by the Service Provider. In many cases the existing developer portals and communities could accommodate RCS.
5. Application and user authentication (out of scope of this document). In an RCS deployment, authentication mechanisms will be defined by the Service Provider, and they could reuse the same authentication used for "regular" clients.

1.4 Definition of Terms

Term	Description
ACR	Anonymous Customer Reference
API	Application Programming Interface
CPM	Converged IP Messaging
IP	Internet Protocol

IS	Image Share
MMS	Multimedia Messaging Service
MSRP	Message Session Relay Protocol
NAB	Network Address Book
NNI	Network-to-Network Interface
OMA	Open Mobile Alliance
PNB	Personal Network Blacklist
RCS	Rich Communication Suite
REST	Representational State Transfer
SME	Small and Medium Enterprises
SMS	Short Message Service
SPCP	Service Provider / Chatbot Platform
UNI	User-to-Network Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VoIP	Voice over IP
WebRTC	Web Real-Time Communication

1.5 Document Cross-References

Ref	Title
[RFC6202]	Known issues and best practices for the Use of Long Polling and Streaming in Bidirectional HTTP http://tools.ietf.org/html/rfc6202
[RFC6455]	The WebSocket Protocol http://tools.ietf.org/html/rfc6455
[OAUTH20]	The OAuth 2.0 Protocol Framework http://tools.ietf.org/html/rfc6749
[RCS5.3]	GSMA PRD RCC.07 RCS 5.3 Advanced Communications Services and Client Specification http://www.gsma.com/rcs/specifications
[RCC07]	GSMA PRD RCC.07 RCS - Advanced Communications Services and Client Specification http://www.gsma.com/rcs/specifications
[RCSR5OMAIMEND]	GSMA PRD RCC.12 RCS Endorsement of OMA SIP Simple IM http://www.gsma.com/rcs/specifications
[IR74]	GSMA IR.74 - Video Share Interoperability Specification http://www.gsma.com/newsroom/technical-documents/technical-documents/
[IR79]	GSMA [IR79] Image Share Interoperability Specification http://www.gsma.com/newsroom/technical-documents/technical-documents/

[IR84]	GSMA IR.84 - Video Share Phase 2 Interoperability Specification http://www.gsma.com/newsroom/technical-documents/technical-documents/
[IR58]	GSMA IR.58 – IMS Profile for Voice over HSPA http://www.gsma.com/newsroom/technical-documents/technical-documents/
[IR92]	GSMA IR.92 – IMS Profile for Voice and SMS http://www.gsma.com/newsroom/technical-documents/technical-documents/
[IR94]	GSMA IR.94 – IMS Profile for Conversational Video Service http://www.gsma.com/newsroom/technical-documents/technical-documents/
[Autho4API_10]	"Authorization Framework for Network APIs", Open Mobile Alliance™, OMA-ER-Autho4API-V1_0 http://www.openmobilealliance.org/
[REST_NetAPI_3PC]	"RESTful Network API for Third Party Call", Open Mobile Alliance™, OMA-TS-REST_NetAPI_ThirdPartyCall-V1_0 http://www.openmobilealliance.org/
[REST_NetAPI_AddressBook]	"RESTful Network API for Address Book", Open Mobile Alliance™, OMA-TS-REST_NetAPI_AddressBook-V1_0 http://www.openmobilealliance.org/
[REST_NetAPI_ACR]	RESTful Network API for Anonymous Customer Reference Management", Open Mobile Alliance™, OMA-TS-REST_NetAPI_ACR-V1_0 http://www.openmobilealliance.org/
[REST_NetAPI_CallNotif]	"RESTful Network API for Call Notification", Open Mobile Alliance™, OMA-TS-REST_NetAPI_CallNotification-V1_0 http://www.openmobilealliance.org/
[REST_NetAPI_ CapabilityDiscovery]	"RESTful Network API for Capability Discovery", Version 1.0, Open Mobile Alliance™, OMA-TS-REST_NetAPI_ CapabilityDiscovery -V1_0 http://www.openmobilealliance.org/
[REST_NetAPI_Chat]	"RESTful Network API for Chat", Open Mobile Alliance™, OMA-TS-REST_NetAPI_Chat-V1_0 http://www.openmobilealliance.org/
[REST_TS_Common]	"Common definitions for RESTful Network APIs", Open Mobile Alliance™, OMA-TS-REST_NetAPI_Common-V1_0 http://www.openmobilealliance.org/
[REST_NetAPI_FileTransfer]	"RESTful Network API for File Transfer", Open Mobile Alliance™, OMA-TS-REST_NetAPI_FileTransfer-V1_0 http://www.openmobilealliance.org/
[REST_NetAPI_ImageShare]	"RESTful Network API for Image Share", Open Mobile Alliance™, OMA-TS-REST_NetAPI_ImageShare-V1_0 http://www.openmobilealliance.org/

[REST_NetAPI_NotifChnl]	"RESTful Network API for Notification Channel", Version 1.0, Open Mobile Alliance™, OMA-TS-REST_NetAPI_NotificationChannel-V1_0 http://www.openmobilealliance.org/
[REST_NetAPI_VideoShare]	"RESTful Network API for Video Share", Version 1.0, Open Mobile Alliance™, OMA-TS-REST_NetAPI_VideoShare-V1_0 http://www.openmobilealliance.org/
[REST_NetAPI_Location]	"RESTful Network API for Terminal Location", Version 1.0, Open Mobile Alliance™, OMA-TS-REST_NetAPI_TerminalLocation-V1_0 http://www.openmobilealliance.org/
[REST_NetAPI_Messaging]	"RESTful Network API for Messaging", Version 1.0, Open Mobile Alliance™, OMA-TS-REST_NetAPI_Messaging-V1_0 http://www.openmobilealliance.org/
[REST_NetAPI_Presence]	"RESTful Network API for Presence", Version 1.0, Open Mobile Alliance™, OMA-TS-REST_NetAPI_Presence-V1_0 http://www.openmobilealliance.org/
[OMACPM-MS]	CPM Message Storage, Version 2.0, Open Mobile Alliance™, OMA-TS-CPM_MessageStorage-V2_0-20150113-C http://www.openmobilealliance.org/
[REST_NetAPI_WRTCS]	RESTful Network API for WebRTC Signaling 1.0, Open Mobile Alliance™, OMA-TS-REST_NetAPI_WebRTCSignaling-V1_0 http://www.openmobilealliance.org/
[REST_NetAPI_NMS]	RESTful Network API for Network Message Storage 1.0, Open Mobile Alliance™, OMA-TS-REST_NetAPI_NMS-V1_0 http://www.openmobilealliance.org/
[W3C_WebRTC]	WebRTC 1.0: Real-time Communication Between Browsers, W3C http://www.w3.org/TR/webrtc/
[RFC3264]	J. Rosenberg and H. Schulzrinne, RFC3264: An Offer/Answer Model with the Session Description Protocol, June 2002 http://www.ietf.org/rfc/rfc3264.txt
[IETF-DRAFT-JSEP]	Javascript Session Establishment Protocol, Version 8, October 27, 2014 https://tools.ietf.org/html/draft-ietf-rtcweb-jsep-08

2 RCS high-level requirements for UNI API

Label	Description	Comment
UNI-HLF-001	The RCS API SHALL be HTTP/REST based.	
UNI-HLF-002	Resource URLs and primitive names SHALL have an intuitive relationship with the functions and resources they are intended to represent.	
UNI-HLF-003	It SHOULD be possible to reuse the Data definitions of the RCS APIs for future	

	bindings.	
UNI-HLF-004	The RCS APIs SHALL allow including the API version in the resource URLs	
UNI-HLF-004b	The RCS APIs SHALL allow an application and a server to negotiate the version of a particular resource	This requirement might use the API version in the URL or not.
UNI-HLF-005	The RCS API SHALL expose a functional abstraction at the user level rather than at the level of underlying protocols.	
UNI-HLF-006	The RCS API SHALL support ”server”-based application clients and ”device”-based application clients. Instantiation examples include applications running on a Web server (where the user interacts with the application via a web browser), or running on a mobile or fixed device as a ”widget” or as a native application.	
UNI-HLF-007	The RCS APIs SHALL support application authorisation based on OAuth2.0.	Cf. requirement [UNI-OAU-001] Ref: [OAUTH2.0] Users are expected to be authenticated by their Service Providers, however the authentication mechanisms for the user and application are out of scope of this document and are therefore out of scope for RCS APIs.
UNI-HLF-008	Subject to the underlying resource capabilities, the RCS APIs SHALL NOT expose the real identities of the user and her/his contacts. In particular, mobile telephone numbers (i.e., MSISDNs) or identities SHALL NOT be exposed either for users or for their contacts. Subject to Service Provider policies, only trusted applications will be authorized to know that information.	
UNI-HLF-009	The RCS APIs SHALL be restricted to the operations and procedures of the RCS UNI as defined by GSMA RCS.	Applications using the RCS APIs should not be able to perform operations not possible to a regular RCS client. Ref: [RCS5.3] Call UNI API Requirements (see section 4.10) exposes RCS UNI for IP Voice and Video Call functionality. Ref: [RCS5.3] ch 3.8 IP Voice Call (IR.92 and IR.58), ch 3.9 IP Video

		Call (IR.94).
UNI-HLF-010	RCS APIs shall be extensible in a backward compatible way	

Informative note: It is expected to be possible for a Service Provider to deploy developer security mechanisms and engagement/registration processes aimed at individual developers. Developer security mechanisms are out of the scope of this document, and therefore out-of-scope for RCS APIs.

3 Authorisation framework for UNI API

Note: Authentication (of user, application, or developer) is out of the scope of this document, because in an RCS deployment authentication mechanisms will be defined by the Service Provider, typically re-using the authentication used for "regular" RCS clients. Application authorisation is under scope as per OAuth flow (see UNI-OAU-002 and ff).

Note: In the context of this section, "widget" should be understood in a general way as to denote a range of device software ranging from web applets to small non-native applications.

3.1 General requirements

Label	Description	Comment
UNI-AUT-001	The Authorisation framework SHALL enable a user owning network resources exposed by a RESTful API to authorize third-party applications to access these resources via this RESTful API on that user's behalf.	
UNI-AUT-002	The Authorisation framework SHALL support network-side Web applications, accessed from the user's Web browser.	
UNI-AUT-003	The Authorisation framework SHOULD support client-side stand-alone widget applications installed on the user's terminal and running outside of a Web browser.	
UNI-AUT-004	The Authorisation framework SHOULD support client-side native code applications installed on the user's terminal.	
UNI-AUT-005	The Authorisation framework SHALL NOT require a user to reveal to third-party applications the credentials he/she uses to authenticate to the Service Provider.	Note: This is an RCS user privacy requirement.
UNI-AUT-006	The Authorisation framework SHALL allow a third-party application to obtain from a Service Provider (e.g., by provisioning or dynamic discovery) the parameters required to request a user's authorisation and to access the user's network resources.	
UNI-AUT-007	The Authorisation framework SHALL support a third-party application to initiate the	

	authorisation request by directing the user to the Service Provider's portal.	
UNI-AUT-008	The Authorisation framework SHALL support presenting the third-party application's authorisation request to the resource owner in the form of an explicit authorisation dialog or a user consent request.	It is assumed that the user has authenticated to the Service Provider before granting authorisation (user authentication is out of scope of the Authorisation framework). Note: Design and handling of this dialog are out of scope for the RCS API. However, the API needs to communicate the parameters needed for the dialog, and/or specified by the user in the dialog
UNI-AUT-009	The Authorisation framework SHOULD facilitate presenting to the resource owner at least the third-party application identity, the resources and the operations on these resources for which authorisation is requested.	Note: Design and handling of the dialog presenting this are out of scope for the RCS API. However, the API needs to communicate the parameters needed for the dialog, and/or specified by the user in the dialog.
UNI-AUT-010	The Authorisation framework SHALL enable the resource owner to authorize or deny access to each of the requested resources and operations.	
UNI-AUT-011	The Authorisation framework MAY enable the resource owner to specify the duration for which his/her access authorisation is granted.	
UNI-AUT-012	The Authorisation framework SHOULD facilitate communicating the resource owner's preferred language and terminal capabilities.	
UNI-AUT-013	In case the user authorizes the third-party application to access the user's resources, the Authorisation framework SHALL be able to provide to the third-party application an access token representing this user's authorisation subject to obtaining it from the issuer.	
UNI-AUT-014	The access token SHALL be usable only by the third-party application for the restricted scope (operations on resources) authorized by the user at the time of authorisation request.	
UNI-AUT-015	~~VOID~~	~~VOID~~
UNI-AUT-016	The Authorisation framework SHALL support the inclusion of an access token (e.g.	

	obtained by the third-party application from the Service Provider for the scope of this request) in requests to resources exposed by the RESTful API.	
UNI-AUT-017	The Authorisation framework SHOULD facilitate the possibility to retrieve the list of the third-party applications that have been authorized before and which resources have been authorized per third-party application by the user.	
UNI-AUT-018	The Authorisation framework SHOULD facilitate the possibility for the user to remove the authorisation for any third-party application that has previously been authorized.	
UNI-AUT-019	Notifications sent to the third-party application SHALL be filtered based on authorisation granted to the third-party application. As such, the server SHALL NOT send notifications regarding a resource for which the application has no authorisation.	Cf. requirement [UNI-NTF-005]

For an informative example, see Annex A.

3.2 Authorisation using OAuth

Label	Description	Comment
UNI-OAU-001	The Authorisation framework SHALL be based on OAuth 2.0 as specified in [OAUTH20].	Cf. requirement [UNI-HLF-007] Ref: [OAUTH20]
UNI-OAU-002	The Authorisation framework SHALL support the OAuth 2.0 “Authorisation Code flow”, where the third-party application is a server-side web application.	
UNI-OAU-003	The Authorisation framework SHALL support OAuth 2.0, where the types of third-party applications can either be client-side installed widget applications or client-side native code applications.	
UNI-OAU-004	For the delivery of authorisation code (“Authorisation Code Flow”) / access token (“Implicit Grant Flow”) to a client-side installed application (widget or native code application), the Authorisation framework SHALL support at least one OS-agnostic and application-type agnostic delivery mechanism, which does not require end-user interaction such as manual input of authorisation code.	Annex 1 provides an informative example of such a mechanism, based on binary-SMS. An alternative option would be to use the notification channel as the delivery mechanism.

UNI-OAU-005	The Authorisation framework MAY support OAuth 2.0 flows other than the “Authorisation Code Flow”.	
UNI-OAU-006	The Authorisation framework SHALL support the OAuth 2.0 “Authorisation Server” and “Resource Server” roles.	
UNI-OAU-007	The Authorisation framework SHALL regard the user’s resources accessed via the RESTful API as the OAuth 2.0 “Protected Resource”.	
UNI-OAU-008	When following the Authorisation Code Flow the Authorisation framework SHALL generate an OAuth 2.0 authorisation code as a result of the user authorisation.	If other flows are used, a similar functionality should be provided.
UNI-OAU-009	The Authorisation framework SHALL support the exchange of an authorisation code for an access token according to OAuth 2.0.	
UNI-OAU-010	The Authorisation framework SHALL bind the authenticated user identity to the generated authorisation code and access token.	Note: The actual authentication mechanism used is out of the scope of this document because it is foreseen that in an RCS deployment authentication mechanisms will be defined by the Service Provider, typically re-using the authentication used for “regular” RCS clients.
UNI-OAU-011	The Authorisation framework SHALL be able to determine the user identity (e.g. MSISDN) from the access token received from the application.	
UNI-OAU-012	The Authorisation framework SHALL validate the access token received from the application according to OAuth 2.0.	
UNI-OAU-013	The values of the OAuth 2.0 “scope” parameter SHALL reflect selected granularity in the usage of RCS enablers/resources via the REST API.	
UNI-OAU-014	The values of OAuth 2.0 “scope” parameter SHALL have a direct mapping (1-to-1 or 1-to-many or many-to-many) to the available RCS APIs primitives.	
UNI-OAU-015	The following minimum set of "scope" values targeted granularity SHALL be supported: a. presence_publish_spi b. presence_publish_servicecapabilities c. presence_subscriptions d. chat e. filetransfer f. videoshare	API design should assign one of these scope values to each operation defined in the APIs. Note that the mandatory requirement applies only to the targeted granularity of

	g. imageshare h. voice_call i. multimedia_call j. call_notification k. pnb_management	the “scope values” and not necessarily to the listed identifiers themselves. The way the identifiers are specified is left to the technical specification.
UNI-OAU-016	In addition to the values defined in requirement [UNI-AUT-015], it SHOULD be possible to define per-Service Provider values of “scope” parameter to accommodate different granularity levels.	

Note: All figures are informative.

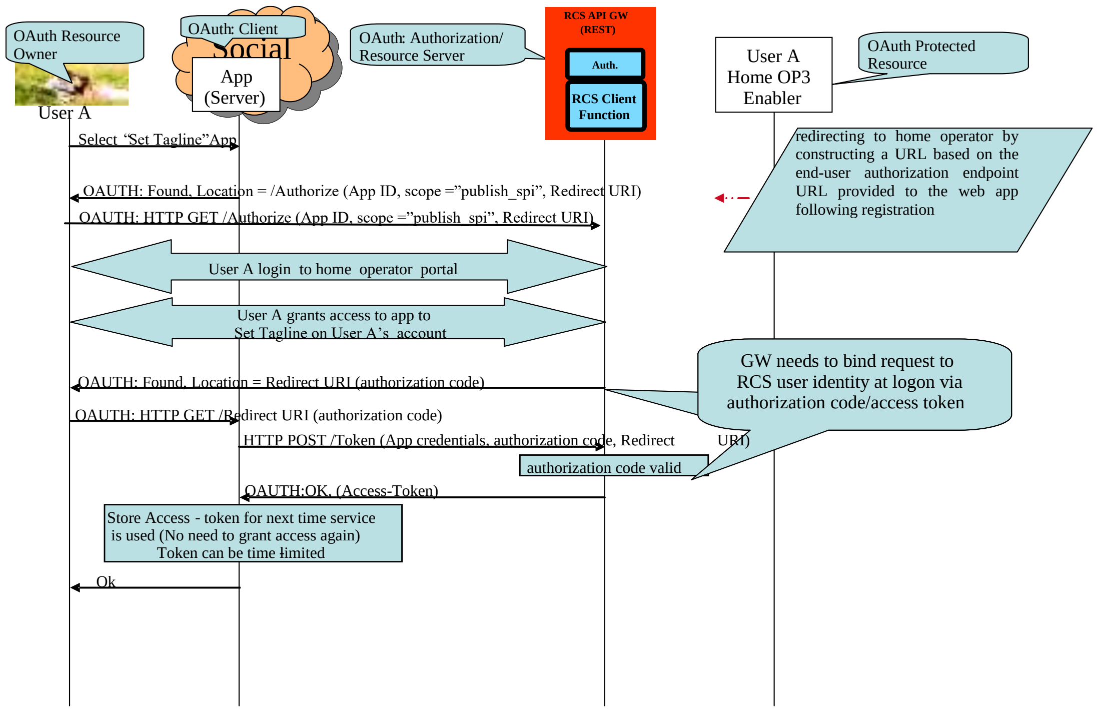

OAuth Resource Owner
OAuth: Client
Social
App (Server)
OAuth: Authorization/ Resource Server
RCS API GW (REST)
Auth.
RCS Client Function
User A Home OP3 Enabler
OAuth Protected Resource
User A
Select 'Set Tagline"App
redirecting to home operator by constructing a URL based on the end-user authorization endpoint URL provided to the web app following registration
OAUTH: Found, Location = /Authorize (App ID, scope ="publish_spi", Redirect URI)
OAUTH: HTTP GET /Authorize (App ID, scope ="publish_spi", Redirect URI)
User A login to home operator portal
User A grants access to app to Set Tagline on User A's account
GW needs to bind request to RCS user identity at logon via authorization code/access token
OAUTH: Found, Location = Redirect URI (authorization code)
OAUTH: HTTP GET /Redirect URI (authorization code)
HTTP POST /Token (App credentials, authorization code, Redirect URI)
authorization code valid
OAUTH:OK, (Access-Token)
Store Access - token for next time service is used (No need to grant access again) Token can be time limited
Ok

Figure 2: Example of Application Authorisation of OAuth 2.0 in RCS Using OAuth Authorisation Code Flow

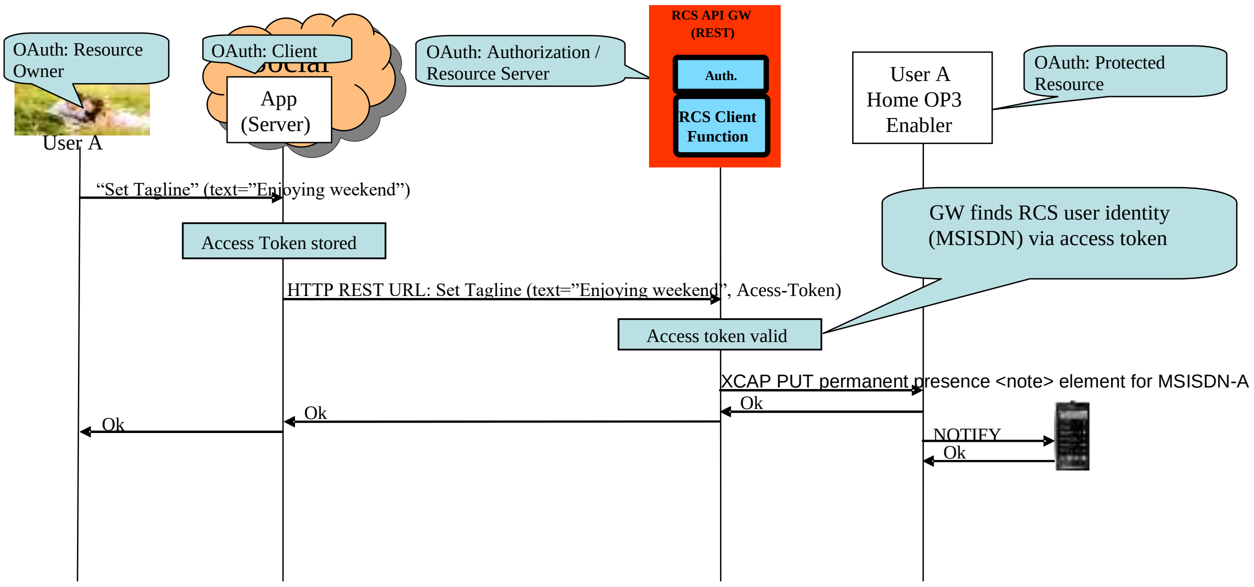

OAuth: Resource Owner
User A
OAuth: Client
Social
App (Server)
OAuth: Authorization / Resource Server
RCS API GW (REST)
Auth.
RCS Client Function
User A Home OP3 Enabler
OAuth: Protected Resource
"Set Tagline" (text="Enjoying weekend")
Access Token stored
GW finds RCS user identity (MSISDN) via access token
HTTP REST URL: Set Tagline (text="Enjoying weekend", Acess-Token)
Access token valid
XCAP PUT permanent presence <note> element for MSISDN-A
Ok
Ok
Ok
NOTIFY
Ok

Figure 3: Example of Application Usage of OAuth 2.0 in RCS

4 UNI API requirements

4.1 General requirements

4.1.1 Common notification channel

Label	Description	Comment
UNI-NTF-001	The RCS APIs SHALL support a common notification mechanism that allows delivery of notifications for multiple different subscriptions to the same endpoint at the application.	Different RCS services need to alert a user of events (incoming chat invite, presence update from buddy, etc.). If each RCS service has its own notification channel, a multi-service application would need to manage multiple such notification channels. This would result in increased complexity and would be impossible to manage in some environments (for example, web browsers have a limitation in the number of open HTTP connections). Similar requirements from disparate domains have driven the development of so called bidirectional HTTP technologies (Comet, Reverse AJAX, long polling), see [RFC6202].
UNI-NTF-002	The RCS APIs SHALL support the delivery of notifications directly to an application-defined	The application establishes a subscription to notifications by

	endpoint (i.e., a callback URL), using HTTP.	providing a call-back URL where the notifications are to be received. This method follows the well-known subscription/notification pattern using REST primitives. It is to be used mainly for server-to-server notifications. Emerging industry standards for such notifications like pubsubhubbub (http://code.google.com/p/pubsubhubbub/) could be taken into consideration.
UNI-NTF-003	The RCS APIs SHALL support the delivery of notifications to the application in an HTTP-based notification channel using the long-polling mechanism (see [RFC6202]).	This method is to be used mainly in environments that cannot receive requests from the network or cannot support server environments, such as browsers, devices, set top boxes, and so on. The application issues a “long” polling request to establish a notification channel for receiving notifications.
UNI-NTF-004	The notification mechanisms according to requirement [UNI-NTF-002] and [UNI-NTF-003] SHALL use the same data format and schemes for notifications.	
UNI-NTF-005	Notifications sent SHALL be filtered based on authorisation granted to the application, so the server SHALL NOT send notifications regarding a resource for which the application has no authorisation.	Cf. requirement [UNI-AUT-019]
UNI-NTF-006	The RCS APIs SHALL support selective subscriptions of the application to notifications about specific events.	As an example, an application that only reads / sets the free text field would not be interested in Video Share-related notifications, or contact list update notifications.
UNI-NTF-007	The RCS APIs SHALL be able to deliver multiple events in one single (long polling) notification.	This mechanism is to be used for long-polling but might be adopted in other cases (e.g., delivering notifications with a callback URL).
UNI-NTF-008	The RCS APIs SHALL support the inclusion of a reference to the relevant resource in the notification.	The application can use the received resource reference to perform relevant actions on the resource (e.g. accept invite or get presence data from buddies). Notification events are expected to be able to include details where applicable (e.g. session progress

		information such as “Chat answered”). Note: Some events will be self-contained, meaning they contain all information the application requires for further processing. Others notifications might require querying a resource, which requires the URL to be included in the event notification.
UNI-NTF-009	RCS APIs SHOULD include an informative description or reference model for the “long polling” notification channel.	Since there are no telco-related standards using these techniques, this would facilitate interworking and guide implementations, including aspects such as when connections should be closed, open or retried. Recommendations and best practices in [RFC6202] for “long polling” to be considered.
UNI-NTF-010	The RCS APIs SHOULD support the delivery of notifications to the application via a Websocket based notification channel (see [RFC6455]).	

4.1.2 Examples (informative)

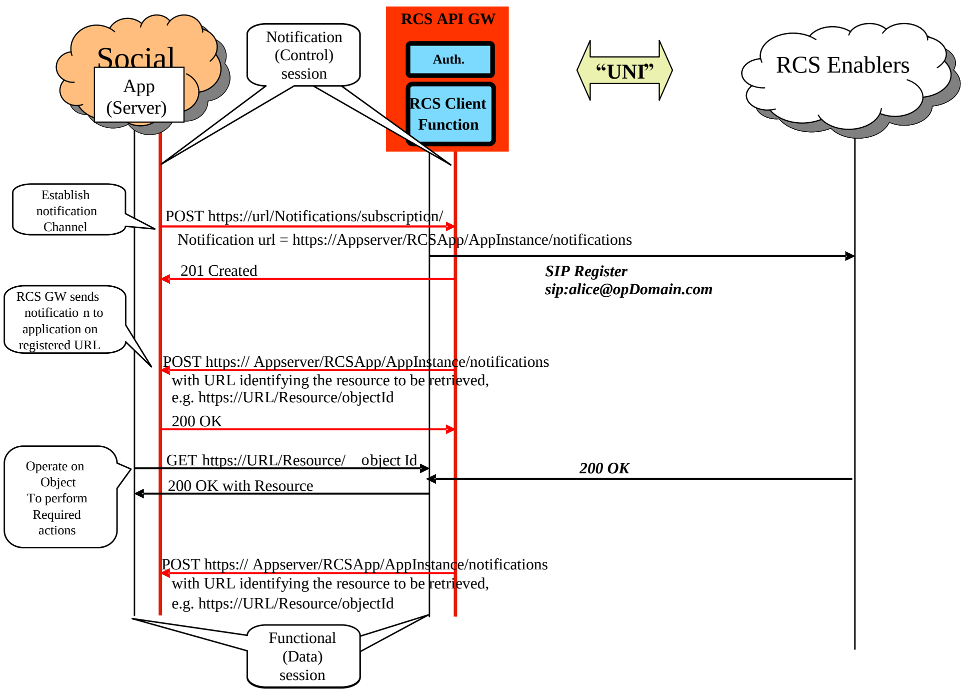

Social
App (Server)
Notification (Control) session
RCS API GW
Auth.
RCS Client Function
“UNI”
RCS Enablers
Establish notification Channel
POST https://url/Notifications/subscription/
Notification url = https://Appserver/RCSApp/AppInstance/notifications
201 Created
SIP Register sip:alice@opDomain.com
RCS GW sends notificatio n to application on registered URL
POST https:// Appserver/RCSApp/AppInstance/notifications with URL identifying the resource to be retrieved, e.g. https://URL/Resource/objectId
200 OK
Operate on Object To perform Required actions
GET https://URL/Resource/ object Id
200 OK with Resource
200 OK
POST https:// Appserver/RCSApp/AppInstance/notifications with URL identifying the resource to be retrieved, e.g. https://URL/Resource/objectId
Functional (Data) session

Figure 4: Notification Channel Using “subscription” Method, Example

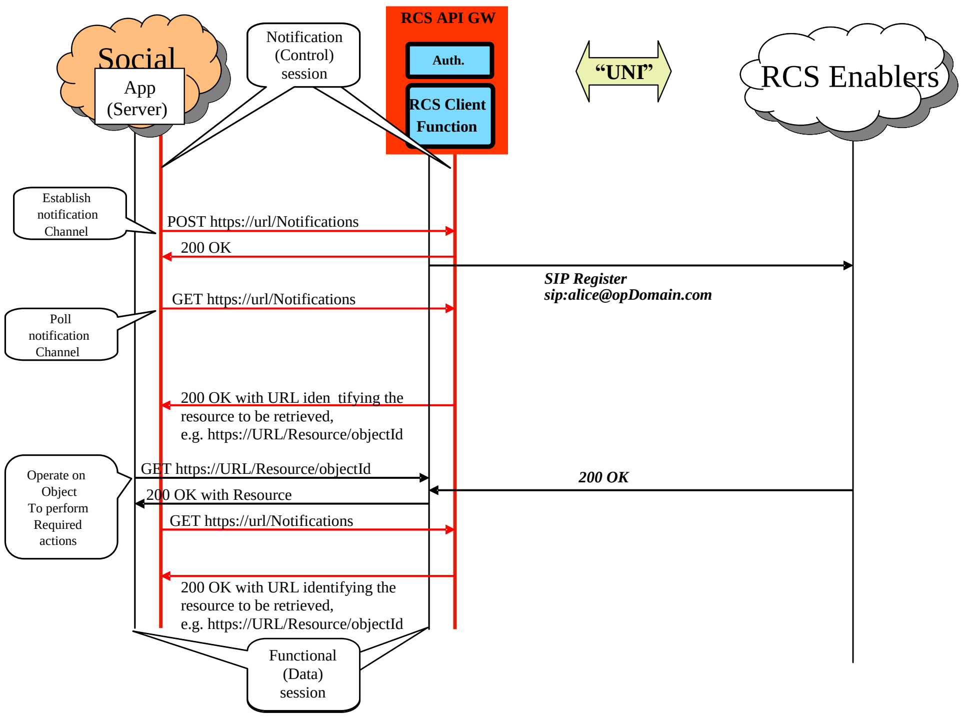

Social
App (Server)
Notification (Control) session
RCS API GW
Auth.
RCS Client Function
"UNI"
RCS Enablers
Establish notification Channel
POST https://url/Notifications
200 OK
SIP Register sip:alice@opDomain.com
GET https://url/Notifications
Poll notification Channel
200 OK with URL iden tifying the resource to be retrieved, e.g. https://URL/Resource/objectId
Operate on Object To perform Required actions
GET https://URL/Resource/objectId
200 OK
200 OK with Resource
GET https://url/Notifications
200 OK with URL identifying the resource to be retrieved, e.g. https://URL/Resource/objectId
Functional (Data) session

Figure 5: Notification Channel Using "long polling" Method, Example

NOTE: In the following sections, the parameters mentioned in the "Required parameters (not complete list)" column should not be construed to be complete and final; the intention is to include only the parameters required by the semantics of each operation. In particular, elements such as a "tag" (to identify and correlate operations and notifications), and so on, are not included as they are understood to be part of the technical design.

4.1.3 Service Tags

Label	Description	Required parameters	Comment
UNI-ServTags-001	The Messaging UNI API, Chat UNI API, File Transfer UNI API, Call Control and Notification UNI API, WebRTC Signaling UNI API, Video Share UNI API shall each allow: ▪ On sending, an IARI value identifying a third party application to be included in the API towards the IMS	oauth_token={access-token} Optional: IARI value Explicit_require tag set to yes or no (default is no)	A third party application can decide whether it requires that ▪ only traffic identified with the IARI shall be accepted between two or more applications using the same IARI (i.e., an application initiating an IARI requires the recipient application to also support the same IARI value) by setting explicit_require to yes, and

	framework below ▪ On receiving, an IARI value identifying the target third party application to be carried from the IMS framework towards the application level ▪ On sending, include an indication whether the recipient is required to be the application corresponding to the IARI value (i.e., the IARI value has to be registered by the receiving application via the Presence or Capability Discovery API)		▪ traffic identified with the IARI can be accepted between two or more applications where not all applications support the same IARI (i.e., an application initiating an IARI does not require the recipient application to also support the same IARI value). by setting explicit_require to no NOTE 1: in protocol terms mandating that the IARI is registered is accomplished by adding both the "explicit" and "require" parameters to the Accept-Contact header carrying the IARI of the corresponding SIP request as per RFC 3841. NOTE 2: It is out of the scope of each particular API what IARI values are allowed. NOTE 3: If the IARI feature is used, the same IARI value must be used in all messages belonging to the same session. See [RCS5.3] section 3.12.4 Extensions – Technical Realisation.
UNI-ServTags-002	The Presence UNI API, and Capability Discovery UNI API shall allow: • declaring the IARI capability • Retrieving IARI supported by remote	oauth_token={access-token}	

	contacts		

4.2 Anonymous Customer Reference (ACR) API Requirements

The API gateway providing the RCS APIs SHALL NOT expose the real identities of the user and their contacts (see UNI-HLF-008). This means that the API will need to use Anonymous Customer References (ACRs).

Nevertheless, some applications do hold the real identities of their users as they get contact data from other sources (e.g., terminal address books, direct user input, Service Provider address books). Therefore, a mechanism to translate real identities (e.g., MSISDNs) into ACRs is needed and shall be provided by gateway.

Label	Description	Required parameters	Comment

UNI-ACR-001	The ACR API SHALL support requesting an Anonymous Customer Reference (ACR) associated to an MSISDN.	oauth_token={access-token}
		msisdn: {msisdn}
		
		return value:
		acr;{Anonymous Customer Reference}
			The ACR needs to be stable for a given MSISDN and application ID if applicable. This means that the anonymized ID returned by the API shall not change over the time for a given MSISDN and application.
			
			For security and end user privacy reasons, it is recommended that the ACRs for a given MSISDN vary with the application ID. That is, it is recommended that two different applications get different anomymized IDs for the same MSISDN.
			
			For MSISDN, the tel: URI scheme [RFC3966] SHOULD be used in the interface for an MSISDN; and the acr: URI scheme as defined in Appendix H of [REST_NetAPI_ACR] SHOULD be for the Anonymous Customer Reference.

4.3 Network Address Book API requirements

This section has an informative character. It captures the discussion of the working group about contact data and Network Address Books (NABs).

Contact data is essential for RCS communication. An RCS application can get contact data from different sources:

1. Direct user input
2. Terminal address book
3. An RCS API provider's NAB
4. NAB of a Service Provider that does not offer the RCS API

The interfaces through which the address book is accessed by the application are implementation-specific. However, a MSISDN or an anonymized identifier is needed to link to an RCS user.

For RCS API Service Providers that also run a NAB as specified below, it is recommended that the NAB works with the ACRs as specified in this document.

4.3.1 General considerations (informative)

NAB API's main use case is to allow applications to fetch contact information and to receive updates regarding contact information (i.e., new contact added, contact information modified, etc). Additional operations are defined to allow applications to update the address book.

Depending on a Service Provider's policies, in general, retrieve operations return a list of contacts, but not the complete information for each one of the contacts. The contact identity returned is the one that should be used by the rest of APIs.

Two different identities can be returned:

1. a human readable identity that the application can show to the user; and
2. an identity for use by the rest of APIs (e.g. a tokenized identifier which is not intended to be human readable).

An ACR for a user/contact is usually assigned by the Service Provider and may be common for all applications that may subsequently use it or may be assigned per each application basis, subject to Service Provider's policies. How a given ACR is generated and how it populates the resource representing the contact in the NAB is out of scope for the NAB API.

Depending on a Service Provider's policies, trusted applications can get complete information (potentially including an MSISDN or URI). OAuth 2.0 mechanisms can be leveraged to that end.

Retrieve address book allows optionally filtering. Only contacts or fields matching specified conditions will be returned.

Note: It is recommended that filtering re-uses existing OMA filtering syntax as much as possible.

4.3.2 RCS NAB basic operations

Label	Description	Required parameters (not complete list)	Comment
UNI-NAB-001	The Network Address Book API SHALL support	oauth_token={access-token}	The answer amounts to retrieval of the list of contacts

	retrieving a filtered list of contacts in the NAB and its associated information subject to Service Provider policies.	Optional: filtering parameters	in the address book, possibly based on some filtering conditions. If filtering is requested then only matching contacts are returned. Subject to Service Provider policies, the retrieved list may not include the contact identity as underlying identifiers (i.e., MSISDN or URI) but instead may include the contact identity as tokenized strings that hide that information (ACRs). The contact identity (i.e., MSISDN, URI, or ACR) returned is the only one that can be used by the rest of APIs (e.g., chat, file transfer, etc.). The contact name, which is the display name, is envisaged as the way for a human user to identify the contacts and it cannot be used as the contact identity to be used by the rest of APIs (e.g. chat, file transfer, etc). The name of the REST resource representing the contact is envisaged as a mechanism to uniquely identify the resource in the context of the NAB API and it cannot be used by the rest of APIs (e.g., chat, file transfer, etc)..

UNI-NAB-002	The Network Address Book API SHALL support retrieving all information for a specified contact in the vCard format.	oauth_token={access-token}
		contact={contactid}
			Retrieve information about an individual contact from the NAB.
			The API should transparently return the vCard as stored by the NAB, with the requirement to support both 2.1 and 3.0 vCard formats at least.

UNI-NAB-003	The Network Address Book API SHALL support delivery of notifications regarding updates to contacts in the		See “Common notification channel” for establishment of notification channel.

	NAB.		
UNI-NAB-004	The Network Address Book API SHALL support deleting temporary resources which were created by the instance of an application (e.g., subscription for notifications).	oauth_token={access-token}	

UNI-NAB-005	The Network Address Book API SHOULD support creating a new contact in the NAB.	oauth_token={access-token}
		contact={contactid}, contact data
			Add a contact in the NAB.
			The answer will contain the contact identity assigned by the server for the new contact. This contact identity should be used by the rest of APIs (e.g., chat, file transfer, etc).
			If the contact already exists, then the operation will be rejected.

UNI-NAB-006	The Network Address Book API SHOULD support updating a new contact in the NAB.	oauth_token={access-token}
		contact={contactid},
		contact data
			Update a contact in the NAB.

4.4 Capability Management API Requirements

4.4.1 Capability Discovery

Capability discovery is one of the key functionalities and shall be exposed by the RCS API gateway.

Subject to a Service Provider policy, applications created using the APIs shall be able to register and exchange new capabilities to ascertain whether the other user supports that application.

This API can be mapped to different Capability Management mechanisms in the underlying network, such as SIP OPTIONS or Presence.

The following table describes the UNI API requirements for the capability discovery:

Label	Description	Required parameters	Comment

UNI-CPD-001	The Capability Discovery API SHALL be able to register a new service capability feature tag related to the application. This capability shall be enabled by UNI-CPD-003 before being exposed by the	oauth_token={access-token}
		capability: {capability_id}
			Use case: Game application using RCS to discover which contacts are also available for gaming.
			
			Note: Registering new application feature tags is subject to operator policies.

	application on behalf the user.		

UNI-CPD-001b	The Capability Discovery API SHALL be able to unregister a previously registered capability feature tag related to the application.	oauth_token={access-token}
		capability: {capability_id}
			Return value can consist of a list of capabilities.

UNI-CPD-002	The Capability Discovery API SHALL be able to enable or disable any standard RCS capability or a custom application registered capability per application instance.	oauth_token={access-token}
		enabled:{true/false}
		capability: {capability_id}	

UNI-CPD-003	The Capability Discovery API SHALL allow an application to query the service capabilities of a certain contact or list of contacts.	oauth_token={access-token}
		contact:{ }
			Return value can consist of a (possibly empty) list of capabilities (per contact).
			
			When the query is for a list of contacts, the return value should be a list of contacts (and their capabilities).
			
			Optionally the API Gateway may return remaining results in subsequent responses (e.g. as Server to Client Notifications)
			
			The network element providing this API should answer any incoming user capability user request (e.g. OPTIONS received from remote user) returning only the feature tags related to the enabled capabilities (see UNI-CPD-002).
			
			The refreshing of the capabilities exposed by the gateway is subject to operator policy, for example, to avoid abuse or impact in network load.

UNI-CPD-003a	The Capability Discovery API SHALL support retrieval of its own service capabilities	oauth_token={access-token}	Return value can consist of a list of capabilities.
UNI-CPD-004	The Capability Discovery API SHALL support receiving real time capability requests from the network and forwarding them to the application	oauth_token={access-token}	See “Common notification channel” for establishment of notification channel. Mechanism to be supported up to Service Provider policy. Applies to Capability Discover based in SIP OPTIONS
UNI-CPD-005	The Capability Discovery API SHALL allow an application to reply to real time capability requests with current capabilities	oauth_token={access-token}	Mechanism to be supported up to Service Provider policy. Applies to Capability Discover based in SIP OPTIONS

4.4.2 User Discovery

User discovery supports an application to find out which of a user’s contacts are RCS enabled. This API is typically called when an application initializes its address book.

Label	Description	Required parameters	Comment

UNI-CPD-004	The Capability Discovery API SHALL allow an application to query if a certain contact or list of contacts is RCS capable or not.	oauth_token={access-token}
		contact: { }
		Return value: {userType}
			Return value: {userType=RCS or empty} (per contact).
			When the query is for a list of contacts, the return value should be a list of contacts (and the associated flag per contact).
			Optionally the API Gateway may return remaining responses in subsequent responses (e.g. as Server to Client Notifications).

4.5 Presence UNI API requirements

4.5.1 Publish Presence information and content

Label	Description	Required parameters (not complete list)	Comment
UNI-PRS-001	The Presence API	oauth_token={access-token}	Ref: [RCS5.3] ch 3.7.1.3

	SHALL support management of "free-text" presence attribute.	text={text} (e.g. "My picture is updated!")	Social Presence Attributes, ch 3.7.4.2.2 Person

UNI-PRS-002	The Presence API SHALL support management of "portrait icon" which includes upload of the icon.	oauth_token={access-token}
		image={image} (jpeg/png etc.)
			RCS specific requirements regarding size, aspect ratio, file type, etc., should be verified by the RCS API GW.
			Ref: [RCS5.3] ch 3.7.1.3 Social Presence Attributes, ch 3.7.4.2.2 Person, ch 3.7.4.3.2.2 Status Icon

UNI-PRS-003	The Presence API SHALL support management of "favourite link" presence attribute.	oauth_token={access-token}
		url={url} (e.g. "http://myblog.blogspot.com")
		label={text} (e.g. "My blog")
			Ref: [RCS5.3] ch 3.7.1.3 Social Presence Attributes, ch 3.7.4.2.2 Person

UNI-PRS-004	The Presence API SHALL support management of "location" presence attribute.	oauth_token={access-token}
		text={text} (e.g. "Herentals, Belgium")
		map_coordinate={coordinate} (format following RCS e.g. "51.1644 4.7880")
		map_radius={radius} (e.g. "10")
		timezone={offset} (e.g. "+120")
			Ref: [RCS5.3] ch 3.7.4.3.3 Geolocation Information, ch 3.7.4.2.2 Person

UNI-PRS-005	The Presence API SHALL support management of "availability status" presence attribute.	oauth_token={access-token}
		status="Available" / "Not Available"
			Ref: [RCS5.3] ch 3.7.1.3 Social Presence Attributes, ch 3.7.2.2 Person

UNI-PRS-005a	The Presence API SHALL support management of multiple Social Presence Information attributes as a set.	oauth_token={access-token} list of attributes (with value) to be modified	This would support updating of multiple attributes out of the set of SPI attributes, in a single request.

4.5.2 Retrieval of presence information, subscriptions, notifications, and presence relationship management

Label	Description	Required parameters (not complete list)	Comment

UNI-PRS-006	The Presence API SHALL support	oauth_token={access-token}
		contact={contactId}
			Adding an additional user to the "rcs" list will trigger a

	invitation of a member to share presence information.	allow_location=true (or false)	presence invitation toward the other party. Contact can be any URI (MSISDN, SIP URI or reference/object to a contact received via the NAB API) Ref: [RCS5.3] ch 3.7.1.4 Social Presence Authorisation, ch 3.7.4.5.4 Client Procedures, Initiation of Presence Sharing

UNI-PRS-007	The Presence API SHALL support cancellation of invitation for sharing presence information.	oauth_token={access-token}
		contact={contactId}
			An presence sharing invitation can be cancelled only before the invitation has been accepted by the presentity (TBD if needed)
			Ref: [RCS5.3] ch 3.7.1.4 Social Presence Authorisation, ch 3.7.4.5.4 Client Procedures, Initiation of Presence Sharing

UNI-PRS-008	The Presence API SHALL support retrieval of presence information for a given contact or list of contacts.	oauth_token={access-token}
		contact={}
			The returned presence information structure is to be defined, but must be on higher abstraction level than the existing protocol (possibly JSON)
			Note that the “contact” parameter is a placeholder for a parameter construct that allows addressing a contact as well as a contact list.
			Ref: [RCS5.3] ch 3.7.1.4 Social Presence Authorisation, ch 3.7.4.3.3 Multidevice Handling, ch 3.7.4.5 Subscriptions and Authorisation
			Note: Requirement placed here to avoid renumbering after editorial changes.

UNI-PRS-009	The Presence API SHALL support subscriptions and notifications for presence sharing invitation.		See “Common notification channel” for establishment of notification channel.
UNI-PRS-010	The Presence API	oauth_token={access-token}	Accepting a presence

	SHALL support management (i.e., accept, block, ignore, revoke) of presence sharing invitations.	contact={contactId}
		allow_location=true (or false)
			invitation is done by adding the user to the “rcs” list or “basic spi only” list
			[RCS5.3] ch 3.7.1.4 Social Presence Authorisation, ch 3.7.4.5.4 Client Procedures, Initiation of Presence Sharing
			
			Adding a contact to blocked list should automatically result in removing the same contact from the “rcs” or “basic spi only“ list
			Ref: [RCS5.3] ch 3.7.1.4 Social Presence Authorisation, ch 3.7.4.5.4 Client Procedures, Initiation of Presence Sharing
			
			Adding a contact to revoke a list should automatically result in removing the same contact from “rcs” or “basic spi only“ list
			Ref: [RCS5.3] ch 3.7.1.4 Social Presence Authorisation, ch 3.7.4.5.5 Client Procedures, Removal of Presence Sharing

UNI-PRS-011	The Presence API SHALL support retrieval of presence information for the own presentity.	oauth_token={access-token}	The returned presence information structure is to be defined, but must be on higher abstraction level than the existing protocol (possibly JSON) Ref: [RCS5.3] ch 3.7.1.4 Social Presence Authorisation, ch 3.7.4.3.3 Multidevice Handling, ch 3.7.4.5 Subscriptions and Authorisation

UNI-PRS-012	The Presence API SHALL support subscriptions and notifications for presence information changes both for its own presentity or a list	oauth_token={access-token}
		contact={}
		“Structured presence information from presentities that the user shares presence information with”
			Receive notifications about presence information changes from the presentities.
			See “Common notification channel” for establishment of notification channel.

	of contacts.		The returned presence information structure to be defined but must be on higher abstraction level than the existing protocol (possibly JSON). Note that the "contact" parameter is a placeholder for a parameter construct that allows addressing a contact as well as a contact list. Ref: [RCS5.3] ch 3.7.1.4 Social Presence Authorisation, ch 3.7.4.5.1 Subscriptions and Authorisation Overview
UNI-PRS-013	The Presence API SHALL support querying for pending presence invitations.	oauth_token={access-token}	Application gets all pending presence invitations (including those possibly received while application is offline).

4.5.3 Services capabilities

The requirements below shall allow a user to read their own Service Capabilities and to request service capabilities for a presentity ("who can I invite").

Label	Description	Required parameters (not complete list)	Comment
UNI-PRS-014	The Presence API SHALL support retrieval of its own service capabilities	oauth_token={access-token}	Ref: [RCS5.3] ch 2.6.1.2.5.1 Service-descriptions for the Selected RCS Services, ch 3.7.4.3.3 Multidevice Handling, ch 3.7.4.5 Subscriptions and Authorisation

UNI-PRS-015	The Presence API SHALL support retrieval of service capabilities for a contact ("who can I invite") or a list of contacts.	oauth_token={access-token}
		contact={ }
			Contact can be any URI (MSISDN, SIP URI or reference/object to a contact received via the NAB API).
			Aggregation via different notifications is possible for the response.
			Ref: [RCS5.3] ch 3.7.1.4 Social Presence Authorisation, ch 2.6.3.7 Social presence, 2.6.1.2.3 Service Capabilities

			Retrieval, ch.2.13.2 Privacy

4.6 Messaging UNI API requirements

This operation allows sending and receiving text and multimedia messages, and being notified about the message delivery status.
For CPM Standalone Messages, three message disposition notifications are specified in RCS, using the same message dispositions that are defined for chat in Section 4.7.5:

1. sent
2. delivered
3. displayed

Label	Description	Required parameters (not complete list)	Comment

UNI-MSG-001	The Messaging API SHALL support sending messages.	oauth_token={access-token}
		recipient = {contact(s)}
		deliveryNotification = "yes"/"no"
		{content}
			Content can be text or multimedia.
			Bearer service selection (SMS, MMS, CPM Standalone Messaging or other) should not be a mandatory parameter, allowing for bearer selection by API GW or Service Provider policies.
			A Message send request resource is created which will exist until the delivery confirmation is provided to the application.
			This resource will be automatically deleted by the messaging server once the delivery confirmation has been provided to the application (regardless of mechanism used – see receive message).
			Ref: [RCS5.3] ch 3.2 Standalone messaging

UNI-MSG-002	The Messaging API SHALL support receiving messages.	oauth_token={access-token}	See "Common notification channel" for establishment of notification channel.

UNI-MSG-003	The Messaging API SHALL support receiving of the message disposition ("sent", "delivered", "displayed") .	oauth_token={access-token}
		result_code={"sent", error condition}
			The message delivery and display notification are requested according to Service Provider policies, when a message is sent on API GW.

			The "sent" disposition is received synchronously as response to the request that sends the message. The "delivered" and "displayed" dispositions are returned asynchronously via the notification channel. See "Common notification channel" for establishment of notification channel. Ref: [RCS5.3] ch 3.2 Standalone messaging

UNI-MSG-004	The Messaging API SHALL support sending "displayed" notifications of message received	oauth_token={access-token}
		message id={message-id}
			The message-id parameter value shall be the one received in the incoming message.
			This operation will be allowed only if the original message included a "displayed" notification request.
			Ref:[RCS5.3] ch 3.2 Standalone messaging

4.7 Chat UNI API requirements

4.7.1 Confirmed One to One Chat

The application is in full control of the session management, requiring an explicit acceptance before the chat session is established. Several parallel sessions between two users inside the application are possible using this model.

Note: Requirements in this section have been rearranged for better understanding and clarity. To avoid impact on external references, requirement numbers have not been changed. As a result, numbering is not consecutive in some cases.

4.7.1.1 Session Management originating side

The operations listed below allow the originating side of a chat to manage the chat session.

Label	Description	Required parameters (not complete list)	Comment

UNI-CHT-001	The Chat API SHALL support starting a 1-to-1 chat.	oauth_token={access-token}
		recipient={contactid}
		subject={text} (e.g. "Dinner tonight")
			Use case: Start a chat.
			Contact can be any URI (MSISDN, SIP URI or reference/object to a contact received via the Address Book API).

			Subject parameter is optional and is the topic of the chat; included when it is provided. Ref: [RCS5.3] ch 3.3 1-to-1 Chat, [RCSR5OMAIMEND] ch 7.1.1 Originating Client Procedures

UNI-CHT-001a	The Chat API SHALL support starting a 1-to-1 chat with initial message	oauth_token={access-token}
		recipient={contactid}
		subject= {text} (e.g. “Dinner tonight”)
		message={text|multimedia content} (e.g. “Hi”)
			Use case: Start a chat.
			Contact can be any URI (MSISDN, SIP URI or reference/object to a contact received via the Address Book API).
			
			This requirement extends the requirement UNI-CHT-001.
			
			Subject parameter is optional and is the topic of the chat; it is included when provided.
			
			Message parameter is optional and is the first message of the chat; it is included when provided, according to Service Provider policies.
			
			Ref: [RCS5.3] ch 3.3 1-to-1 Chat, [RCSR5OMAIMEND] ch 7.1.1 Originating Client Procedures

UNI-CHT-003a	The Chat API SHALL support cancelling a 1-to-1 chat invitation	oauth_token={access-token}	Use case: User cancels a chat invitation. Cancellation is only possible as long as the invitation has not been accepted. Ref: [RCSR5OMAIMEND] ch 7.1.1 Originating Client Procedures
UNI-CHT-004a	The Chat API SHALL support notifications about chat (accepted, cancelled; declined, ended)	oauth_token={access-token}	

UNI-CHT-005	The Chat API SHALL support ending a 1-to-1 chat session by the originating side	oauth_token={access-token}	Use case: User ends 1-to-1 chat. Ref: [RCSR5OMAIMEND] ch 7.1.1 Originating Client Procedures
UNI-CHT-006	VOID	VOID	VOID

4.7.1.2 Session Management terminating side

The operations listed below allow the terminating side of a chat to manage its participation in a chat session.

Label	Description	Required parameters (not complete list)	Comment
UNI-CHT-007a	The Chat API SHALL support notifications about incoming chat invite.	Information about inviting user; subject if provided; and/or first message if provided	Use case: The user is invited to a chat session. It might be possible that the inviting user is not in the contact list. See “Common notification channel” for establishment of notification channel. Ref: [RCS5.3] ch 3.3 1-to-1 chat, [RCSR5OMAIMEND] ch 7.1.2 Terminating Client Procedures
UNI-CHT-008a	The Chat API SHALL support accepting a chat invitation.	oauth_token={access-token}	Use Case: User accepts chat invitation. Ref: [RCS5.3] ch 3.3 1-to-1 Chat, [RCSR5OMAIMEND] ch 7.1.2 Terminating Client Procedures
UNI-CHT-009a	The Chat API SHALL support declining a chat invitation.	oauth_token={access-token}	Use Case: User declines chat invitation. Ref: [RCS5.3] ch 3.3 1-to-1 Chat, [RCSR5OMAIMEND] ch 7.1.2 Terminating Client Procedures
UNI-CHT-010	The Chat API SHALL support ending a 1-to.1 chat by the terminating side.	oauth_token={access-token}	Use case: User ends chat. Ref: [RCS5.3] ch 3.3 1-to-1 Chat, [RCSR5OMAIMEND] ch 7.1.2 Terminating Client Procedures
UNI-CHT-012a	The Chat API SHALL support notifications about “chat ended”.		Use case: Remote user ends chat. Application of the terminating user receives a notification about that event. See “Common notification

			channel" for establishment of notification channel. [RCSR5OMAIMEND] ch 7.1.2 Terminating Client Procedures

4.7.2 Adhoc One to One Chat

In this chat model there is no explicit chat invitation associated to the 1-to-1 chat. From the functional point of view the user sends a message to another user and it is responsibility of the client implementation to open any underlying SIP/MSRP sessions to deliver that message. This complexity is hidden to the user.

Also from the receiver point of view, the user does not accept or decline a 1-to-1 chat invitation; they simply receive a new message from a user. Therefore, it is not possible for a user to be able to accept or reject an SIP/MSRP session from the client application and the establishment mechanism is controlled by the client application according to the MNO rules.

Thus, no functional requirements associated with 1-to-1 chat establishment (for either the originating or terminating side) are required by this model.

However, information regarding the technical establishment or ending of the underlying IM session (i.e., SIP and MSRP session) are out of scope of this API specification.

The only requirements applicable then to the 1-to-1 chat in this model are the ones related to the media and the notifications.

4.7.3 Group chat

The operations listed below allow managing a group chat. In this release only long-lived group chats are supported. Hence requirements in this section superseded by requirements in section 4.7.4 are removed.

Label	Description	Required parameters (not complete list)	Comment
UNI-CHT-002b	VOID	VOID	VOID Note: Covered by UNI-CHT-030
UNI-CHT-003b	The Chat API SHALL support cancelling a group chat invitation.	oauth_token={access-token}	Use case: User cancels a chat invitation. Cancellation is possible only as long as the invitation has not been accepted. Ref: [RCSR5OMAIMEND] ch 7.1 IM Client Procedures for IM Sessions
UNI-CHT-004b	The Chat API SHALL support notifications about group chat (accepted, cancelled; declined, ended) as well as all services supported within group	If the group chat session is accepted the notification shall also carry the list of supported services within the group chat.	The list of services supported by the RCS enabler within the group chat shall be considered during related API calls, e.g. UNI-FLT-001.

	chat.		
UNI-CHT-007b	VOID	VOID	VOID Note: Covered by UNI-CHT-032
UNI-CHT-008b	VOID	VOID	VOID
UNI-CHT-009b	VOID	VOID	VOID
UNI-CHT-011	VOID	VOID	VOID Note: Covered by UNI-CHT-033
UNI-CHT-012b	VOID	VOID	VOID Note: Covered by UNI-CHT-038
UNI-CHT-013	VOID	VOID	VOID Note: Covered by UNI-CHT-039
UNI-CHT-014	VOID	VOID	VOID Note: Covered by UNI-CHT-031
UNI-CHT-015	VOID	VOID	VOID Note: Covered by UNI-CHT-040
UNI-CHT-016	VOID	VOID	VOID Note: Already covered by the initial subscription of the client to chat related notifications.
UNI-CHT-017	VOID	VOID	VOID Note: Covered by UNI-CHT-036

4.7.4 Long Lived Group Chat

In the Long Lived Group, the session management complexity is handled internally by the gateway and only the high level functionality related to the Long Lived Group chat user experience is exposed in the API.

Apart from the media and notification requirements in chapters 4.7.5 and 4.7.6 which are shared with the session aware group chat requirements in chapter 4.7.3, the following requirements shall be fulfilled:

Label	Description	Required parameters (not complete list)	Comment

UNI-CHT-030	The Long Lived Group Chat API SHALL support a user to create a new Long Lived Group	oauth_token={access-token}
		recipient={contact1}, {contact2}, …
		subject={text} (e.g. “Hi”)
			The group chat ID will be generated internally by the API GW and used according to the RCS 5.1 spec chapter 3.4.4.1.1 Initiating a Group

	indicating the list of participants and the subject of the group.	closed={true, false}
		gc_services={ft, geopushft}
		
		returns {group_chat_id}
			Chat.
			The group chat is regular by default if closed parameter is not specified (consistent with UNI-CHT-002b and UNI-CHT-013).
			A list of all RCS services supported by the application within the group chat shall be provided during group chat setup in related API calls and notifications. Currently, with [RCS5.3] the list may include File Transfer and Geolocation Push. It may be extended later. Without parameter gc_services it is assumed that no further service is supported within group chat.

UNI-CHT-031	The Long Lived Group Chat API SHALL allow a user to add a user or a list of users to a Long Lived Group Chat.	oauth_token={access-token}
		group_chat_id={group_chat_id}
		recipient={contact1}, {contact2}, …	

UNI-CHT-032	The Long Lived Group Chat API SHALL notify the user when it has been added to Long Lived Group Chat.	The notification SHALL contain information regarding the Long Lived Group Chat.	The list of services supported by the RCS enabler within the group chat shall be provided and considered during related API calls, e.g. UNI-FLT-001.

UNI-CHT-033	The Long Lived Group Chat API SHALL allow an user to leave a Long Lived Group Chat.	oauth_token={access-token}
		group_chat_id={group_chat_id}
			When the user leaves a Long Lived Group Chat it SHALL not be allowed to post any new messages to it.
			
			The time to keep storing the information regarding a Long Live group chat in the API GW after the user has left it, is up to service provider polices.

UNI-CHT-034	The Long Lived Group Chat API SHALL allow a user to query the Long Lived Group Chats	oauth_token={access-token} return {list of group chat ids + subjects}	Based on service provider policies the list of group chats returned for a user may be restricted to just the ones that the calling

	and their subjects for the user.		application created for the user.

UNI-CHT-035	The Long Lived Group Chat API SHALL allow querying the detailed information about a Long Lived Group Chat	oauth_token={access-token}
		group_chat_id={group_chat_id}
		
		return {subject, participant list, open/close,...}
			The information SHALL contain at least the participant list and the subject of the group chat and the supported services.

UNI-CHT-036	The Long Lived Group Chat API SHALL notify the application when the participant list of a Long Lived Group chat has changed.	The notification SHALL contain the list of new participants and/or participants leaving it.	
UNI-CHT-037	The Long Lived Group Chat API SHALL notify the application when the list of supported services of a Long Lived Group chat have changed.	The notification SHALL contain the new list of services supported during group chat.	The list of services supported by the RCS enabler within the group chat shall be provided and considered during related API calls (e.g., UNI-FLT-001).
UNI-CHT-038	The Long Lived Group Chat API SHALL notify the application when a Long Lived Group Chat is no longer available.		A long lived group chat is no longer available when it is removed from the list of group chats stored by the API GW. The decision when to disable a long lived group chat is based on service provider policies.

UNI-CHT-039	The Long Lived Group Chat API SHALL allow to extend a 1-to-1 confirmed to a Long Lived Group chat	oauth_token={access-token}
		chat_id={chat_id}
		contact={contactId1,contactId}
		closed={true, false}
		gc_services={ft, geopushft}
		
		return {group_chat_id}
			Based on service provider policies this operation may not be allowed.
			The group chat is regular by default if closed parameter is not specified (consistent with UNI-CHT-002b and UNI-CHT-013).
			A list of all RCS services supported by the application within the group chat shall be provided during group chat setup in related API

			calls and notifications. Currently, with [RCS5.3] the list may include File Transfer and Geolocation Push. It may be extended later. Without parameter gc_services it is assumed that no further service is supported within group chat.

UNI-CHT-040	The Long Lived Group Chat API SHALL allow a user to re-join a long lived group chat after the user has left it.	oauth_token={access-token}
		group_chat_id={group_chat_id}
			Based on service provider policies this operation may not be allowed.

4.7.5 Media

The operations listed below allow handling the media in a chat.

Label	Description	Required parameters (not complete list)	Comment

UNI-CHT-018	The Chat API SHALL support sending text messages	oauth_token={access-token}
		message_content={content}
		chat_id={contactid|sessionid|group chat id}
		
		return:
		status: {success, pending, failure}
			Use case: The application sends a chat message.
			Content can be text or multimedia according to RCS specifications. This API is for text message support. The multimedia content support is covered by UNI-CHT-026.
			The chat_id parameter can be contactid for ad-hoc 1-to-1 chat, or sessionid for confirmed 1-1 chat and group chat, or group chat id for Long Lived group chat.
			The status of the request for sending message is returned:success, pending,failure.
			One example for the failure case is the chat id is invalid.
			In case the transaction is to take too much time to be completed it shall be possible to return a "pending" response and

			return the final delivery status asynchronously via the notification channel. Ref: [RCSR5OMAIMEND] ch 7.1 IM Client Procedures for IM Sessions

UNI-CHT-019	The Chat API SHALL support sending of "isComposing".	oauth_token={access-token}
		isComposing="active"/"idle" "timeout=xx"" …
		chat_id={contactid|sessionid|group chat id}
			Use case: The application sends "isComposing" which indicates that a user is currently composing a message.
			The chat_id parameter can be contactid for ad-hoc 1-to-1 chat, or sessionid for confirmed 1-1 chat and group chat, or group chat id for Long Lived group chat.
			Same as [UNI-CHT-018] with "isComposing" as a special kind of content, parameters according to RFC 3994.
			If the message delivery was successful a "success" response is returned.
			Ref: [RCSR5OMAIMEND] ch 7.1 IM Client Procedures for IM Sessions

UNI-CHT-020	The Chat API SHALL support receiving messages.	oauth_token={access-token}	Use case: The application receives a chat message via the notification mechanism. Timestamp value shall be also notified to the application if it was included in the message. Information regarding "display" notification request for the message shall be also included if present in the original message. The chat_id (sessionid, contactid, or group chat id) information is included in the notification for application to identify the chat session. See "Common notification channel" for establishment of notification channel.

			Ref: [RCSR5OMAIMEND] ch 7.1 IM Client Procedures for IM Sessions
UNI-CHT-021	The Chat API SHALL support receiving the “isComposing” message.	oauth_token={access-token}	Use case: the application receives via the notification mechanism an indication that a user is currently composing a message. The chat_id (sessionid, contactid, or group chat id) information is included in the notification for application to identify the chat session. Same as [UNI-CHT-020] with “isComposing” as a special kind of content. Ref: [RCSR5OMAIMEND] ch 7.1 IM Client Procedures for IM Sessions

UNI-CHT-026	The Chat API SHALL support sending multimedia chat messages.	oauth_token={access-token}
		message_content = Body{multimedia content}
		content type={content type}
		chat_id={contactid|sessionid|group chat id}
		
		return:
		status: {success, pending, failure}
			Use case: The application sends a multimedia chat message (e.g., image, video clip, audio clip, etc).
			The chat_id parameter can be contactid for ad-hoc 1-to-1 chat, or sessionid for confirmed 1-1 chat and group chat, or group chat id for Long Lived group chat.
			The status of the request for sending message is returned:success, pending, failure.
			One example for the failure case is the chat id is invalid.
			In case the transaction is to take too much time to be completed it shall be possible to return a “pending” response and return the final delivery status asynchronously via the notification channel.
			Ref: [RCS5.3] ch 3.2.1.1 Standalone messaging and ch 3.3.1 1to-1 Chat Feature description, [RCSR5OMAIMEND] ch 7.1

			IM Client Procedures for IM Sessions

UNI-CHT-027	The Chat API SHALL support notifications indicating that a multimedia chat message has been received and is available for download	content-type={type}
		url={file url}
			The API gateway will send this notification to the client with an URL to download the content.
			The chat_id (sessionid, contactid, or group chat id) information is included in the notification for application to identify the chat session.
			The server which the URL is pointed to SHALL be ready to receive download requests when the notification is sent.

4.7.6 Notifications

In the RCS specification, three notifications associated to messages have been specified:

1. "Sent" notification: generated when the RCS client has successfully sent the message. In the case of the APIs it should be generated by the API gateway and the application should be notified when it has successfully sent the message.
2. "Delivery" notification: generated when the message arrives at the final destination. In the case of the APIs, the API gateway will receive the notification from the IM Server about a previously sent message and it will notify the application accordingly. The API gateway is also responsible for sending back the delivery notifications of incoming messages as they are received by the application. To avoid sending delivery notifications for messages that are not correctly received (i.e., the application fails to fetch the message while it is in the notification channel), it is highly recommended that the API gateway sends the "delivery" notification for incoming messages only after the message has been successfully delivered to the application in the notification channel.
3. "Displayed" notification: generated by the RCS client when a message is displayed on the RCS device. For privacy issues, an RCS user is able to enable or disable the sending of "displayed" notifications. In the case of APIs, the application is responsible for generating these "displayed" notifications accordingly. The API gateway shall also be able to receive them and notify the application.

References: [RCS5.3] Section 3.3 and 3.4.

The operations listed below allow handling of the message related notifications.

Label	Description	Required parameters (not complete list)	Comment
UNI-CHT-022	The Chat API SHALL support receiving		Message notifications SHALL be returned

	messages notifications [“sent”, “delivered” and “displayed”] for messages sent in a 1 to 1 session.		asynchronously via the notification channel except the “sent” notification. As stated in UNI-CHT-018 when “success” response is returned it SHALL be considered as the sent notification.

UNI-CHT-023	The Chat API SHALL support sending “displayed” notifications of 1 to 1 message received.	oauth_token={access-token}
		message id={message-id}
			The message-id shall be the one received in the incoming message.
			This operation will be allowed only if the original message included a “displayed” notification request.
			If the confirmed (session aware) model is used it shall be possible to send the “displayed” notifications even if the chat session has been terminated

UNI-CHT-024	The Chat API SHALL support receiving messages notifications [“sent”, “delivered” and “displayed”] for messages sent in group chat.		Message notifications SHALL be returned asynchronously via the notification channel except the “sent” notification. As stated in UNI-CHT-018 when “success” response is returned it SHALL be considered as the sent notification.

UNI-CHT-025	The Chat API SHALL support sending “displayed” notifications of group message received.	oauth_token={access-token}
		message id={message-id}
			The message-id shall be the one received in the incoming message.
			This operation will be allowed only if the original message included a “displayed” notification request.

4.8 File Transfer UNI API requirements

4.8.1 Introduction (informative)

The following tables show the functional requirements for the file transfer API. A file could be sent to a single recipient or to multiple recipients within an active group chat if supported by the related RCS enabler. The file transfer API can also be used for RCS 5.1 location and VCard features by sending and receiving the location or VCard data as file content.

4.8.2 Originating side

Label	Description	Required parameters (not complete list)	Comment

UNI-FLT-001	The File Transfer API SHALL support initiating a file transfer to a single recipient or to a group of recipients within a group chat.	oauth_token={access-token}
		
		recipient={contactid}
		or
		chat_id={session_id|group_chat_id}
		
		file-icon={reduced image}
		file-name={file name}
		file-size={size}
		file-type={type}
		file={file}
		
		url={url to the file}
		or
		BODY{image file}
			Initiate a file transfer session with the selected recipient or re-use an active group chat session for file transfer to all group chat members. In case a group chat does not exist it is to be initiated by using UNI-CHT-002b or UNI-CHT-30.File transfer within a group chat is supported only if notified by the RCS enabler.
			A SIP INVITE request is sent to the remote party (i.e., the contact).
			A file transfer instance is created at the reception of indication that invite and initial message were delivered (SIP 180).
			
			The file could be sent either in the body of the request or via an URL to the actual file.
			
			Ref: [RCS5.3] ch 3.5 File Transfer, ch 3.5.4.2 File Transfer in Group Chat, [RCSR5OMAIMEND] ch 10.1 File Transfer

UNI-FLT-002	The File Transfer API SHALL support cancelling a file transfer invitation by the originating side.	oauth_token={access-token}	Use case: An ongoing file transfer session is to be cancelled. Only the user who created the invitation can cancel it, and it is offered

			only before the file transfer is accepted or rejected. Ref: [RCSR5OMAIMEND] ch 10.1 File Transfer
UNI-FLT-003	The File Transfer API SHALL support ending a file transfer session by the originating side.	oauth_token={access-token}	The selected resource (i.e., the file transfer session, is to be closed. A SIP BYE request for the selected session is sent to the remote party. Ref: [RCSR5OMAIMEND] ch 10.2 File Transfer Session Release
UNI-FLT-004	The File Transfer API SHALL support notifications about “File Transfer” (accepted, declined, cancelled, ended) to the originating side.		The final set of applicable notification types will be determined in the technical work phase. See “Common notification channel” for establishment of notification channel.
UNI-FLT-004b	The File Transfer API SHALL support indication of file transfer progress status, including indication of resumption		Use Case: Support of a progress bar in the Application UI. In case of file transfer resumption, the application informs the user of the resumption (i.e., anticipating longer transferring time). The gateway sends the application the progress status to the application at a specified interval. (i.e., every xx second or xx% of the file size). As the API gateway supports the file transfer resume operation (initiated by either sending or receiving client), the API gateway will notify the application of the resumption using a

			unique status code . The final set of applicable notification codes/types will be determined by OMA in its technical API work. Ref: [RCS5.3] ch 3.5 File Transfer, ch 3.5.3 High Level Requirements

4.8.3 Terminating side

Label	Description	Required parameters (not complete list)	Comment
UNI-FLT-005	The File Transfer API SHALL support notifications about file transfer invitation for 1-to-1 file transfer and file transfer within a group chat.	Information about the file transfer originator and in case of file transfer within group chat about the group chat.	Use case: The user is invited to a file transfer session. The file may be sent during a group chat to all chat users. In that case, a reference to the related group chat shall be provided. See “Common notification channel” for establishment of notification channel. Ref: [RCS5.3] ch 3.5 File Transfer, ch 3.5.4.2 File Transfer in Group Chat, [RCSR5OMAIMEND] ch 10.3 Client Receiving File Transfer Request Session Release
UNI-FLT-006	The File Transfer API SHALL support accepting a file transfer invitation by the terminating side.	oauth_token={access-token}	Use case: File transfer session is to be accepted. Ref: [RCSR5OMAIMEND] ch 10.3 Client Receiving File Transfer Request
UNI-FLT-007	The File Transfer API SHALL support declining a file transfer invitation by the terminating side.	oauth_token={access-token}	Use case: File transfer session is to be rejected. The SIP INVITE request is then rejected with a SIP 603 response. Ref: [RCSR5OMAIMEND] ch 10.3 Client Receiving File Transfer Request
UNI-FLT-008	The File Transfer API SHALL support ending a file transfer by the terminating	oauth_token={access-token}	Use case: File transfer session is to be closed. A SIP BYE request for the

	side.		selected session is sent to the remote party. Ongoing file transfer can be cancelled only after the session is established. Ref: [RCSR5OMAIMEND] ch 10.1 File Transfer
UNI-FLT-009	The File Transfer API SHALL final state notifications about the MSRP transfer session (“success”, “abort” and “error”)to the terminating side.		The final set of applicable notification types will be determined in the technical work phase. See “Common notification channel” for establishment of notification channel.
UNI- FLT-010	The File Transfer API SHALL support notifications indicating that the file transfer content is available for download.	url={file url}	The gateway will send this notification to the client with an URL to download the image. The URL SHALL be ready to start downloading when the notification is sent. It is up to the implementation to decide whether this is sent when the first chunks of MSRP data are received and allow it to simultaneously receive data from the MSRP session and HTTP download or if it waits for the MSRP session to be completed and only allows the download to be started when the whole file has been received. In any case the notification SHALL be sent before the final state notification is sent.
UNI- FLT-011	The File Transfer API SHALL support indication of file transfer progress status, including indication of resumption.		Use Case: Support of a progress bar in the application UI. In case of file transfer resumption, the application informs the user of the resumption (i.e., anticipating longer transferring time). The gateway sends the application the progress status to the application at a

			specified interval (i.e., every xx second or xx% of the file size). As the gateway supports the file transfer resume operation (initiated by either sending or receiving client), it will notify the application of the resumption with a unique status code. The final set of applicable notification codes/types will be determined by OMA in its technical API work. Ref: [RCS5.3] ch 3.5 File Transfer; ch 3.5.3 High Level Requirements

4.9 Call Control and Notification UNI API requirements

The Call Control and Notification UNI API requirements are based on OMA Parlay REST Third-Party Call Control and Call Notification APIs.

4.9.1 Call Functionality available to originating side

The operations listed below allow an application to manage a call session and to receive call progress notifications on behalf of the originating side (i.e., “calling participant”, “A-Party”).

Label	Description	Required parameters (not complete list)	Comment

UNI-CLL-001	The Call API(s) SHALL support initiating a call session with a called party.	oauth_token={access-token}
		recipient={contactid}
			Use case: The user initiates a call between its own terminal and another user.
			Initiating a session results in all of the user’s terminals being rung. The user answers on one of his terminals. After this, the call is set up to the intended recipient.

UNI-CLL-002	The Call API(s) SHALL support the cancellation of the call session initiation.	oauth_token={access-token}	Use case: The user interrupts call attempt.

4.9.2 Call functionality available to originating side and terminating side

The operations listed below allow an application to receive call progress notifications and to terminate a call session on behalf of the call participants [“calling participant” (“A-Party”) as well as “called participant” (“B-Party”)). The term “user” listed below therefore subsumes both “A-party” as well as “B-party”.

Label	Description	Required parameters (not complete list)	Comment
UNI-CLL-003	The Call API(s) SHALL support notifications about “call attempt”.		Use case: Application receives call invitation notification that a call session is being set up to the user’s phone. See “Common notification channel” for establishment of notification channel.
UNI-CLL-004	The Call API(s) SHALL support notifications about “call accepted”.		Use case: Application receives notification that the user’s phone accepted the call. See “Common notification channel” for establishment of notification channel.
UNI-CLL-005	The Call API(s) SHALL support notifications about “busy”.		Use case: Application receives notification that the user’s phone is busy. See “Common notification channel” for establishment of notification channel.
UNI-CLL-006	The Call API(s) SHALL support notifications about “not reachable”.		Use case: Application receives notification that the user’s phone is disconnected. See “Common notification channel” for establishment of notification channel.
UNI-CLL-007	The Call API(s) SHALL support notifications about “no answer”.		Use case: Application receives notification that the user’s phone did not react to the call. See “Common notification channel” for establishment of notification channel.
UNI-CLL-008	The Call API(s) SHALL support notifications about “disconnected”.		Use case: Application receives notification that the user’s phone has ended the call.

			See “Common notification channel” for establishment of notification channel.
UNI-CLL-009	The Call API(s) SHALL support terminating a call session.	oauth_token={access-token}	Use case: The call session is terminated by the application rather than by one of the call participants on-hooking the phone.
UNI-CLL-010	VOID	VOID	VOID

4.9.3 Media Information

The operations listed below indicate how media is handled in a multimedia call.

Label	Description	Required parameters (not complete list)	Comment
UNI-CLL-011	The Call API SHALL allow indication of multiple media types; in particular, both audio and video.		Use case: The application may request media other than voice (e.g. video, text) in starting a multimedia telephony call. Ref: [IR94] ch 2.2.2 Call Establishment and Termination [IR.92] Annex B.2 Global Text Telephony
UNI-CLL-012	The Call API SHALL allow getting of current media status of a single call participant, or all the participants.		Use case: The application may request the current status of media other than voice (e.g. video, text) during an active multimedia telephony call either for a specific participant or all participants. The status includes information about the list of media types in the session, plus their direction.
UNI-CLL-013	VOID	VOID	VOID
UNI-CLL-014	VOID	VOID	VOID
UNI-CLL-015	VOID	VOID	VOID
UNI-CLL-016	VOID	VOID	VOID
UNI-CLL-017	The Call API SHALL allow control the media stream direction (i.e., unidirectional, bi-directional) for each		Use Case: To comply with privacy requirements in certain regions, the application may request at call setup that the video

	media type at call setup.		stream in a video call be in either simplex or duplex mode. Ref: [IR94] ch 2.2.2 Call Establishment and Termination

4.10 WebRTC Signaling API requirements

This section defines the requirements for a WebRTC Signaling API.

The main intent of this API is to support web applications running in a web browser that make use of the WebRTC API [W3C_WebRTC], but it should also work with other media engines that are based on [RFC3264]. The WebRTC Signaling API provides a signalling mechanism for these applications to access a Voice or Video over IP service in the network, e.g. based on the IMS.

This API shall also support requirements for the RCS Extension to Extension service in the context of WebRTC architecture.

RCS Extension to Extension service is described in section 3.12.4.2.2 of [RCS5.3]. There are two types of media types supported on the RCS UNI for this Extension to Extension service:

- based on MSRP (see section 3.12.4.2.2.1 of [RCS5.3]), or
- based on RTP (see section 3.12.4.2.2.2 of [RCS5.3]).

4.10.1 Call Functionality available to originating side

Label	Description	Required parameters	Comment

UNI-WRTCS-001	The WebRTC Signaling API SHALL support initiating a VoIP call to a called party.	oauth_token={access-token}
		recipient={contactId | E.164 number}
		
		optional:
		service={“rcsipcall”}
		allow_video_upgrade={“yes”|”no”}
		
		return {callId}
			Depending on the operator policies and if it has deployed CS breakout, the destination may need to be a VoIP user.
			
			In case the rcsipcall service is set, a capability exchange may be needed in order to ensure that the remote peer also supports the rcsipcall service if break out is not allowed by the service provider or the service is not interworked with other IP services. (VoLTE for example) See [RCS5.3] ch 3.8 for more information.
			
			If no service is indicated, the generated INVITE to establish the VoIP call will not include

			the bevoicetag (just mmtel+audio). The API GW will return an error if the operation is not allowed by the service provider policies.

UNI-WRTCS-002	The WebRTC Signaling API SHALL support initiating a Video over IP call to a called party.	oauth_token={access-token}
		recipient={contacted | E.164}
		
		optional:
		service={"rcsipcall"}
		
		return {callId}
			In case the service is not interconnected with other IP services (e.g., VoLTE), a capability exchange may be needed in order to ensure that the remote peer also supports the rcsipcall service.
			See [RCS5.3] ch 3.9 for more information.
			
			If no service is indicated, the generated INVITE to establish the VoIP call will not include the bevoicetag (just mmtel+audio+video).
			
			The API GW will return an error if the operation is not allowed by the service provider policies.

UNI-WRTCS-003	The WebRTC Signaling API SHALL support the cancellation of Voice or Video over IP call setup.	oauth_token={access-token}
		callId={callId}
		
		return {success/failed}
			The cancellation of call setup is only possible while the call is not successfully established.

UNI-WRTCS-004	The WebRTC Signaling API SHALL support notification about the VoIP call setup state.	The notifications supported may at least be "busy", "not reachable", "no answer", "declined" and "accepted". If a VoIP call is accepted, the notification shall carry also the information regarding if it is possible to upgrade the VoIP call to a Video Call as specified by the termination side.	
UNI-WRTCS-005	The WebRTC Signaling API SHALL support notification about	The notifications supported may at least be "busy", "not reachable", "no answer", "declined" and "accepted".	The terminating user may have accepted the Video over IP call but decided to not send back video. This information will be

	the Video over IP call setup state.		available by the media negotiation supported by the media requirements in the section 4.10.4 “Media”.

4.10.2 Call Functionality available to terminating side

Label	Description	Required parameters (not complete list)	Comment
UNI-WRTCS-006	The WebRTC Signaling API SHALL support notification about a new incoming VoIP call.	Information about the VoIP originator, service if present (i.e., “rcsipcall”) and callId. The notification shall also carry the information regarding whether it is possible to upgrade the VoIP call to a Video over IP call once it is set up.	
UNI-WRTCS-007	The WebRTC Signaling API SHALL support notification about a new Video over IP call.	Information about the VoIP originator, service if present (i.e. “rcsipcall”) and callId.	

UNI-WRTCS-008	The WebRTC Signaling API SHALL support accepting a VoIP call by the terminating side.	oauth_token={access-token}
		callId={callId}
		
		allow_video_upgrade={“yes”|”no”}
			The terminating side will inform whether it supports the upgrade to video. This is done via notifying to the originating side.

UNI-WRTCS-009	The WebRTC Signaling API SHALL support rejecting a VoIP call by the terminating side.	oauth_token={access-token}
		callId={callId}	

UNI-WRTCS-010	The WebRTC Signaling API SHALL support accepting a Video over IP call by the terminating side.	oauth_token={access-token}
		callId={callId}
			When accepting a Video over IP call, the user may accept it but decide to not send video back. This requirement will be supported by the media requirements in the section 4.10.4 “Media”.

UNI-WRTCS-011	The WebRTC Signaling API SHALL support rejecting a Video	oauth_token={access-token}
		callId={callId}	

	over IP call by the terminating side.		

4.10.3 Call Functionality available to originating side and terminating side

Label	Description	Required parameters (not complete list)	Comment

UNI-WRTCS-012	The WebRTC Signaling API SHALL support terminating a VoIP or Video over IP call.	oauth_token={access-token}
		callId={callId}	

UNI-WRTCS-013	The WebRTC Signaling API SHALL allow the application to request a VoIP call to be upgraded to a Video over IP call.	oauth_token={access-token}
		callId={callId}
			Only for VoIP calls which have been signalled to support upgrade to a video call in the new call or accepted notification.
			
			If the upgrade is not allowed by the service provider or fails to be requested, an error will be returned to the application.

UNI-WRTCS-014	The WebRTC Signaling API SHALL support notification of a request to upgrade a VoIP call to a Video over IP call.		

UNI-WRTCS-015	The WebRTC Signaling API SHALL allow to accept or reject the upgrade of a VoIP call to a Video over IP call.	oauth_token={access-token}
		callId={callId}
		action={accept/reject}
		send_video = {true/false}
			If the user accepts the upgrade, it will also be allowed to specify whether it wants to send back video or not.
			
			This requirement will be supported by the media requirements in the in the section 4.10.4 “Media”.

UNI-WRTCS-016	The WebRTC Signaling API SHALL support notification of the result of upgrading a VoIP call to a Video over IP call.	The notification shall carry also the information whether if the remote side accepting the video upgrade is sending video back or not.	The receiver user may have accepted the upgrade to a Video over IP call but decided to not send back video. This information will be available by the media negotiation supported by the

			media requirements in the in the section 4.10.4 “Media”.

UNI-WRTCS-017	The WebRTC Signaling API SHALL allow to downgrade a Video over IP call to a VoIP call	oauth_token={access-token}
		callId={callId}
			The API GW will return an error if the operation is not allowed by the service provider policies.

UNI-WRTCS-018	The WebRTC Signaling API SHALL support notification about a downgrade of a Video over IP call to a VoIP call.		

4.10.4 Media

Label	Description	Required parameters (not complete list)	Comment
UNI-WRTCS-019	The WebRTC Signaling API SHALL support exchanging SDPs in a way that allows to be used by a media stack compatible with [RFC3264] and/or WebRTC [W3C_WebRTC].		When referring to WebRTC [W3C_WebRTC] enabled implementations, it is required to use extensions/additions to the mechanisms in [RFC 3264] as defined in [IETF-DRAFT-JSEP]
UNI-WRTCS-020	The WebRTC Signaling API SHALL provide an optional way to exchange data with the Extension for MSRP based sessions.		MSRP is terminated at the client with a protocol stack based on webRTC datachannel in agreement with the architecture specified in appendix D of [RCS5.3] and related 3GPP specifications.

4.11 Video Share UNI API requirements

References for Video Share: GSMA IR.74 [IR74] as endorsed by RCS.

4.11.1 Video Share use cases (informative)

To clarify the requirements in the next sections, the intended basic use cases of the Video Share API are:

1. API Originated: Sharing a recorded or stored video file from application to client.

The application acts as an originating client in a Video Share session. For instance, a music television station offers its customers to browse a catalogue of music videos, and stream them to clients using a “click to play” interaction. The application uses a video file as the source of the video stream of the Video Share.

Figure 6 illustrates a schematic flow. For option 1, the file is included as the body of the API request to create the Video Share session. This ensures that the video file is available when the video share session is accepted. The method to upload the media file to the repository in option 2 is out of the scope.

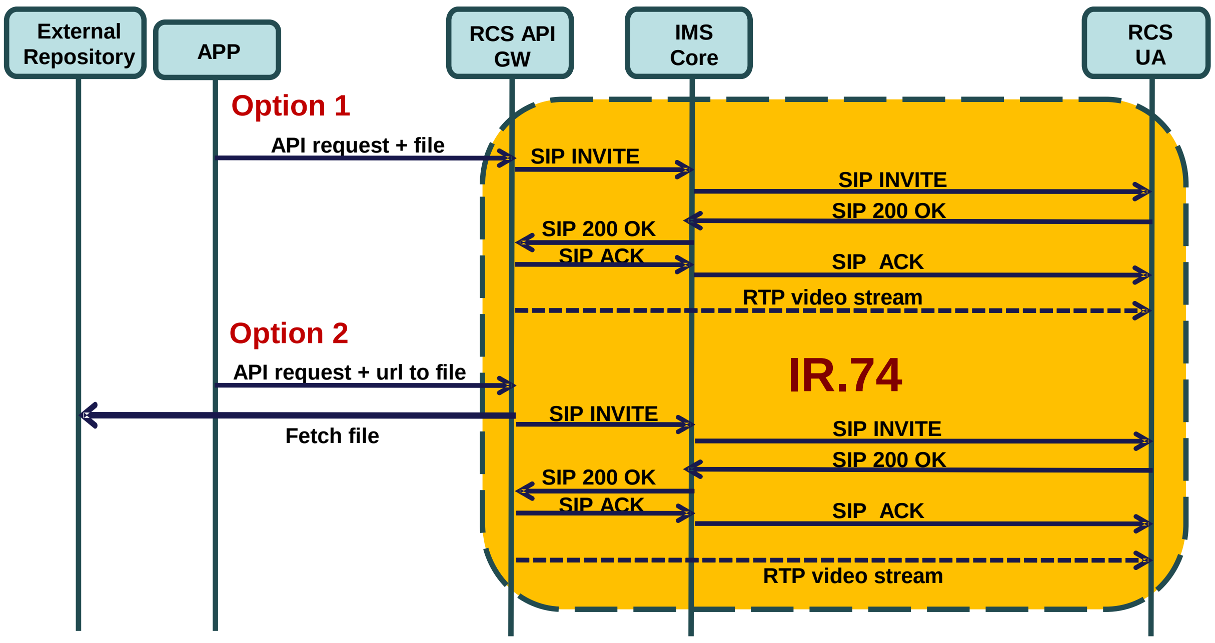


Figure 6: Schematic flow for Video Share Use Case 1

2. API Originated: Sharing real time video from application to client.

The application acts as an originating client in a Video Share session. For instance, the application streams video from a live video feed to clients.

The application creates a new Video Share session and announces to the API gateway which formats (i.e., transport protocol, codecs, etc.) it supports. The API gateway processes the list and selects one of the offered formats (i.e., transport protocol, codecs, etc.). The API gateway then makes a Video Share invitation to the IR.74 compliant client. When the client accepts the Video Share session, the API gateway sends a notification to the application using the notification channel indicating the chosen format and the media URL and/or access parameters, to which the application shall subsequently send the media.

The API will provide an open and extensible mechanism to signal the media formats (i.e., transport protocol, codecs, etc.), but the specification of the media protocols and connection/play mechanisms are out of the scope of this API specification (marked in green in Figure 7).

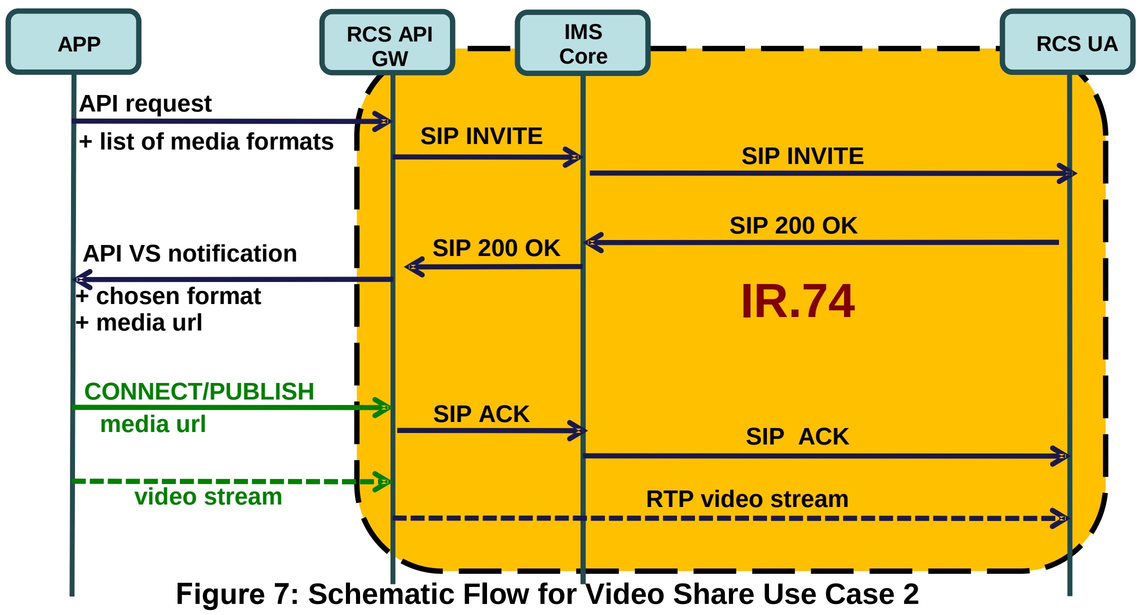


Figure 7: Schematic Flow for Video Share Use Case 2

3. API terminated: Sharing video from client to application

   The application acts as a terminating client in a Video Share session. For instance, it could allow a user to watch in real time, from a web browser, the video that was shared. Another example would be an application that records the shared video for later use.

   A summarized interaction would be as follows: The Video Share session is started by an IR.74 compliant handset. The API gateway receives the IR.74 invitation and notifies the application about it indicating a list of formats (i.e., transport protocol, codecs, etc...) in which the media can be made available.

   The application searches the list for the most suitable format according to the platform/software it is running and then accepts the Video Share session indicating the chosen format. In the response to this acceptance request, the gateway will return the URL and/or any other access parameters which the client needs to access the media.

   The API will provide an open and extensible signalling mechanism for codecs, formats, transports, etc., however, the specification of the media protocols and connection/play mechanisms are out of the scope of this API specification (marked in green in Figure 8).

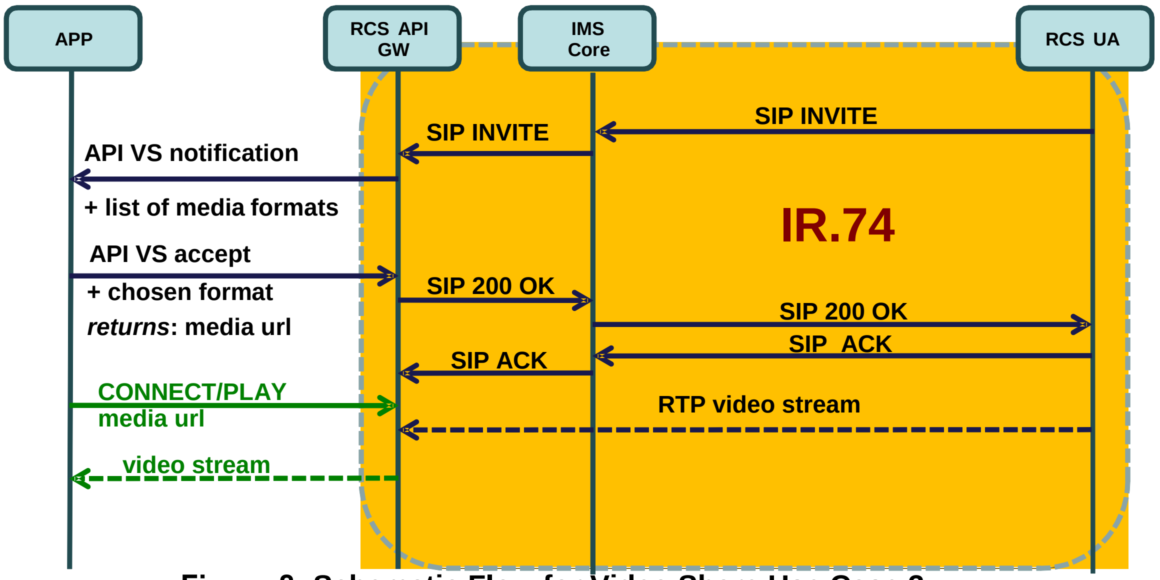


Figure 8: Schematic Flow for Video Share Use Case 3

More complicated use cases can be built based on these basic scenarios. Also note that IR.74 compliant clients can support these three use cases with no changes.

4.11.2 Video Share functionalities available to originating side

Label	Description	Required parameters	Comment

UNI-VSH-001	The Video Share API SHALL support initiating a Video Share session using a video file.	oauth_token={access-token}
		
		recipient={contactid}
		or
		call={callObjectID}
		
		formats={list of media formats}
			See use case 2 for more details about this requirement.
			
			Arguments need to contain at least either a reference to an existing call or a recipient.
			When the Video Share is established with the call ID, the API gateway will link the "initiate Video Share" request to the ongoing call.
			
			Video Share object instance is created and returned immediately to accommodate cancelling before alerting.
			
			The video file could be sent either in the body of the request (option 1) or via an URL to the media file (option 2). The application shall send the list of formats (i.e., transport protocol, codecs, etc.) that it supports.

UNI-VSH-001b	The Video Share API SHALL support initiating a Video Share session using real time video feed.	oauth_token={access-token}
		
		recipient={contactid}
		or
		call={callObjectID}
		
		formats={list of media formats}
			See use case 2 for more details about this requirement.
			
			Arguments need to contain at least either a reference to an existing call or a recipient.
			When the Video Share is established with the call ID, the API gateway will link the "initiate Video Share" request to the ongoing call.
			
			Video Share object instance is created and returned immediately to accommodate cancelling before alerting.
			
			The application shall send the list of formats (i.e., transport protocol, codecs, etc.) that it supports.

UNI-VSH-002	VOID	VOID	VOID
UNI-VSH-003	VOID	VOID	VOID
UNI-VSH-004	The Video Share API SHALL support cancelling a Video Share by the originating side.	oauth_token={access-token}	Use case: Application on originating side interrupts Video Share attempt. Only the user who created the invitation can cancel it, and it is offered only before the file transfer is accepted or rejected.
UNI-VSH-005	The Video Share API SHALL support notifications about Video Share ("alerting", "accepted", "ended", "declined", "failed")	If "accepted" the notification can include the following information: Choosen media format Media Url.	The final set of applicable notification types will be determined in the technical work phase. See "Common notification channel" for establishment of notification channel. If the video share session was initiated using a live video feed as indicated in the UNI-VSH-002 requirement, the APIs shall include the chosen format and media URL to which the application shall send the media in the "accepted" notification. See use case 2 for more details.

UNI-VSH-006	The Video Share API SHALL support ending Video Share by the originating side.	oauth_token={access-token}	Use case: Application on originating side stops Video Share. A SIP BYE is sent to the remote end.

4.11.3 Video Share functionality available to terminating side

Label	Description	Required parameters	Comment
UNI-VSH-007	The Video Share API SHALL support receiving a Video Share invitation.	Inviting contact or Reference to an ongoing call List of media formats	See use case 3 for more details on this requirement. The API gateway receives the Video Share session invitation, and notifies the application about it indicating a list of formats (i.e., transport protocol, codecs, etc.) in which the media can be made available.
UNI-VSH-008	VOID	VOID	VOID

UNI-VSH-009	The Video Share API SHALL support accepting a Video Share by the terminating side.	oauth_token={access-token}
		
		format={format}
		
		returns:
		media_url={media_url}
		parameters={param1,..}
			When the user accepts the Video Share session invitation, the application will search the list for the most suitable format according to the platform/software it is running and indicate the chosen format in the acceptance request.
			
			In the response to this acceptance request, the gateway will return the URL and/or any other access parameters which the client needs to access the media.

UNI-VSH-009b	The Video Share API SHALL support rejecting a Video Share by the terminating side.	oauth_token={access-token}	
UNI-VSH-010	The Video Share API SHALL support ending a Video Share by the terminating side.	oauth_token={access-token}	Use case: Application on terminating side ends Video Share. Triggers sending a BYE to the originating side.
UNI-VSH-011	The Video Share API SHALL support		The final set of applicable notification types will be

	notifications about "Video Share" ("ended", "cancelled", "failed") to the terminating side.		determined in the technical work phase. See "Common notification channel" for establishment of a notification channel.

4.12 Image Share UNI API requirements

References for Image Share: GSMA IR.79 [IR79] as endorsed by RCS.

4.12.1 Image Share use cases (informative)

To clarify the requirements in the next sections, the intended basic use cases of the Image Share API are:

1. API Originated: Sharing a file from application to client.

   The Image Share session is started by the application using the API. The application uses an image file as the source of the Image Share transfer. The image file can be either included in the initial API call or retrieved from an external repository. Method to upload the image file to the repository is outside of the scope of this document.

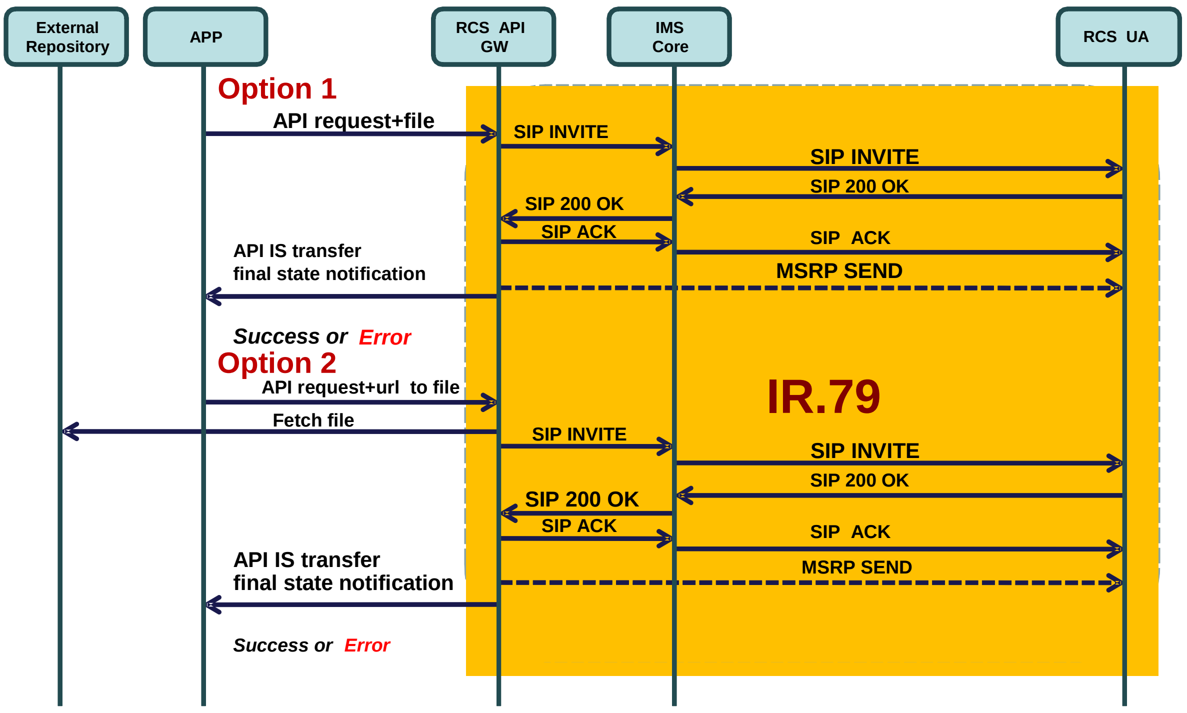


Figure 9: Schematic flow for Image Share Use Case 1

2. API Terminated: Sharing a file from application to client.

   The Image Share session is started by an IR.79 compliant client. The API gateway receives the IR.79 invitation, and notifies the application. If the application accepts the invitation, the IS will be established between the API gateway and the UA. When the Image Share session is correctly established, the application will be notified and given a URL in which the file can be downloaded.

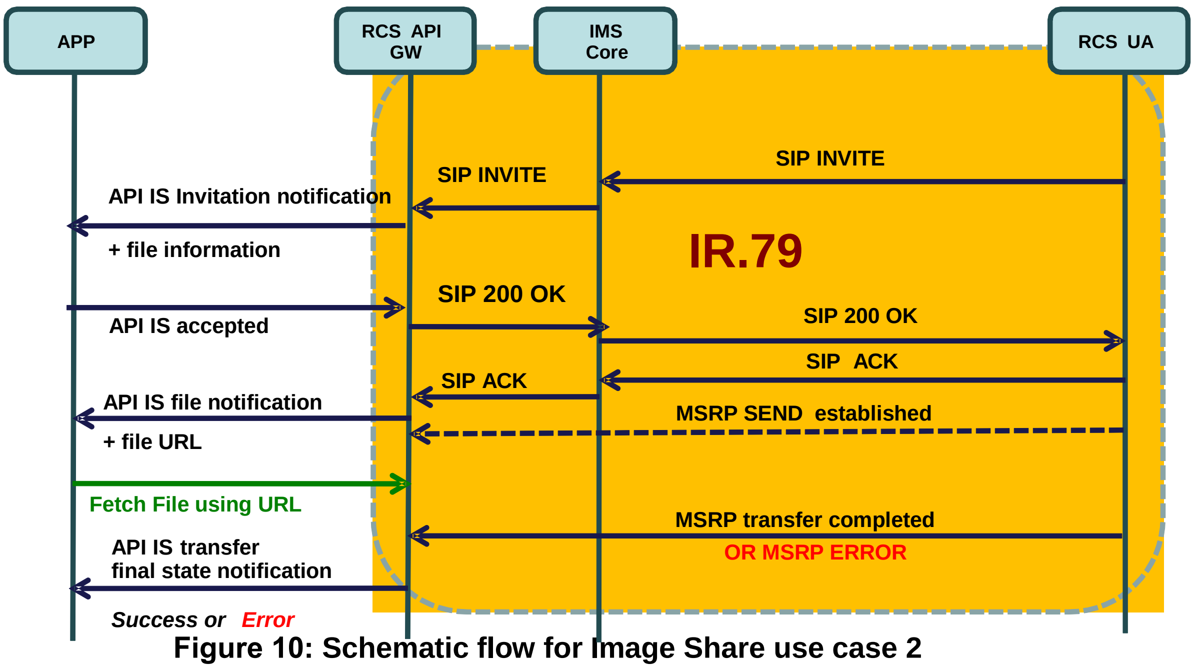


Figure 10: Schematic flow for Image Share use case 2

4.12.2 Image Share functionality available to originating side

Label	Description	Required parameters (not complete list)	Comment

UNI-ISH-001	The Image Share API SHALL support initiating a Image Share to a user.	oauth_token={access-token}
		recipient={contactid}
		call={callObjectID}
		
		url={url to the image file}
		or
		BODY{image file}
			Use case: Application on originating side initiates Image Share.
			Arguments need to contain at least either a reference to an existing call (callObjectId) for [IR79] Image Share or a Recipient for Image Share without call (i.e., using OMA IM File Transfer).
			
			The image file could be sent either in the body of the request (option 1) or sent via an URL to the image file (option 2)

UNI-ISH-002	VOID	VOID	VOID
UNI-ISH-003	VOID	VOID	VOID
UNI-ISH-004	The Image Share API SHALL support cancelling an Image Share by the originating side.	oauth_token={access-token}	Use case: Application on originating side interrupts Image Share attempt. It is offered only before the session is accepted.

UNI-ISH-005	The Image Share API SHALL support notifications about Image Share (“alerting”, “accepted”, “ended”, “declined”, “failed”)		The final set of applicable notification types will be determined in the technical work phase. See “Common notification channel” for establishment of notification channel.
UNI-ISH-006	The Image Share API SHALL support ending Image Share by the originating side.	oauth_token={access-token}	Use case: The application on originating side stops Image Share. A SIP BYE is sent to the remote end.

4.12.3 Image Share functionality available to terminating side

Label	Description	Required parameters (not complete list)	Comment
UNI-ISH-007	The Image Share API SHALL support receiving an Image Share invitation.	Inviting contact Reference to an ongoing call (for IR.79)	Use case: The application on the terminating side receives an Image Share invitation. See “Common notification channel” for establishment of notification channel.
UNI-ISH-008	VOID	VOID	VOID
UNI-ISH-009	The Image Share API SHALL support accepting or rejecting an Image Share by the terminating side.	oauth_token={access-token}	Use case: The application on the terminating side accepts an Image Share session. This triggers sending a SIP 200 (if accepted) or a suitable rejection cause (if declined) to the originating side.
UNI-ISH-010	The Image Share API SHALL support ending an Image Share by the terminating side.	oauth_token={access-token}	Use case: The application on terminating side ends an Image Share session. This triggers sending BYE to the originating side.
UNI-ISH-011	The Image Share API SHALL support final state notifications about the Image Share MSRP transfer session (“success”, “abort” and “error”).		The final set of applicable notification types will be determined in the technical work phase. See “Common notification channel” for establishment of notification channel.
UNI-ISH-012	The Image Share API SHALL support notifications indicating that	url={img url}	The gateway will send this notification to the client with URL to download the image.

	the image share content is available for download		The server which the URL is pointed to SHALL be ready to start downloading when the notification is sent. It is up to the implementation to decide if this is sent when the first chunks of MSRP data are received and allow to simultaneously receiving of data from the MSRP session and HTTP downloading; or if it waits for the MSRP session to be completed and only allow the download to be started only when the whole file has been received. In any case the notification SHALL be sent before the final state notification is sent.

4.12.4 Capability Query UNI API requirements

Refer to section 4.5.3 (Services capabilities).

4.13 Location Pull

The Location PULL API provides a RESTful interface allowing an RCS application to query the location of an RCS user's mobile devices, which are connected to a mobile operator network, using network based positioning method.

The Location PULL API requirement herein is based on the UNI specification of RCS 5.3; therefore, additional parameters or information available from the OMA Terminal Location API are outside the scope of this specification.

References: [RCS5.3] Section 3.10.4.2 Geolocation pull

Label	Description	Required parameters (not complete list)	Comment

UNI-LPU-001	The location PULL API SHALL support the request to pull the geolocation coordinate (x,y) of a target mobile device registered in cellular network.
	
	The Location PULL API
		oauth_token={access-token}
		contact={contactId}
		requested_accuracy={requested-accuracy}
			If the positioning attempt is successful, Longitude and Latitude will be provided as the (x,y) coordinate of the geographic position. The application may optionally use other available contactID attribute (ACR) to request pulling the location of a given contact in the address book.

	SHALL support the request of positioning accuracy in meters.		The requested accuracy of the positioning result is expressed in meters. Typically, a request for higher positioning accuracy may take longer to retrieve than a request for coarse accuracy.

4.14 RCS Personal Network Blacklists basic operations

Personal Network Blacklists (PNB) are used to block incoming/outgoing service flow received/initiated by specified senders or recipients. Three RCS services are concerned: Messaging, 1-to-1 Chat and File transfer. PNB functionality (incl lists) are defined in [RCS5.3] chapter 2.15 -- “Personal Network Blacklists (PNB)”.

It should be noted that this API is reserved for “Trusted” Applications as it exposes user identities and their management.

The trusted status of applications is managed by the service provider

Label	Description	Required parameters (not complete list)	Comment

UNI-PNB-001	The Network PNB API SHALL support retrieval of PNB contacts.	oauth_token={access-token}
		list={}
			The target list of the request is empty or one or more of the six PNB lists:
			rcs_pnb_chat_blockedusers,
			rcs_pnb_ft_blockedusers,
			rcs_pnb_standalone_blockedusers,
			rcs_pnb_outchat_blockedusers,
			rcs_pnb_outft_blockedusers,
			rcs_pnb_outstandalone_blockedusers
			
			If no PNB list is provided, it means all six PNB lists are requested.
			
			The answer amounts to retrieval of the set of contacts in the list. A contact identity can be a MSISDN or a SIP URL.

UNI-PNB-002	The Network PNB API SHALL support update of PNB list (i.e., addition of a new contact as well as deletion of an existing contact from a particular	oauth_token={access-token}
		action={add,delete}
		list={}
		contact={}
			Add new contact(s) (i.e., MSISDN or SIP URL) to a PNB list(s) or delete existing contact(s) (i.e., MSISDN or SIP URL) from PNB list(s)
			The target list of the request is empty or one or more of the six PNB lists:

	PNB list (s))		rcs_pnb_chat_blockedusers, rcs_pnb_ft_blockedusers, rcs_pnb_standalone_blockedusers, rcs_pnb_outchat_blockedusers, rcs_pnb_outft_blockedusers, rcs_pnb_outstandalone_blockedusers If no PNB list is provided, it means all six PNB lists.

UNI-PNB-003	The Network PNB API SHALL support delivery of notifications when updates to the PNB lists are done	list={listid}
		contact={contactid}	

4.15 RCS Network Message Storage UNI API requirements

The Network Message Storage (NMS) is a repository for all message/files exchanges for a RCS user. It is also used to synchronize the conversation history to all devices of a RCS user. The RCS Network Message Storage is based on OMA CPM Message Storage [OMACPM-MS] specification.

The NMS APIs provide the ability for an application to retrieve a RCS user's conversation histories stored in the network and only exposed to "Trusted" applications.

The trusted status of applications is managed by the service provider. It is understood that the {access-token} in the oauth_token parameter uniquely identifies the RCS user for whom these operations are invoked.

Note that in the requirements below, in those cases when folders and messages are modelled as REST resources, their ID will be a complete URL which will contain the information stated in the requirements below.

Label	Description	Required parameters (not complete list)	Comment

UNI-NMS-001	The NMS API SHALL support retrieval of folders contained in the root folder	oauth_token={access-token}
		
		returns:
		folders={foldernames,…}
			Returns all folder names under the root folder, or "empty" if no folder exists.

UNI-NMS-002	The NMS API SHALL support retrieval of information (including all sub-folders, messages, transferred files,	oauth_token={access-token}
		
		folder={foldername}
		
		optional search criteria in request:
		start_datetime={UTC datetime}
			Returns all information (including sub-folders, messages, transferred files and objects) stored under this specific folder, or "empty" if no information exists.
			Folders are returned as folder names, while messages,

	objects etc.). Search criteria may be combined and applied based on: a specific folder; date/time window; sender; recipient; or key word.	end_datetime={UTC datetime} sender_address={parameter} recipient_address={parameter} key_word={parameter}
		
		returns: info={foldernames|object UIDs …}
			transferred files, and other objects are returned as objectUID values, with the format: "<foldername>/<UID>".
			The end_datetime must be greater or equal to the start_datetime UTC time value.
			
			The key_word parameter could be free-text or a "regular expression" that allows more accurate control of how the search is applied.

UNI-NMS-003	The NMS API SHALL support retrieval of a specific message(s) with message UID(s)	oauth_token={access-token}
		
		Message_UID={UID,…}
			If there is metadata associated with the retrieved message, it should be returned with the message.

UNI-NMS-004	The NMS API SHALL support changing message status (flags) of a specific message(s) with message UID(s)	oauth_token={access-token}
		
		Message_UID={UID,…} Message_status={UID|flags,…) ….
			The returned information is a list of tuples consisting of the message UID and corresponding flags.
			The flags are defined in IETF RFC 3501

UNI-NMS-005	The NMS API SHALL support management of folder status (flags) of a specific folder with foldername	oauth_token={access-token}
		
		folder={foldername} folder_status={flags)
			The returned information is a tuple consisting of the foldername and the corresponding flags.
			The flags of a folder should be the same as defined for a message in IETF RFC 3501, but at least the \Flagged flag is required.

4.16 RCS Extension to Extension API requirements

This section defines the requirements for the RCS Extension to Extension service.

RCS Extension to Extension service is described in section 3.12.4.2.2 of [RCS5.3]. There are two types of media types supported on the RCS UNI for this Extension to Extension service:

- based on MSRP (see section 3.12.4.2.2.1 of [RCS5.2]), or
- based on RTP (see section 3.12.4.2.2.2 of [RCS5.3]).

As initiating media (MSRP or RTP) sessions is always similar, the Extensions can use an Extension API Gateway to make their development easier:

- For MSRP media, the API gateway should terminate the MSRP itself and just give some easy API calls manage sessions and exchange data. This way, the Extension doesn't have to worry about implementing anything about the MSRP protocol. See Figure 1.

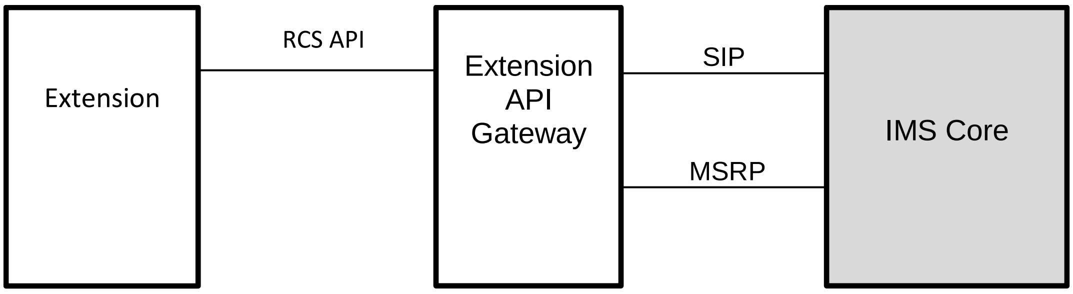


Figure 1: Schematic view for an extension using MSRP

- For RTP media, the API gateway should behave as a UNI Extension towards the IMS Core, so it should implement the Extension UNI specification. This means it should use SIP and RTP towards the IMS and then provide a way (or many ways) to exchange real time media with the Extension. Note: the media flow does not necessarily go through the gateway. See Figure 2 for an schematic view of a possible implementation.

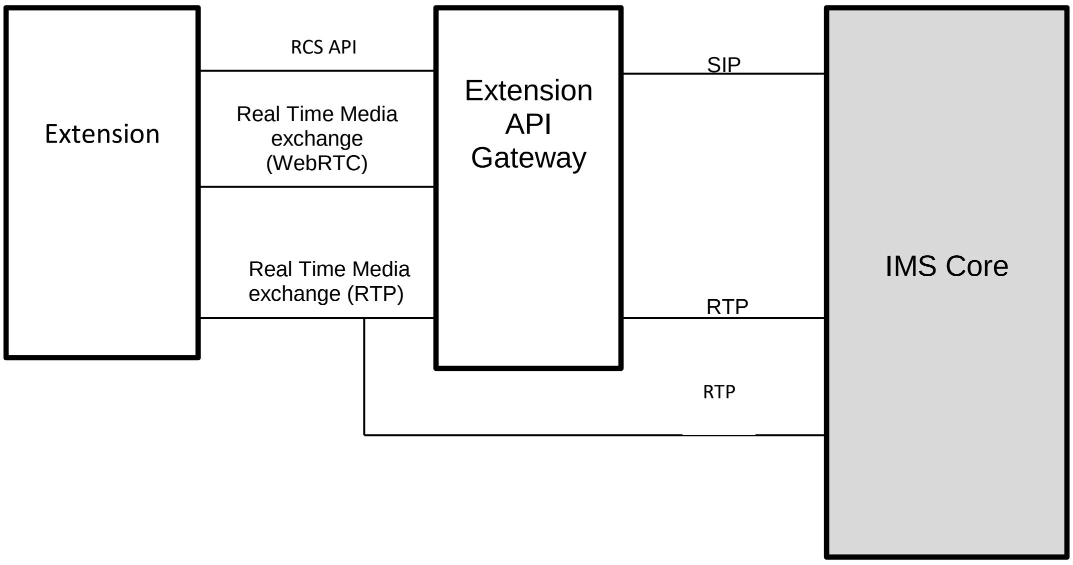

Extension
RCS API
Real Time Media exchange (WebRTC)
Real Time Media exchange (RTP)
Extension API Gateway
SIP
RTP
RTP
IMS Core

Figure 2: Schematic view for an extension using RTP

4.16.1 Extension to Extension functionality available to originating side

Label	Description	Required parameters (not complete list)	Comment
UNI-E2E-001	The Extension to Extension API shall support initiating an Extension to Extension session. The API shall allow indicating the IARI value	Mandatory: • oauth_token={access-token} • recipient={contactId | E.164 number} • IARI value Optional : • Media transport to be	The recipient is required to be the application corresponding to the IARI. NOTE 1: in protocol terms mandating that the IARI is registered is accomplished by adding both the "explicit" and "require" parameters to the Accept-Contact header carrying

	identifying the application using the API. The API shall support specifying if MSRP or RTP shall be used in the IMS RCS infrastructure.	used (MSRP or RTP). Default = MSRP return {callId}	the IARI of the corresponding SIP request as per RFC 3841. NOTE2: Only one to one session is supported. See [RCS5.3] section 3.12.4.2.2 Extensions – Technical Realisation.

UNI-E2E-002	The Extension API shall support cancellation of an Extension to Extension session setup.	oauth_token={accesstoken}
		callId={callId}
		return {success/failed}
			The cancellation of call setup is only possible while the call is not successfully established.

UNI-E2E-003	The Extension API shall support notification about the Extension to Extension setup phase.	The notification supported may be at least "ringing" "accepted", "declined", "no_answer" "not_reachable"	

4.16.2 Extension to Extension functionality available to terminating side

Label	Description	Required parameters (not complete list)	Comment
UNI-E2E-005	The Extension to Extension API shall support notification about new incoming Extension to Extension session invitation. This notification shall contain an IARI value identifying the target third party.	Information about the originator, callID, and IARI	
UNI-E2E-006	The Extension to	oauth_token={accesstoken}	

	Extension API shall support accepting the Extension to Extension session by the terminating side.	callId={callId}	

UNI-E2E-007	The Extension to Extension API shall support rejecting the Extension to Extension session by the terminating side.	oauth_token={accesstoken}
		callId={callId}	

4.16.3 Extension to Extension functionality available to originating and terminating side

Label	Description	Required parameters (not complete list)	Comment

UNI-E2E-008	The Extension to Extension API shall support terminating the Extension to Extension session.	oauth_token={accesstoken}
		callId={callId}	

4.16.4 Media

Label	Description	Required parameters (not complete list)	Comment
UNI-E2E-009	The Extension to Extension API SHALL support sending and receiving real time media towards the IMS via the NetAPI gateway.		The API Gateway must behave as a client in the UNI side. It must send RTP as a normal Extension would do without API.
UNI-E2E-010	The Extension to Extension API for real time media SHALL support the configuration of different		The API Gateway must have a way to configure the RTP profiles that are sent in the SDP to the UNI.

	profiles for RTP transmission towards the IMS.		
UNI-E2E-011	The Extension to Extension API for real time media SHALL support media link at least based on RTP or WebRTC to exchange real time media exchange with the Extension.		The API Gateway may or may not be the endpoint communication with the Extension.
UNI-E2E-012	For the Extension to Extension API, the netAPI gateway shall support sending and receiving MSRP media towards the IMS.		
UNI-E2E-013	The Extension to Extension API shall support a way to exchange data with the Extension for MSRP based sessions.		MSRP is terminated at the gateway level. The client using NetAPI is not required to embed an MSRP stack. The client could receive the data received in MSRP through the common notification channel (e.g. websocket or long polling). The client could send data through a dedicated REST API.

5 Service Provider / Chatbot Platform (SPCP) API requirements

5.1 General

There are no requirements for API functionality involving group conversations at this time.

The APIs for Chatbot Platform / Aggregator interaction with a Service Provider API GW shall extend existing OMA APIs as far as possible (where the base functionality exists).

It shall be possible for the Chatbot Platform to send requests (API calls) to the Service Provider API GW.

It shall be possible for the Chatbot Platform to receive asynchronous events from the Service Provider API GW.

It is Service Provider policy on how to configure the URL for the Chatbot Platform in the API GW and how to expose the URL for the API GW to the Chatbot Platform.

All callback (HTTP POST) requests will contain the callback URL as the Request-URI.

5.2 Authorization Framework

5.2.1 Introduction (informative)

The following tables show the functional requirements for authorization of the Chatbot Platform to the Service Provider network.

5.2.2 General requirements

Label	Description	Required parameters (not complete list)	Comment
SPCP-AUT-001	For all APIs consider whether a Server2Server model is used, and how the Auth token is obtained by the Chatbot Platform from the issuing authority	oauth_token = {access-token} resultofcommand: {on, off, failure, etc}	Could be one key used between the Chatbot Platform and Service Provider for all chatbot traffic, leaving individual authorization per chatbot to the chatbot / Chatbot Platform API.

5.3 Capability Management SPCP API Requirements

5.3.1 Capability Discovery

This API can be mapped to different capability management mechanisms in the underlying network, such as SIP OPTIONS or Presence.

The following table describes the Service Provider / Chatbot Platform API requirements for the capability discovery:

Label	Description	Required parameters (not complete list)	Comment

SPCP-CPD-001	The Capability Discovery API SHALL allow a Chatbot Platform to query the service capabilities of a certain user.	oauth_token = {access-token}
		chatbot querying = {SIP URI}
		chatbot capabilities = { }
		target user={token or MSISDN}
			Return value shall consist of a (possibly empty) list of capabilities for the target user.
			
			The chatbot app version capability with its list of app versions supported needs to be included as part of the chatbot capabilities. See section 3.6.2.2 of [RCC07] for more information on the chatbot application version which groups together a set of chatbot rich cards.
			
			NOTE: The refreshing of the capabilities exposed by the gateway is subject to Service Provider policy, for example, to avoid abuse or impact in network load. The Chatbot Platform is responsible for avoiding to send too many capability requests.

SPCP-CPD-002	The Capability Discovery API SHALL support receiving real time capability requests from the network and forwarding them to the Chatbot Platform.	oauth_token = {access-token}
		user querying = {token or MSISDN}
		user capabilities = { }
		target chatbot = {SIP URI}
			The Chatbot Platform should answer any incoming user capability request (e.g. OPTIONS received from remote user) returning only the feature tags related to the enabled capabilities.
			The chatbot app version capability with its list of app versions supported needs to be included as part of the chatbot capabilities. See section 3.6.2.2 of [RCC07] for more information on the chatbot application version which groups together a set of chatbot rich cards.
			See section 3.6.5 of [RCC07] on privacy protection for more information on the token used to hide a user's MSISDN from a chatbot.

SPCP-CPD-003	The Capability Discovery API SHALL allow a Chatbot Platform to reply to real time capability requests with current capabilities.	oauth_token = {access-token}
		chatbot answering = {SIP URI}
		chatbot capabilities = { }
		target user = {token or MSISDN}
			Mechanism to be supported is up to Service Provider policy.
			Applies to Capability Discovery based in SIP OPTIONS.
			The chatbot app version capability with its list of app versions supported needs to be included as part of the chatbot capabilities.

5.4 Service Provider / Chatbot Platform API requirements

5.4.1 Introduction (informative)

The following tables show the functional requirements for the one-to-one chat API between a Service Provider network and a Chatbot Platform / Aggregator.

5.4.2 One-to-One Chatbot Session API Requirements, including Revoke

Label	Description	Required parameters (not complete list)	Comment

SPCP-MSG-001	The Messaging API SHALL support sending messages from Chatbot Platform to the API GW.	oauth_token = {access-token}
		sending chatbot = {MSISDN or SIP URI}
		recipient = {target user(s) SIP URI or MSISDN}
		chatbot app version = {list of app versions}
		traffic type = { }
		message-id = {message-id}
		requested disposition notification = {“delivered”, “displayed”}
		anonymized = {no, tokenused}
		{content}
			Message content can be text, XML or JSON defined content types as per section 3.2 and section 3.6.10 of [RCC07].
			Bearer service selection SHALL be Chat. (SMS/MMS/Standalone Messaging shall not be supported).
			Messages from the Chatbot Platform shall include:
			- the chatbot application version as per section 3.6.2.2 of [RCC07]
			- the traffic type identifier, which may be empty or one of "advertisement", "payment", "premium", "subscription", "plugin", or any other value to be defined, as per section 3.6.7 of [RCC07].
			If the Chatbot Platform Anonymization Function used a token when communicating with the chatbot on behalf of the recipient, the anonymized parameter shall be set to “tokenused”. Otherwise it shall be set to “no”.
			• NOTE: If the AF is in the operator’s network and a token is used for the recipient in this request, the Chatbot Platform should set the anonymized parameter to “no”.

SPCP-MSG-002	The Messaging API SHALL support receiving messages from API GW to Chatbot Platform. The Messaging API SHALL indicate whether the Chatbot Platform shall	oauth_token = {access-token}
		sending user = {token or MSISDN, or both}
		recipient chatbot = {SIP URI}
			Same as for SPCP-MSG-001
			Sending user shall have two values when anonymize is set to tokenlink and when the AF is in the operator’s network. In all other cases, it shall have one value.

	anonymize messages or not.	chatbot app version = {list of app versions }
		traffic type = { }
		message-id = {message-id}
		requested disposition notification = {“delivered”, “displayed”}
		anonymize = {no, usetoken, tokenlink}
		{content}
			The anonymize parameter shall be used as follows:
			- “no”: used if the Chatbot Platform AF is not required to do anything. The Anonymization Function may or may not have already been invoked;
			- “usetoken”: used if anonymization is requested;
			- “tokenlink”: used if the Chatbot Platform shall provide both the user’s identity and the token to the chatbot:
			o If the sending user parameter already has two values (one token and one MSISDN), the Chatbot Platform shall provide those two values to the chatbot;
			o If the sending user parameter only has one value, the Chatbot Platform shall take the token/MSISDN pair from its own local AF;
			If the anonymize parameter is set to “usetoken” and the Chatbot Platform cannot provide anonymization, it shall return an error to the API call.

SPCP-MSG-003	The Messaging API SHALL support receiving of the message disposition notifications (“delivered”, “displayed”) from API GW to Chatbot Platform. The Messaging API SHALL indicate whether the Chatbot Platform shall anonymize messages or not.	oauth_token = {access-token}
		sending user = {token or MSISDN, or both}
		recipient chatbot = {SIP URI}
		message-id = {message-id}
		disposition notification = {“delivered”, “displayed”}
		anonymize = {no, usetoken, tokenlink}
			The message delivery and display notification are requested according to Service Provider policies, when a message is sent on API GW.
			Sending user shall have two values when anonymize is set to tokenlink and when the AF is in the operator’s network. In all other cases, it shall have one value.
			The message-id parameter value shall be the one sent in the message from the Chatbot Platform to the API GW.
			The anonymize parameter shall be used as follows:
			- ”no”: used if the Chatbot Platform AF is not required to do anything. The Anonymization Function may or may not have already been invoked;

			- "usetoken": used if anonymization is requested; - "tokenlink": used if the Chatbot Platform shall provide both the user's identity and the token to the chatbot; o If the sending user parameter already has two values (one token and one MSISDN), the Chatbot Platform shall provide those two values to the chatbot; o If the sending user parameter only has one value, the Chatbot Platform shall take the token/MSISDN pair from its own local AF; If the anonymize parameter is set to "usetoken" and the Chatbot Platform cannot provide anonymization, it shall return an error to the API call.

SPCP-MSG-004	The Messaging API SHALL support sending of the message disposition notifications ("delivered", "displayed") from Chatbot Platform to API GW.	oauth_token = {access-token}
		sending chatbot = {MSISDN or SIP URI}
		recipient = {target user(s) SIP URI or MSISDN or token}
		message id = {message-id}
		disposition notification = {"delivered", "displayed"}
		anonymized = {no, tokenused }
			The message-id parameter value shall be the one received in the incoming message from the API GW to the Chatbot Platform.
			If the Chatbot Platform Anonymization Function used a token when communicating with the chatbot on behalf of the recipient, the anonymized parameter shall be set to "tokenused". Otherwise it shall be set to "no".
			NOTE: If the AF is in the operator's network and a token is used for the recipient in this request, the Chatbot Platform should set the anonymized parameter to "no".
			This operation will be allowed only if the original message included the requested notification request (e.g., "delivered" or "displayed").

SPCP-MSG-005	The Messaging API SHALL support sending of "isComposing" from Chatbot Platform to the API GW.	oauth_token = {access-token}
		sending chatbot = {MSISDN or SIP URI}
		recipient = {target user(s) SIP URI or MSISDN or token}
			Use case: The chatbot sends "isComposing" which indicates that a chatbot is currently composing a message.
			If the Chatbot Platform Anonymization Function used a token when communicating with the chatbot on

		isComposing = "active"/"idle", "timeout=xx"" …
		anonymized = {no, tokenused}
			behalf of the recipient, the anonymized parameter shall be set to "tokenused". Otherwise it shall be set to "no".
			NOTE 1: If the AF is in the operator's network and a token is used for the recipient in this request, the Chatbot Platform should set the anonymized parameter to "no".NOTE 2: this shall only apply when a chat session has been established by the API GW.

SPCP-MSG-006	The Messaging API SHALL support sending the "isComposing" message from API GW to the Chatbot Platform.	oauth_token = {access-token}
		sending user = {token or MSISDN, or both}
		recipient chatbot = {SIP URI}
		isComposing = "active"/"idle", "timeout=xx"" …
		anonymize = {no, usetoken, tokenlink}
			Use case: the chatbot is provided with an indication that a user is currently composing a message.
			Sending user shall have two values when anonymize is set to tokenlink and when the AF is in the operator's network. In all other cases, it shall have one value.
			The anonymize parameter shall be used as follows:
			- "no": used if the Chatbot Platform AF is not required to do anything. The Anonymization Function may or may not have already been invoked;
			- "usetoken": used if anonymization is requested;
			- "tokenlink": used if the Chatbot Platform shall provide both the user's identity and the token to the chatbot;
			o If the sending user parameter already has two values (one token and one MSISDN), the Chatbot Platform shall provide those two values to the chatbot;
			o If the sending user parameter only has one value, the Chatbot Platform shall take the token/MSISDN pair from its own local AF;
			If the anonymize parameter is set to "usetoken" and the Chatbot Platform cannot provide anonymization, it shall return an error to the API call.
			NOTE: this shall only apply when a chat

			session has been established by the API GW.

SPCP-MSG-007	The Messaging API SHALL support the ability of a Chatbot Platform to REVOKE a message that is currently in an undelivered state on the Service Provider network equipment.	oauth_token = {access-token}
		sending chatbot = {MSISDN or SIP URI}
		recipient = {target user(s) SIP URI or MSISDN or token}
		message id = {message-id}
		anonymized = {no, tokenused}
			Sending a message from the chatbot shall specify that revoke may be needed. See section 3.2.3.8.2 of [RCC07]. The message-id parameter shall be used to identify the message to be REVOKED as per section 3.2.3.8.2.4 of [RCC07].
			The status of the REVOKE action shall be received by the sending chatbot platform as per section 3.2.3.8.2.4 of [RCC07].
			If the Chatbot Platform Anonymization Function used a token when communicating with the chatbot on behalf of the recipient, the anonymized parameter shall be set to “tokenused”. Otherwise it shall be set to “no”.
			NOTE: If the AF is in the operator’s network and a token is used for the recipient in this request, the Chatbot Platform should set the anonymized parameter to “no”.

5.5 Privacy Management (Alias Function) API requirements

5.5.1 Introduction (informative)

The following tables show the functional requirements for the Privacy Management (Anonymization Function) API.

Used when the Anonymization Function is deployed in the Chatbot Platform.

5.5.2 Service Provider network to Chatbot Platform

Label	Description	Required parameters	Comment
SPCP-PRI-001	ability for the API GW to delete the anonymization token in the Chatbot Platform. This is a control API to the Chatbot Platform from the Service Provider API GW	target chatbot: {SIP URI} TokenDelete: User Identity (MSISDN or SIP URI)	The token delete command to be mapped to an API is defined in section 3.6.5 of [RCC07]. Used when the Anonymization Function is deployed in the Chatbot Platform

5.6 Spam Report Function API requirements

5.6.1 Introduction (informative)

The following table shows the functional requirements for the Spam Report Function API.

Used when the spam report function is deployed in the Service Provider network.

5.6.2 Service Provider network to Chatbot Platform

Label	Description	Required parameters	Comment
SPCP-SPR-001	spam report message from API GW to the Chatbot Platform	User identity (MSISDN or SIP URI) Target chatbot = {SIP URI (only one, not a list)}, List of Message-ID values (zero up to 10) of messages being reported as spam received by the user from the chatbot	The send spam report message command to be mapped to an API is defined in section 3.6.6 of [RCC07]. The spam report message is addressed to the chatbot. It is up to Chatbot Platform provider whether to report it to the chatbot

Annex A RCS API Authentication and Authorisation – Use Cases

A.1 Overview

Use case examples and flows for detailing requirements regarding:

- Application Registration (Developer)
- Application Usage (End-User)
- Application Authentication
- User Authorisation
- Application Authentication control

Using MSISDN for user authentication and OAuth for application authorisation

Type of application: network-side web application, illustrated with two variant, both of them following the OAuth Authorisation Code flow.

(A) Generic Web App, aggregating RCS (and other) resources

- The developer creates and deploys an RCS Set Tagline web app on e.g. his web site (in practice, the Web App would offer more RCS primitives than just “Set Tagline”)
- The end-user has an account on an RCS Set Tagline web app
- The end-user accesses to the RCS Set Tagline web app from any browser

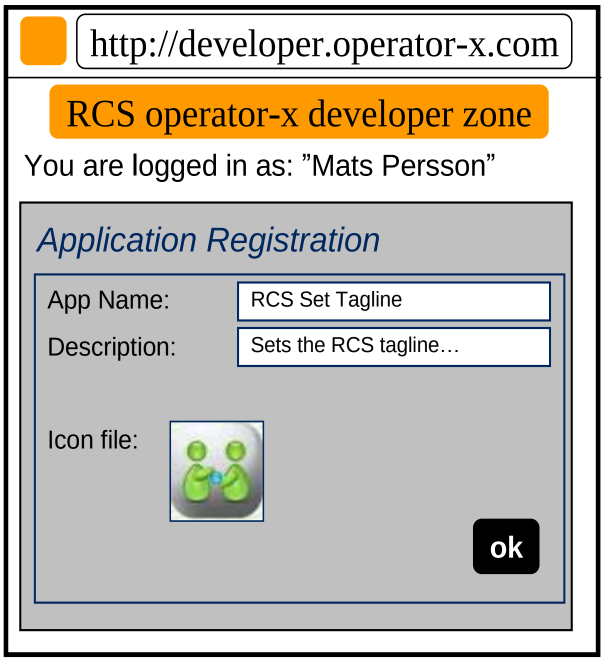
(B) “App on Facebook”

- The developer creates and hosts an RCS Set Tagline App on e.g. his web site
- Facebook imports and publishes the RCS Set Tagline App as a “Facebook App”
- The end-user has an account on Facebook
- The end-user accesses (the App on) Facebook from any browser

A.1.1 Application registration – Developer view

A.1.1.1 (A) General

- The developer has developed an RCS Set Tagline Web App, offering to RCS users the ability to set their RCS tagline from a Web browser,
- The developer has established a developer-account with operator-x (as in example).
- The developer may also have a RCS subscription at the operator that may be linked to the developer account (optional).
- The developer registers the application in the operator’s portal.
- Provided information: Application Name, Description.


http://developer.operator-x.com

RCS operator-x developer zone

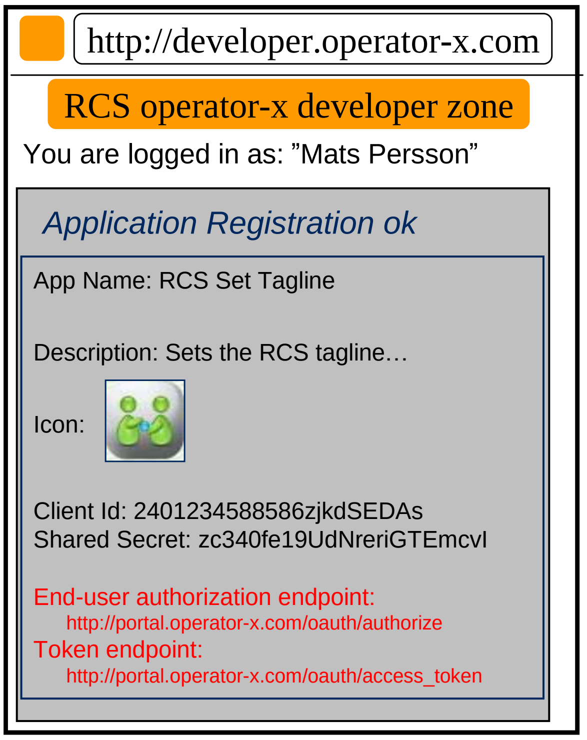
You are logged in as: ”Mats Persson”

Application Registration

App Name:	RCS Set Tagline
Description:	Sets the RCS tagline…
Icon file:	

Figure 3: Generic Web App – Developer application registration panel

- The portal generates unique Application credentials (Client Identifier, Shared Secret) to be used to identify and authenticate the application when used.
- The portal also provides the endpoint URLs specific to the operator’s Authorisation Server (end-user authorisation endpoint and token endpoint).
- The application is then deployed in the target environment (e.g., developer’s website or Facebook).
- Application credentials and endpoint URLs are stored as per operator with whom the developer has registered the application.
- The developer has to undergo the above registration procedure with all operators with whom the developer wants to engage the application.


http://developer.operator-x.com

RCS operator-x developer zone

You are logged in as: ”Mats Persson”

Application Registration ok

App Name: RCS Set Tagline

Description: Sets the RCS tagline…

Icon:

Client Id: 2401234588586zjkdSEDAs
Shared Secret: zc340fe19UdNreriGTEmcvl

End-user authorization endpoint:
http://portal.operator-x.com/oauth/authorize
Token endpoint:
http://portal.operator-x.com/oauth/access_token

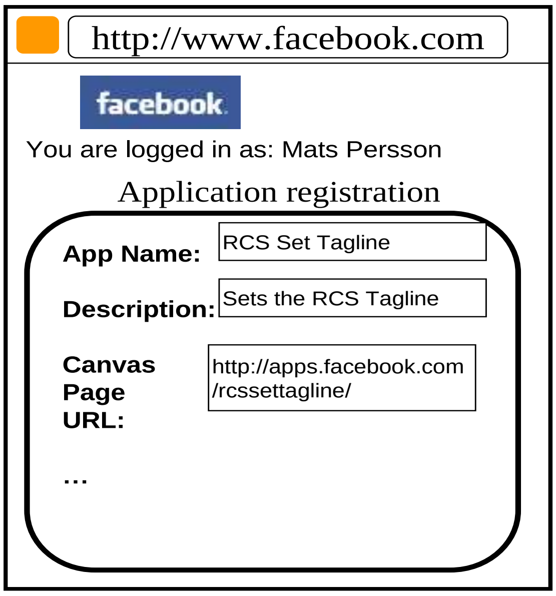
Figure 4: Generic Web App – Application registration by developer completed

A.1.1.2 (B) Additional step in case of Facebook variant

- The developer wants to publish their ”RCS Set Tag Line” web app as an ”App on Facebook”.
- The developer logs in to their Facebook account.
- The developer provides in the Facebook registration form information such as the “Canvas Callback URL”, pointing the “start” resource of his web app that is hosted on his web site.

Note: Facebook will assign to this app some OAuth 2.0 credentials; however they are used only when the web app calls Facebook APIs (i.e., access to photos, wall, etc.). Not to be confused with the OAuth credentials used by the web app to call RCS APIs).

See http://developers.facebook.com/docs/guides/canvas/


http://www.facebook.com

facebook

You are logged in as: Mats Persson

Application registration

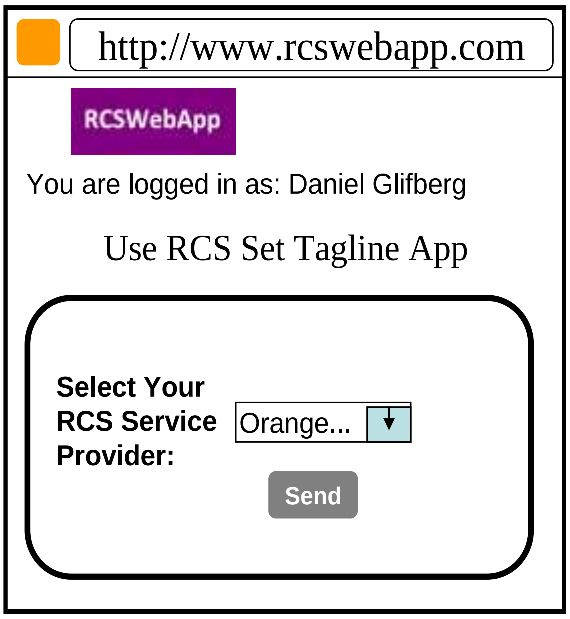
App Name:	RCS Set Tagline
Description:	Sets the RCS Tagline
Canvas Page URL:	http://apps.facebook.com/rcssettagline/
…	

Figure 5: Facebook App – Developer application registration panel

A.1.2 Application authorisation – User view

A.1.2.1 Application discovery - (A): Generic Web App variant

- An RCS user has discovered the “RCS Set Tagline” web app on the web.
- The process of discovery is out of scope. For example, it could be accomplished through an “RCS Application Store” portal setup by the Service Provider.
- The user may have to create an account on this app portal to use the application (not in scope of RCS).
- The user must authorize the application to access to his RCS resources on his account and indicate his/her (RCS) Service Provider
- The latter for the application to select the right operator portal to connect to (if supporting multiple operators)
- When pressing the “send” button, the user’s browser is re-directed to the user’s operator portal.
- The endpoint URL to the operator portal was obtained from app registration.
- In the authorisation request, the application provides Application ID, target RCS resources (scope), and Redirect URI.



http://www.rcswebapp.com

You are logged in as: Daniel Glifberg

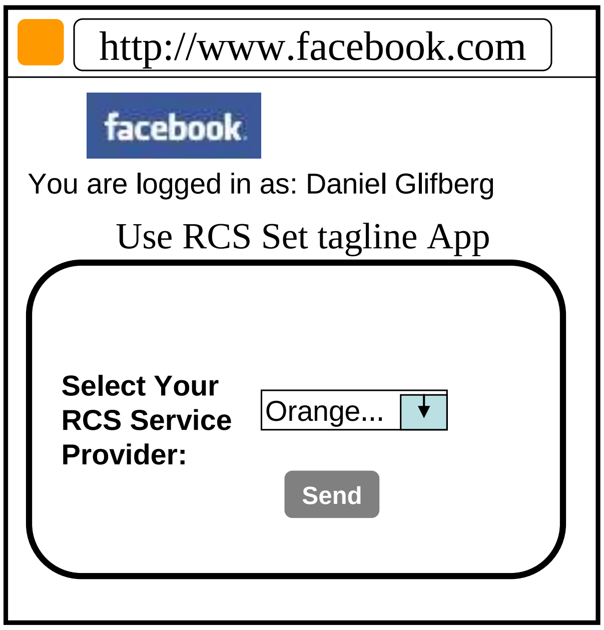
Use RCS Set Tagline App

Select Your RCS Service Provider: Orange...

Figure 6: Generic Web App – End user application management panel

A.1.2.2 Application discovery - (B): Facebook variant

- A (Facebook) user has discovered the “RCS Set Tagline” application.
- Following app selection in Facebook, the user must authorize the application to Set Tag Line on his account, and indicate his/her (RCS) Service Provider.
- The latter for the application to select the right operator portal to connect to (if supporting multiple operators).
- When pressing “send” button, the user’s browser is re-directed to the user’s operator portal.
- Endpoint URL to the operator portal was obtained from app registration.
- In the authorisation request, the application provides Application ID, target RCS resources (scope), and Redirect URI.



http://www.facebook.com

You are logged in as: Daniel Glifberg

Use RCS Set tagline App

Select Your RCS Service Provider: Orange...

Figure 7: Facebook App – End user application management panel

A.1.2.3 User Authentication (informative)

- User authentication is out of the scope of RCS API requirements. The following information is an example included for completeness.
- At the user's home operator portal, the user has to log in providing their user credentials.
- If the user has no password, the portal can offer the possibility to create one.
- If the user has no RCS/operator account, the portal can offer the possibility to create one.

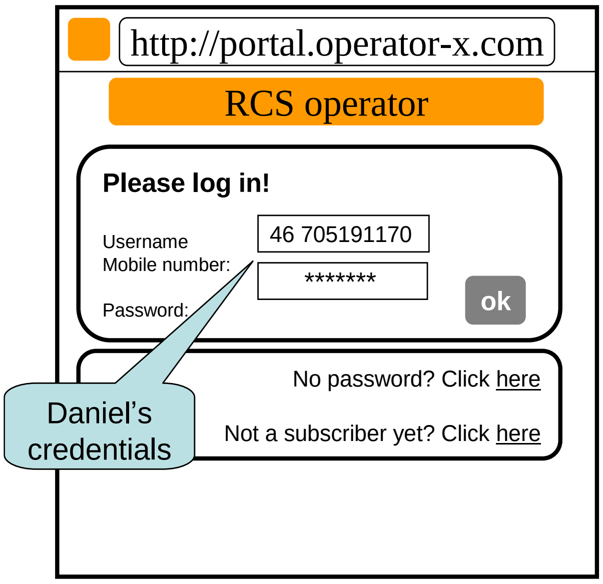
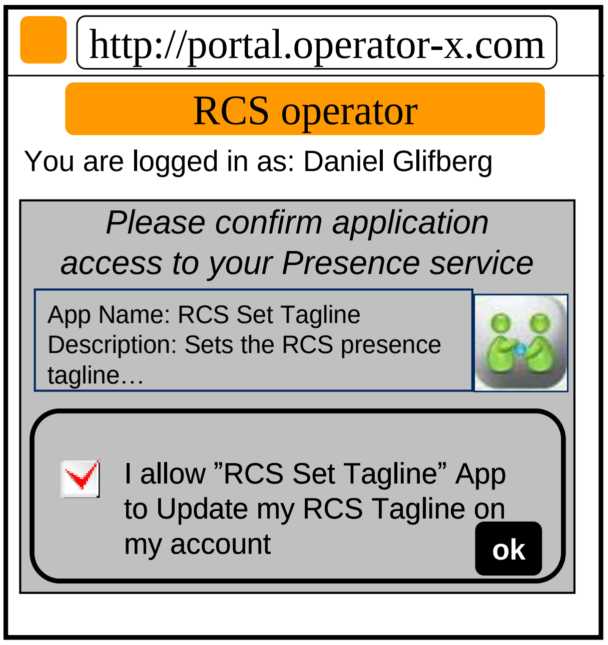


Figure 8: Generic Web App – End user application management entry panel

A.1.2.4 Application authorisation - (B): Facebook variant

- When logged in, the user is requested to grant the application access (i.e., authorize the application to access) the requested resource (e.g. my Location, SMS or Presence).
- This Authorisation Dialog is constructed from client_id and scope values supplied in the Authorisation Request previously sent to operator portal.
- The client_id, which identifies the application, was obtained from this operator in the previous application registration.
- The scope value(s), which identify a set of access permissions on resource(s), are typically found by the developer in API documentation and coded in the app.
- The Authorisation Dialog may be tailored according to the end-user's preferred language and device/browser type.
- After granting access, the user is redirected back to the original page, passing an authorisation code to the app.
- The portal/GW stores the binding between user identity, scope, authorisation code and application credentials.
- The web app can authenticate to the portal/GW to obtain an access token from the authorisation code.
- The application authorisation can also be for example time-limited or [to be standardized] based on usage (number of requests), etc.

- When expired, the user must again authorize the application to use the requested resource.


http://portal.operator-x.com
RCS operator
You are logged in as: Daniel Glifberg
Please confirm application access to your Presence service
App Name: RCS Set Tagline
Description: Sets the RCS presence tagline…
I allow "RCS Set Tagline" App to Update my RCS Tagline on my account
ok

Authorization Dialog

Figure 9: Generic Web App – End user application authorisation

- The application is now authorized to access to the resource of the user's RCS account.
- The RCS presence tagline can now be published from this app via the Presence enabler of the user's RCS Service Provider.
- The user can be charged for the request according to his Service Provider's policy (e.g. status updates through the API are included in his RCS subscription).

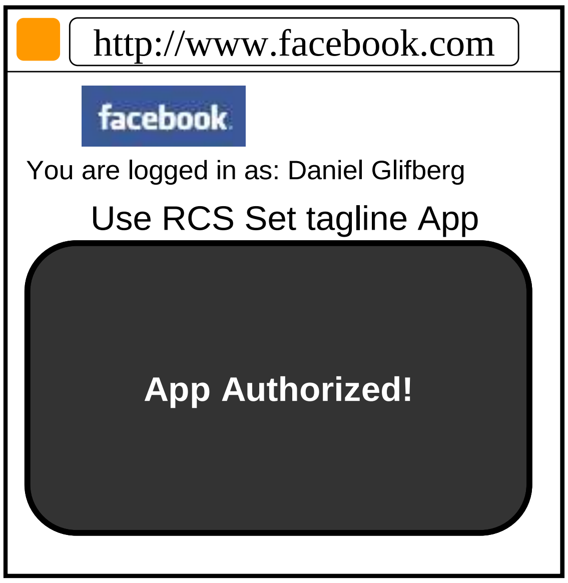


Figure 10: Facebook App – End user application authorisation completed

Note: Generic Web App variant is similar.

A.1.2.5 Application Authorisation - (C): Native Application on SMS-capable Device

In the case of Native application, the return of the Authorisation Code from the user agent (browser) to the application may not be possible depending on the characteristics of the application and device OS. To overcome this issue it is possible to deliver the Authorisation Code directly to the application via a binary SMS, provided that the device is SMS-capable. Alternatively other Push technologies can also be used (e.g., OMA connectionless Push over SMS, SIP Push).

The mechanism to be used in this case only differs only from the OAuth “Authorisation Code flow” used in the Facebook App and Generic Web App cases at the Authorisation Response step. In this case, the Authorisation Server does not redirect the User Agent to the OAuth Client in order to provide the Authorisation Code but instead it provides the code directly to the OAuth Client by sending it in a binary-SMS to the device aimed at a previously agreed-upon port.

It is for further study at the technical specification phase the means by which the application and the Authorisation Server agree on the delivery of the Authorisation Code via binary-SMS and the specific port where the binary SMS is to be delivered. This can be done at the application registration phase or otherwise indicated at the Authorisation Request.

This mechanism is valid for applications residing in non-RCS devices as well as in RCS devices. However, in the latter case it is valid only for applications installed in the RCS primary device.

The following figure depicts the Authorisation mechanism for Native applications described above.

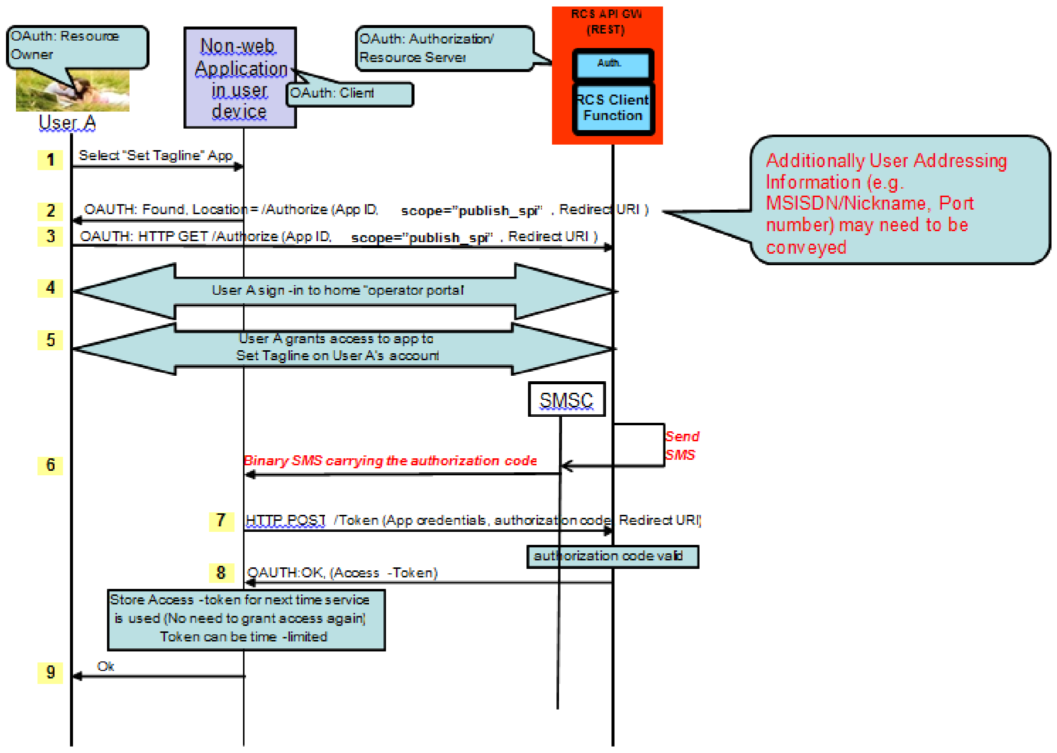
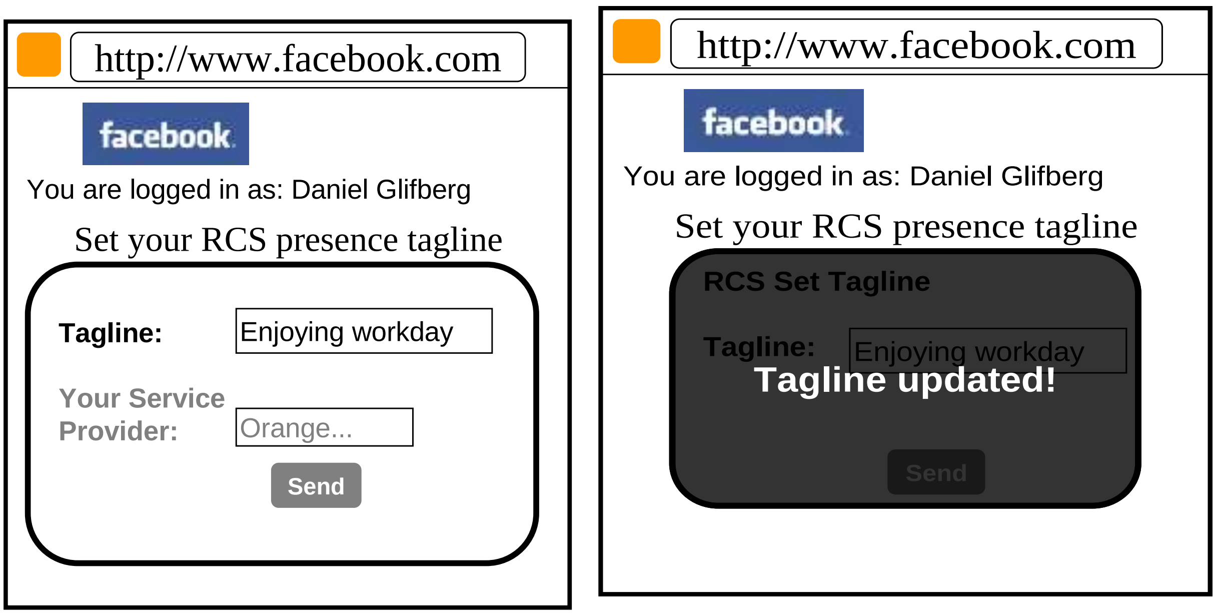

OAuth: Resource Owner
User A
Non-web Application in user device
OAuth: Client
OAuth: Authorization/ Resource Server
RCS API GW (REST)
Auth.
RCS Client Function
Additionally User Addressing Information (e.g. MSISDN/Nickname, Port number) may need to be conveyed
1 Select "Set Tagline" App
2 OAUTH: Found, Location= /Authorize (App ID, scope="publish_spi" , Redirect URI)
3 OAUTH: HTTP GET /Authorize (App ID, scope="publish_spi" , Redirect URI)
4 User A sign -in to home "operator portal
5 User A grants access to app to Set Tagline on User A's account
SMSC
Send SMS
6 Binary SMS carrying the authorization code
7 HTTP POST /Token (App credentials, authorization code, Redirect URI)
authorization code valid
8 OAUTH:OK, (Access -Token)
Store Access -token for next time service is used (No need to grant access again) Token can be time -limited
9 Ok

Figure 11 Application Authorisation – Native Application on SMS Capable Device

A.1.3 Application usage – User view

- The (Facebook) user can now use the “RCS Set Tagline” application.
- As the application now has a valid authorisation (connected to the user’s RCS Service Provider), the user will no longer be asked to authorize the application to Set Tagline on his account.
- The user is not required to select his Service Provider again.
- The application has been authorised access to the user’s RCS tagline resource.
- The new RCS presence tagline is now published via the Presence enabler of the user’s RCS Service Provider.
- The user can be charged for the request according to his Service Provider's policy (e.g. status updates through the API are included in his RCS subscription).


http://www.facebook.com

facebook

You are logged in as: Daniel Glifberg

Set your RCS presence tagline

Tagline: Enjoying workday

Your Service Provider: Orange...

Send

http://www.facebook.com

facebook

You are logged in as: Daniel Glifberg

Set your RCS presence tagline

RCS Set Tagline

Tagline: Enjoying workday

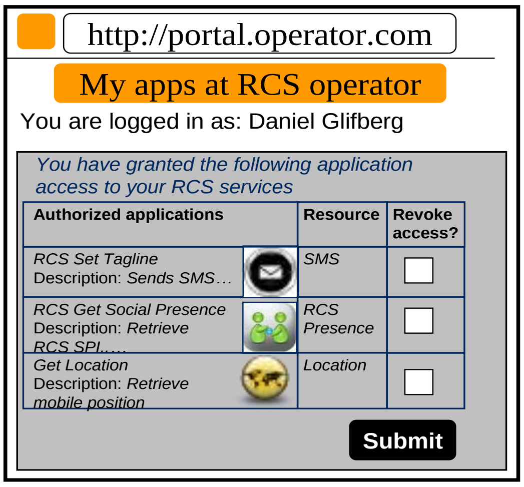
Tagline updated!

Send

Figure 12: Facebook App – End user application flow

A.1.4 Application authorisation control – User view

- The user is managing which applications they have been granted access to.
- The user can log on to their operator portal and get a list of applications they have been granted access to, which resource is granted for each app, and the possibility to revoke the access for an application.



http://portal.operator.com

My apps at RCS operator

You are logged in as: Daniel Glifberg

You have granted the following application access to your RCS services

Authorized applications	**Resource**	**Revoke access?**
RCS Set Tagline Description: *Sends SMS…*	*SMS*	☐
RCS Get Social Presence Description: *Retrieve RCS SPI….*	*RCS Presence*	☐
Get Location Description: *Retrieve mobile position*	*Location*	☐

Figure 13: End user application authorisation panel

Annex B Document Management

B.1 Document History

Version	Date	Brief Description of Change	Approval Authority	Editor / Company
1.0	29 April 2014	PRD edition prepared from RCS API requirements document version 2.4, incorporating the changes approved in GSG#12 and subsequent reviews, and GSMA quality review comments	RCC TF	Jose M Recio Solaiemes
2.0	28 February 2015	Update for RCS 5.3, see section 1.1.1 for a detailed list of changes	PSMC	Jose M Recio Comverse
3.0	19 October 2017	API updates and additions to support RCS Messaging as a Platform requirements	TG	Erdem Ersoz / GSMA
4.0	24 February 2018	API updates on MaaP anonymization features in section 5	TG	Erdem Ersoz / GSMA
5.0	16 October 2019	Add UNI-NMS-005 requirement in section 4.15 following the approval of CR1002	NG	Tom Van Pelt / GSMA

B.1.1 Other Information

Type	Description
Document owner	Network Group, Global Specification Group
Editor/company	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@gsma.com

Your comments or suggestions & questions are always welcome.