Rich Communication Suite

RCS API Detailed Requirements

Version 2.1

5 July 2012

This is a non-binding input document from the GSM Association to OMA.

Security Classification – NON CONFIDENTIAL GSMA MATERIAL

Copyright Notice

Antitrust Notice

Table of Contents

1 Introduction

1.1 Overview

GSMA RCS main objective is to bring a suite of services (using enablers from OMA and other SDOs) to market.

RCS is entering a new phase in its evolution; the introduction of 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 on the RCS APIs.

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. The definition is somewhat rough but has been very instrumental in the discussions:

- Device APIs
- B2B/Wholesale APIs
- 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 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. The B2B APIs will be considered a future work item for GSMA RCS and should be considered for a later stage.

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 to embed RCS enablers in very lightweight environments (such as pure web browser applications).

Throughout the rest of this document, the focus will be on the UNI/Long Tail APIs.

1.3 Architecture

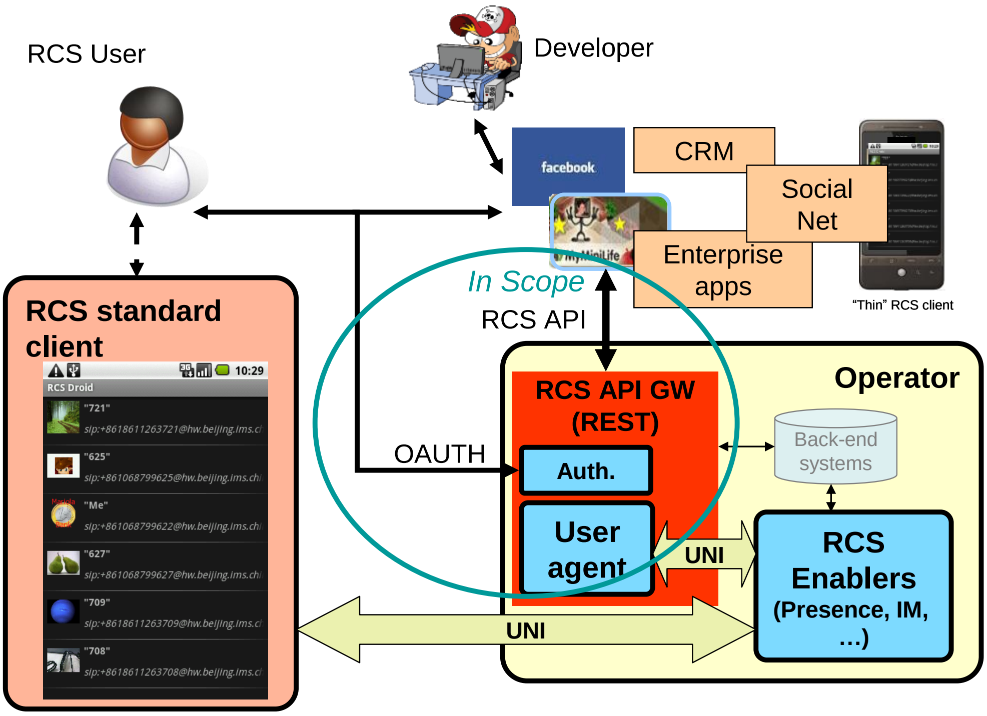

RCS User
Developer
facebook
CRM
Social Net
Enterprise apps
"Thin" RCS client
In Scope
RCS API
RCS standard client
10:29
RCS Droid
"721"
sip:+8618611263721@hw.beijing.ims.ch
"625"
sip:+861068799625@hw.beijing.ims.chi
"Me"
sip:+861068799622@hw.beijing.ims.chi
"627"
sip:+861068799627@hw.beijing.ims.chi
"709"
sip:+8618611263709@hw.beijing.ims.ch
"708"
sip:+8618611263708@hw.beijing.ims.ch
OAUTH
Operator
RCS API GW (REST)
Auth.
User agent
Back-end systems
UNI
RCS Enablers (Presence, IM, …)
UNI

Figure 1 RCS API Architecture

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

1. Application authorization 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. Operating on 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. It is foreseen that in many cases the existing developer portals and communities could accommodate RCS.
5. Application and user authentication (out of scope of this document). It is foreseen that in an RCS deployment, authentication mechanisms will be defined by the Service Provider, they could reuse the same authentication used for “regular” clients.

1.4 Definition of Terms

Term	Description
API	Application Programming Interface
CPM	Converged IP Messaging
IP	Internet Protocol
MMS	Multimedia Messaging Service
NNI	Network-to-Network Interface

RCS	Rich Communication Suite
REST	Representational State Transfer
SME	Small and Medium Enterprises
SMS	Short Message Service
UNI	User-to-Network Interface

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
[OAUTH20DRAFT]	The OAuth 2.0 Protocol Framework – Last DRAFT http://tools.ietf.org/html/draft-ietf-oauth-v2
[RCS5]	RCS 5.0 Advanced Communications Services and Client Specification http://www.gsmworld.com/document/ rcs5.0_advanced_communications_specification v1.0.pdf
[RCSR5OMAIMEND]	RCS 5.0 Endorsement of OMA SIP Simple IM http://www.gsmworld.com/document/ RCS5.0 Endorsement of OMA SIP Simple IM_1.0.pdf
[IR74]	GSMA IR.74 - Video Share Interoperability Specification http://gsmworld.com/documents
[IR79]	GSMA [IR79] Image Share Interoperability Specification http://gsmworld.com/documents
[IR84]	GSMA IR.84 - Video Share Phase 2 Interoperability Specification http://gsmworld.com/documents
[IR58]	GSMA IR.58 – IMS Profile for Voice over HSPA http://gsmworld.com/documents
[IR92]	GSMA IR.92 – IMS Profile for Voice and SMS http://gsmworld.com/documents
[IR94]	GSMA IR.94 – IMS Profile for Conversational Video Service http://gsmworld.com/documents
[ACRDRAFT]	The acr URI for anonymous users – Last DRAFT http://tools.ietf.org/html/draft-uri-acr-extension

2 RCS high-level requirements

Label	Description	Comment
UNI-HLF-001	The RCS API SHALL be HTTP/REST based.	
UNI-HLF-002	Resource URLs and primitives 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	This requirement might use the API

	and a server to negotiate the version of a particular resource	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 authorization 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, 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 neither for users nor 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] Call UNI API Requirements (see section 4.10) exposes RCS UNI for IP Voice and Video Call functionality. Ref: [RCSR5] 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 to individual developers. Developer security mechanisms are out of the scope of this document, and therefore out-of-scope for RCS APIs.

3 Authorization framework

Note: Authentication (of user, application, or developer) is out of the scope for 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.

Note: In the context of this section, “widget” should be understood in a loose 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 Authorization 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 Authorization framework SHALL support network-side Web applications, accessed from the user’s Web browser.	
UNI-AUT-003	The Authorization 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 Authorization framework SHOULD support client-side native code applications installed on the user’s terminal.	
UNI-AUT-005	The Authorization 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 Authorization 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 authorization and to access the user’s network resources.	
UNI-AUT-007	The Authorization framework SHALL support a third-party application to initiate the authorization request by directing the user to the Service Provider’s portal.	
UNI-AUT-008	The Authorization framework SHALL support presenting the third-party application’s authorization request to the resource owner in a form of an explicit authorization dialog or a user consent request.	It is assumed that the user has authenticated to the Service Provider before granting authorization (user authentication is out of scope of the Authorization 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 Authorization 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 authorization 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 Authorization framework SHALL enable the resource owner to authorize or deny access to each of the requested resources and operations.	
UNI-AUT-011	The Authorization framework MAY enable the resource owner to specify the duration for which his/her access authorization is granted.	
UNI-AUT-012	The Authorization 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 Authorization framework SHALL be able to provide to the third-party application an access token representing this user's authorization subject to obtaining it from the issuer.	
UNI-AUT-014	The access token SHALL only be usable by the third-party application for the restricted scope (operations on resources) authorized by the user at the time of authorization request.	
UNI-AUT-015	~~VOID~~	~~VOID~~
UNI-AUT-016	The Authorization 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 Authorization 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 Authorization framework SHOULD facilitate the possibility for the user to remove the authorization 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 authorization granted to the third-party application. As such, the server SHALL NOT send	Cf. requirement [UNI-NTF-005]

	notifications regarding a resource for which the application has no authorization.	

For an informative example, see Annex 1.

3.2 Authorization using OAuth

Label	Description	Comment
UNI-OAU-001	The Authorization framework SHALL be based on OAuth 2.0 as specified in [OAUTH20DRAFT].	Cf. requirement [UNI-HLF-007] Ref: [OAUTH20DRAFT]
UNI-OAU-002	The Authorization framework SHALL support the OAuth 2.0 “Authorization Code flow”, where the third-party application is a server-side web application.	
UNI-OAU-003	The Authorization 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 authorization code (“Authorization Code Flow”) / access token (“Implicit Grant Flow”) to a client-side installed application (widget or native code application), the Authorization 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 authorization code.	Annex 1 provides an informative example of such mechanism, based on binary-SMS. An alternative option would be to use the notification channel as the delivery mechanism.
UNI-OAU-005	The Authorization framework MAY support OAuth 2.0 flows other than the “Authorization Code Flow”.	
UNI-OAU-006	The Authorization framework SHALL support the OAuth 2.0 “Authorization Server” and “Resource Server” roles.	
UNI-OAU-007	The Authorization framework SHALL regard the users resources accessed via the RESTful API as the OAuth 2.0 “Protected Resource”.	
UNI-OAU-008	When following the Authorization Code Flow the Authorization framework SHALL generate an OAuth 2.0 authorization code as a result of the user authorization.	If other flows are used, a similar functionality should be provided.
UNI-OAU-009	The Authorization framework SHALL support the exchange of an authorization code for an access token according to OAuth 2.0.	
UNI-OAU-010	The Authorization framework SHALL bind the authenticated user identity to the generated authorization code and access token.	Note: The actual authentication mechanism used is out of the scope for 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 Authorization framework SHALL be able to determine the user identity (e.g. MSISDN) from the access token received from the application.	Note that in deployments this feature may not be available, or only available to privileged applications, in order to support privacy (e.g. using a Service Provider policy). See also UNI-HLF-008.
UNI-OAU-012	The Authorization 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: - presence_publish_spi - presence_publish_servicecapabilities - presence_subscriptions - chat - filetransfer - videoshare - imageshare - voice_call - multimedia_call - call_notification	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 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.

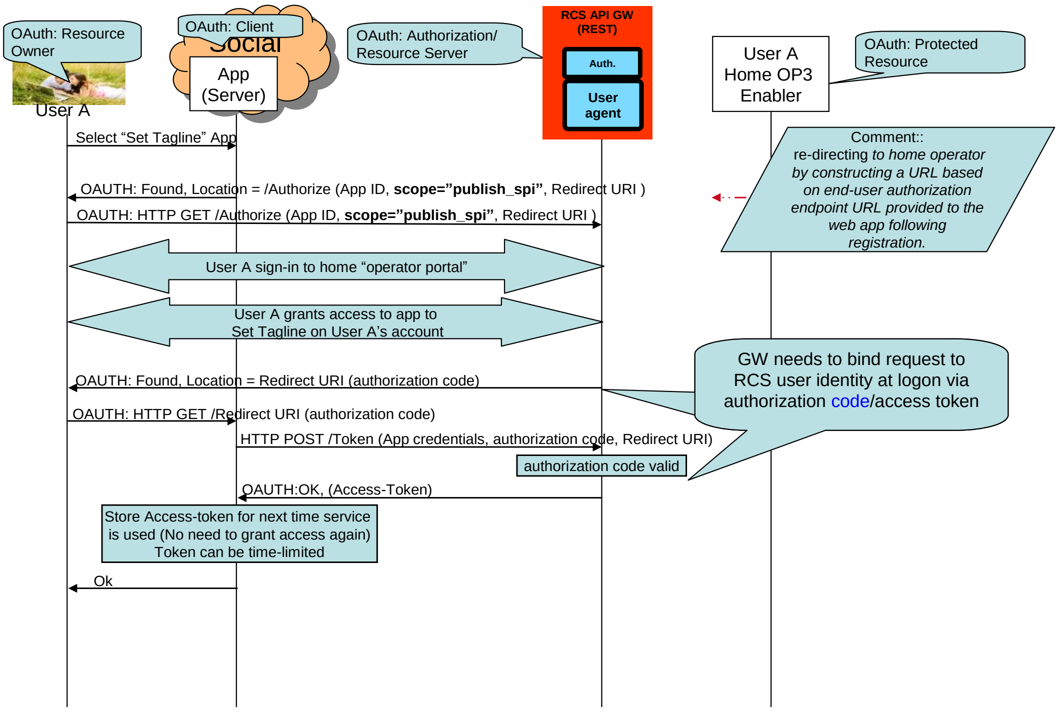

OAuth: Resource Owner
User A
OAuth: Client
Social
App (Server)
OAuth: Authorization/ Resource Server
RCS API GW (REST)
Auth.
User agent
User A Home OP3 Enabler
OAuth: Protected Resource
Select "Set Tagline" App
Comment:: re-directing *to home operator by constructing a URL based on end-user authorization 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 sign-in 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 Authorization of OAuth 2.0 in RCS Using OAuth Authorization Code Flow

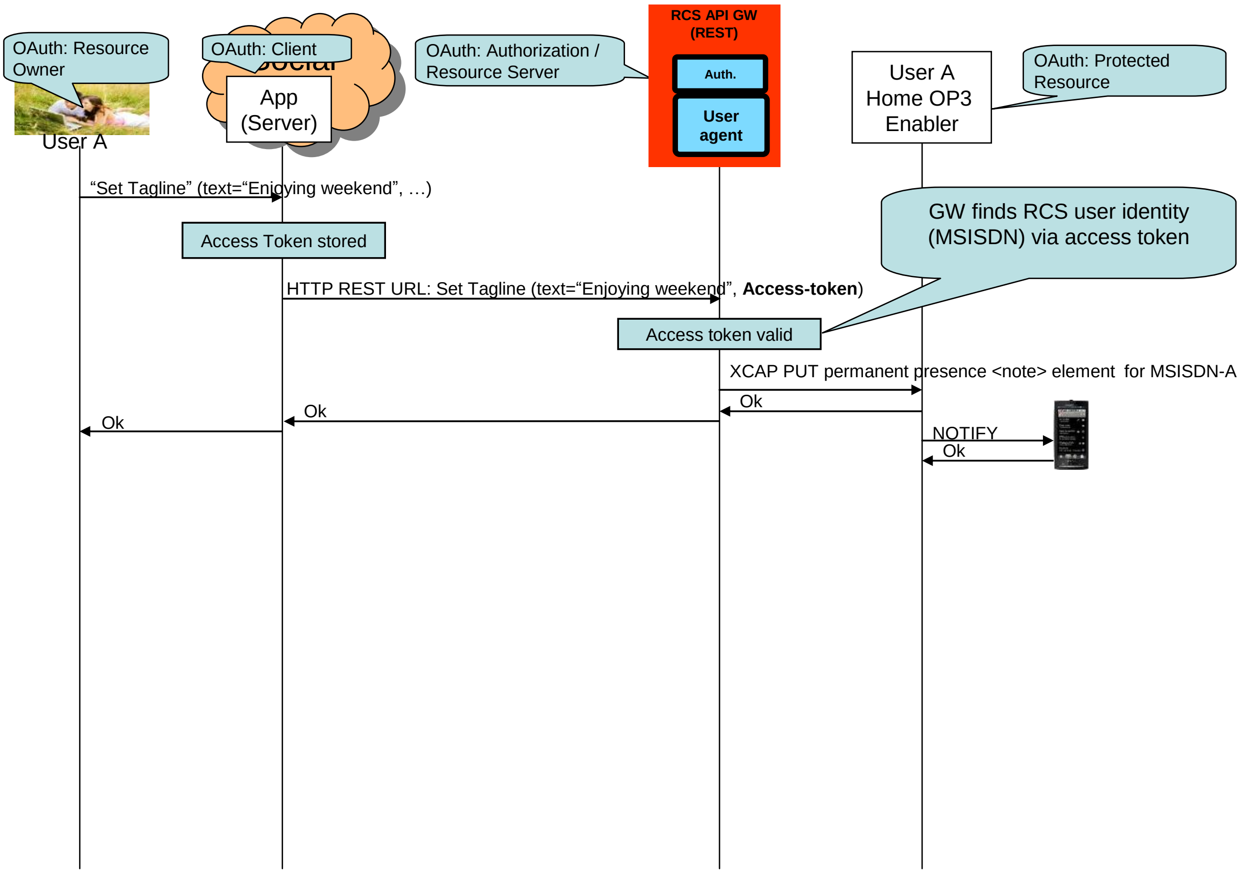

OAuth: Resource Owner
User A
OAuth: Client
Social
App (Server)
OAuth: Authorization / Resource Server
RCS API GW (REST)
Auth.
User agent
User A Home OP3 Enabler
OAuth: Protected Resource
"Set Tagline" (text="Enjoying weekend", …)
Access Token stored
GW finds RCS user identity (MSISDN) via access token
HTTP REST URL: Set Tagline (text="Enjoying weekend", Access-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 needs to alert a user of events (incoming chat invite, presence update from buddy, etc.). If each RCS service would have their 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 (as an 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 endpoint (i.e. a callback URL), using HTTP.	The application establishes a subscription to notifications by 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 foreseen 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 foreseen to be used mainly in environments that can not receive requests from the network or can not 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 authorization granted to the application, so the server SHALL NOT send notifications regarding a resource for which the application has no authorization.	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 is probably not interested on 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 foreseen 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”). EDITOR’s NOTE: It is foreseen that 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.	As there are no telco-related standards using these techniques, to 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.

4.1.2 Examples (informative)

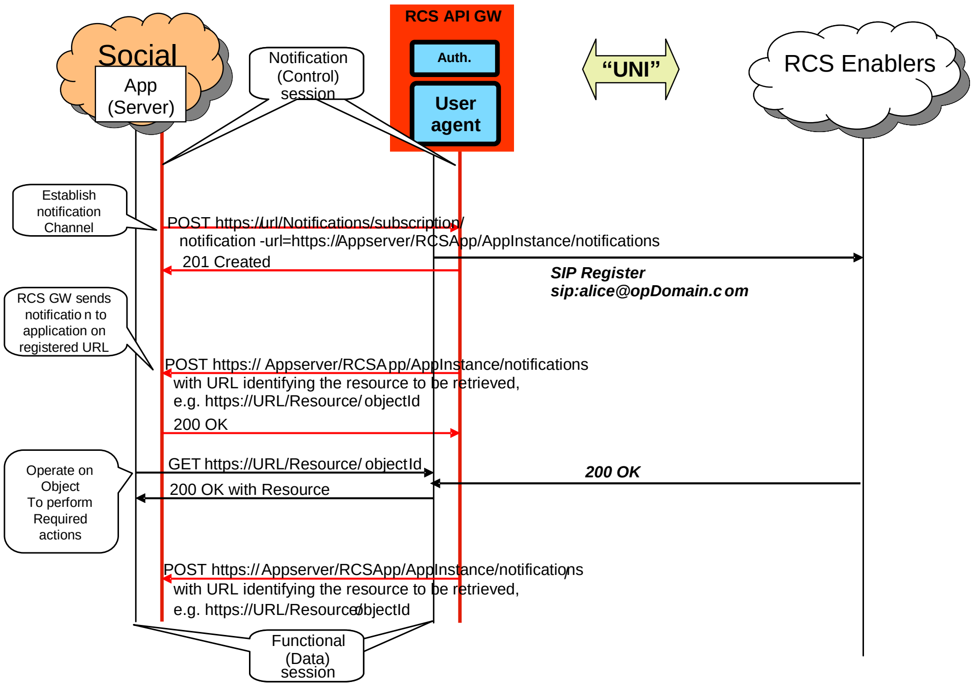

Social
App
(Server)
Notification
(Control)
session
RCS API GW
Auth.
User
agent
“UNI”
RCS Enablers
Establish
notification
Channel
POST https://url/Notifications/subscription/
notification -url=https://Appserver/RCSApp/AppInstance/notifications
201 Created
SIP Register
sip:alice@opDomain.c om
RCS GW sends
notificatio n to
application on
registered URL
POST https:// Appserver/RCSA pp/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/ objectId
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

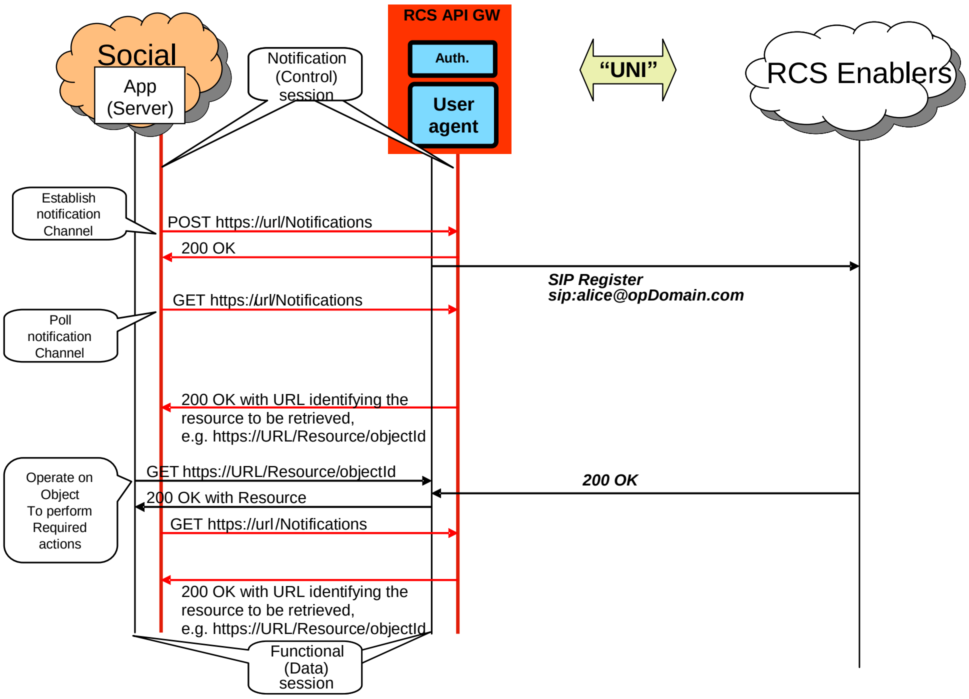

Social
App (Server)
Notification (Control) session
RCS API GW
Auth.
User agent
"UNI"
RCS Enablers
Establish notification Channel
POST https://url/Notifications
200 OK
SIP Register sip:alice@opDomain.com
Poll notification Channel
GET https://url/Notifications
200 OK with URL identifying the resource to 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 understood as 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.

4.2 Anonymized Customer Reference (ACR) API Requirements

The API gateway providing the RCS APIs SHALL NOT expose the real identities of the user and her/his contacts (see UNI-HLF-008). This means that the API will need to use anonymized 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 anonymized customer reference (ACR) associated to an MSISDN.	oauth_token={access-token}
		msisdn: {msisdn}
		
		return value:
		acr;{anonymized 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 varies 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 ([ACRDRAFT]) SHOULD be used for the anonymized customer reference.

4.3 Network Address Book API requirements

This chapter has an informative character. It captures the discussion of the working group on contact data and Network Address Books.

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 Network Address Book
4. Network Address Book of a Service Provider that does not offer the RCS API

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

For RCS API Service Providers that also run a network address book as specified below, it is recommended that the address book works with the anonymized customer references (ACRs) as specified in this document.

4.3.1 General considerations (informative)

1. NAB API’s main use case is to allow applications to get contact information and to receive updates on contact information (i.e. new contact added, contact information modified, etc). Additional operations are defined to allow applications to update the address book.
2. Depending on 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 Network Address Book is out-of-scope for the NAB API.

3. Depending on Service Provider's policies, trusted applications can get complete information (potentially including MSISDN or URI). OAuth 2.0 mechanisms can be leveraged to that end.
4. Retrieve address book allows optionally filtering. Only contacts or fields matching specified conditions will be returned.

   EDITOR 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 retrieving a filtered list of contacts in the Network Address Book and their associated information subject to Service Provider policies.	oauth_token={access-token} Optionally: filtering parameters	The answer amounts to retrieval of the list of contacts in the address book, possibly based on some filtering conditions. If filtering is requested then only matching contacts will be 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 info (ACRs). The contact identity (i.e. MSISDN or URI or ACR) returned is the only one that can be used by the rest of APIs (e.g. chat, file transfer, etc.). 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 Network Address Book.
			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 Network Address Book.		See “Common notification channel” for establishment of notification channel.
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 Network Address Book.	oauth_token={access-token}
		contact={contactid},
		contact data
			Add a contact in the Network Address Book.
			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 Network Address Book.	oauth_token={access-token}
		contact={contactid},
		contact data
			Update a contact in the Network Address Book.

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 Service Provider policy, applications created using the APIs shall be able to register and exchange new capabilities in order to be able to know when other user supports that application.

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 application on behalf the user.	oauth_token={access-token}
		capability: {capability_id}
			Use case: Game application using RCS to discover which contacts are also available for gamming.
			Note: Registering new application feature tags is subject to operator policies.

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}
			The network element providing this APIs should answer any user capability user request (via OPTIONS) returning only the feature tags related to the enabled capabilities.

UNI-CPD-003	The Capability Discovery API SHALL allow an application to query the service capabilities of a certain contact.	oauth_token={access-token}
		contact:{contact_id}
			Return value can consist of a list of capabilities.

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 is RCS capable or not.	oauth_token={access-token}
		contact: {contact_id}
		Return value: {true, false}	

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 SHALL support management of "free-text" presence attribute.	oauth_token={access-token} text={text} (e.g. "My picture is updated!")	Ref: [RCSR5] ch 3.7.1.3 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 on size, aspect ratio, file type, etc. should be verified by the RCS API GW.
			Ref: [RCSR5] 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: [RCSR5] 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: [RCSR5] 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: [RCSR5] ch 3.7.1.3 Social Presence Attributes, ch 3.7.2.2 Person

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 invitation of a member to share presence information.	oauth_token={access-token}
		contact={contactId}
		allow_location=true (or false)
			Adding an additional user to the "rcs" list will trigger a presence invitation towards the other party.
			Contact can be any URI (MSISDN, SIP URI or

			reference/object to a contact received via the Address Book API) Ref: [RCSR5] ch 3.7.1.4 Social Presence Authorization, 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 only be cancelled before the invitation has been accepted by the presentity (TBD if needed)
			Ref: [RCSR5] ch 3.7.1.4 Social Presence Authorization, 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: [RCSR5] ch 3.7.1.4 Social Presence Authorization, ch 3.7.4.3.3 Multidevice Handling, ch 3.7.4.5 Subscriptions and Authorization
			Editorial 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 SHALL support management (i.e. accept, block, ignore, revoke) of presence sharing invitations.	oauth_token={access-token}
		contact={contactId}
		allow_location=true (or false)
			Accepting a presence invitation is done by adding the user to the “rcs” list or “basic spi only” list
			[RCSR5] ch 3.7.1.4 Social Presence Authorization, 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: [RCSR5] ch 3.7.1.4 Social Presence Authorization, 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: [RCSR5] ch 3.7.1.4 Social Presence Authorization, 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: [RCSR5] ch 3.7.1.4 Social Presence Authorization, ch 3.7.4.3.3 Multidevice Handling, ch 3.7.4.5 Subscriptions and Authorization

UNI-PRS-012	The Presence API SHALL support subscriptions and notifications for presence information changes both for the own presentity or list of contacts.	oauth_token={access-token}
		contact={}
		“Structured presence information from presentities that the user share presence information with”
			Receive notifications about presence information changes from the presentities.
			See “Common notification channel” for establishment of notification channel.
			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: [RCSR5] ch 3.7.1.4 Social Presence

			Authorization, ch 3.7.4.5.1 Subscriptions and Authorization 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 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 own service capabilities	oauth_token={access-token}	Ref: [RCSR5] 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 Authorization

UNI-PRS-015	The Presence API SHALL support retrieval of service capabilities for a contact ("who can I invite")	oauth_token={access-token}
		contact={contactId}
			Contact can be any URI (MSISDN, SIP URI or reference/object to a contact received via the Address Book API).
			Ref: [RCSR5] ch 3.7.1.4 Social Presence Authorization, 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

The operations allow 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:

- sent
- delivered
- 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)}
			Content can be text or multimedia.
			Bearer service selection

		deliveryNotification = “yes”/”no”
		{content}
			(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: [RCSR5] 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: [RCSR5] 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:[RCSR5] 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.

EDITOR 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. “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).
			Subject parameter is optional; included when available.
			Chat object instance is created at reception of indication that invite and initial message was delivered (SIP 180), and a response was received.
			Ref: [RCS5] 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 their 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 and subject header	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: [RCSR5] 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: [RCSR5] 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: [RCSR5] 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: [RCSR5] 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 anymore. 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 session 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; he just receives a new message from a user. So there is no way that a user is 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.

Due to that fact, no functional requirements associated to one to one chat establishment (for either the originating or terminating side) are required by this model.

Also 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.

Label	Description	Required parameters (not complete list)	Comment

UNI-CHT-002b	The Chat API SHALL support starting a group chat.	oauth_token={access-token}
		recipient={contact1}, {contact2}, …
		subject={text} (e.g. “Hi”)
			Use case: Start a group chat (ad-hoc group).
			Subject parameter is optional; included when available.
			Chat session id must be returned to application
			Ref: [RCSR5] ch 3.4 Group Chat, [RCSR5OMAIMEND] ch 7.1 IM Client Procedures for IM Sessions

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 only possible 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).	oauth_token={access-token}	
UNI-CHT-007b	The Chat API SHALL support notifications about incoming chat invite.	Information about inviting user, subject header, and other invited participants (in case of group chat)	Use case: 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: [RCSR5] ch 3.4 Group Chat, [RCSR5OMAIMEND] ch 7.1 IM Client Procedures for IM Sessions
UNI-CHT-008b	The Chat API SHALL support accepting a group chat invitation.	oauth_token={access-token}	Use Case: User accepts a group chat invitation. Ref: [RCSR5] ch 3.4 Group Chat, [RCSR5OMAIMEND] ch 7.1 IM Client Procedures for IM Sessions
UNI-CHT-009b	The Chat API SHALL support declining a group chat invitation.	oauth_token={access-token}	Use Case: User declines a group chat invitation. Ref: [RCSR5] ch 3.4 Group Chat, [RCSR5OMAIMEND] ch 7.1 IM Client Procedures for IM Sessions
UNI-CHT-011	The Chat API SHALL support leaving a group chat.	oauth_token={access-token}	Use Case 1: User leaves a group chat. This ends the chat for this user. Use Case 2: If group chat originating user leaves the group chat, depending on the operator policies the group chat session could be terminated or not. Ref: [RCSR5OMAIMEND] ch 7.1 IM Client Procedures for IM Sessions
UNI-CHT-012b	The Chat API SHALL support notifications about “group chat ended”.		In case of group chat termination the users will receive a notification about that event. See use case 2 of previous requirement. See “Common notification channel” for establishment of notification channel. Ref: [RCSR5OMAIMEND] ch 7.1 IM Client Procedures for IM Sessions

UNI-CHT-013	The Chat API SHALL support extending a confirmed 1-to-1 chat to a group chat.	oauth_token={access-token}
		recipient={contact1}, {contact2}, …
			Use Case: User adds one or more participants to the 1-to-1 chat. All participants except the originator receive a chat invitation.
			Ref: [RCSR5] ch 3.4 Group Chat , [RCSR5OMAIMEND] ch 7.1 IM Client Procedures for IM Sessions

UNI-CHT-014	The Chat API SHALL support adding a set of users to a group chat.	oauth_token={access-token}
		recipient={contact1}, {contact2}, …
			Use Case: User adds one or more participants to the group chat. The new participant(s) receive(s) a

			chat invitation. Ref: [RCSR5OMAIMEND] ch 7.1 IM Client Procedures for IM Sessions

UNI-CHT-015	The Chat API SHALL support re-joining a group chat.	oauth_token={access-token}
		chat conference id={sessionid}
			Use Case: User wants to join a chat (possible use cases: invitation has expired, user left and wants to rejoin, and so on). As a result, user successfully re-joined chat (if chat/session found), or alternatively an indication is returned that chat/session not found (due to expiry).
			Ref: [RCSR5OMAIMEND] ch 7.1 IM Client Procedures for IM Sessions

UNI-CHT-016	VOID	VOID	VOID EDITOR NOTE: Already covered by the initial subscription of the client to chat related notifications.
UNI-CHT-017	The Chat API SHALL support notifications about participant information in a group chat. An initial notification SHALL be sent to an invited participant upon invitation acceptance. Subsequent notifications SHALL be sent to all connected participants including the originator, when the set of participants changes		Use case: The application receives notifications about the changing set of participants in a group chat session. See “Common notification channel” for establishment of notification channel. Ref: [RCSR5OMAIMEND] ch 7.1 IM Client Procedures for IM Sessions

4.7.4 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 messages and acknowledging the successful sending of those messages.	oauth_token={access-token}
		message_content={content}
		return:
		status: {success,pending}
			Use case: The application sends a chat message. Content can be only text according to RCS specifications.
			If the message was successfully delivered a “success” response is returned.

			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”” …
			Use case: The application sends “isComposing” which indicates that a user is currently composing a message.
			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. See “Common notification channel” for establishment of notification channel. EDITOR NOTE: Add: Store & Forward use case. 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. 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 and acknowledging the successful sending of those messages.	oauth_token={access-token}
		message_content = Body{multimedia content}
		content type={content type}
		return:
		status: {success,pending}
			Use case: The application sends a multimedia chat message (e.g. image, video clip, audio clip, etc).
			If the message was successfully delivered a "success" response is returned.
			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: [RCSR5] 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 a url to download the content.
			The server which the URL is pointed to SHALL be ready to receive download requests when the notification is sent.

4.7.5 Notifications

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

- "Sent" notification: generated when the RCS client has successfully sent the message to the next hop (i.e. IM Server if store and forward is enabled on the network). In the case of the APIs it should be generated by the API gateway and notified to the application when it successfully has sent the message.
- "Delivery" notification: generated when the message arrives to 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 of sending back the delivery notifications of incoming messages as they are received by the application. In order 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.

- “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 the responsible of generating these “displayed” notifications accordingly. The API gateway shall also be able to receive them and notify the application.

References: [RCSR5] 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 messages notifications [“sent”, “delivered” and “displayed”] for messages sent in a 1 to 1 session.		Message notifications SHALL be returned asynchronously via the notification channel except the “sent” notification. As stated in the UNI-CHT-018 the “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 confirmed (session aware) model is used it shall be possible to send the “displayed” notifications even it 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 the UNI-CHT-018 the “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.

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.	oauth_token={access-token}
		recipient={contactid}
		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.
			A SIP INVITE request is sent to the remote party (i.e. the contact).
			A file transfer instance is created at 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: [RCSR5] ch 3.8 IP Voice Call (IR.92 and IR.58), [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 that created the invitation can cancel it, and it is only offered 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,		The final set of applicable notification types will be determined in the technical work phase.

	cancelled, ended) to the originating side.		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, 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 their technical API work. Ref: [RCS5] 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.		Use case: User is invited to a file transfer session. See "Common notification channel" for establishment of notification channel. Ref: [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 side.	oauth_token={access-token}	Use case: File transfer session is to be closed. A SIP BYE request for the selected session is sent to the remote party. Ongoing file transfer can only be cancelled once 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 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 if this is sent when the first chunks of MSRP data are received and allow to simultaneously receiving 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 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, 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 their technical API work. Ref: [RCS5] ch 3.5 File Transfer; ch 3.5.3 High Level Requirements

4.9 Call UNI API requirements

The Call UNI API requirements are based on OMA ParlayREST 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: User initiates a call between its own terminal and another user.
			Initiating a session results in the user’s all terminals being ring. 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: 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 alerting”.		Use case: Application receives call invitation notification that the user’s phone is ringing. 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 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	The Call API(s) MAY		Note that this event may or

	support notifications about “call declined”.		may not be generated by the actual API gateway, depending on the underlying network infrastructure. In a SIP environment, this maps to 603 Decline. See “Common notification channel” for establishment of notification channel.

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 for all the participants.		Use case: The application may request current status of media other than voice (e.g. video, text) during active multimedia telephony call either for a specific participant or all participants.
UNI-CLL-013	The Call API SHALL allow indication of addition or removal of media stream in particular video.		Use case: The application may request to get notification of loss of video stream due to poor network coverage, or an end user dropping video stream for an ongoing multimedia call
UNI-CLL-014	The Call API SHALL allow indication of change of media stream direction in particular video.		Use case: The application may request to get notification if media stream direction changes (e.g. in the case of an ongoing video stream changes from duplex to simplex). Ref: [IR94] ch 2.2.2 Call Establishment and Termination
UNI-CLL-015	The Call API SHALL allow indication of the media interaction for a call participant.		Use case: The application may request to get notification of media events (e.g. the end user pausing

			playback of a media stream).
UNI-CLL-016	The Call API SHALL allow addition and removal of media streams, in particular video and text		Use case: The application may request media other than voice (e.g. video, text) after having started a voice only telephony call.
UNI-CLL-017	The Call API SHALL allow control the media stream direction (i.e. unidirectional, bi-directional) for each media type.		Use Case: To comply with privacy requirements in certain regions, the application may request that the video stream in a video call be changed between simplex or duplex mode. Ref: [IR94] ch 2.2.2 Call Establishment and Termination

4.10 Video Share UNI API requirements

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

4.10.1 Video Share use cases (informative)

To clarify the requirements in the next sections, the intended basic use cases of the VideoShare 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 their customers to browse a catalogue of music videos, and stream them by click to clients. The application uses a video file as the source of the video stream of the VS.

   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.

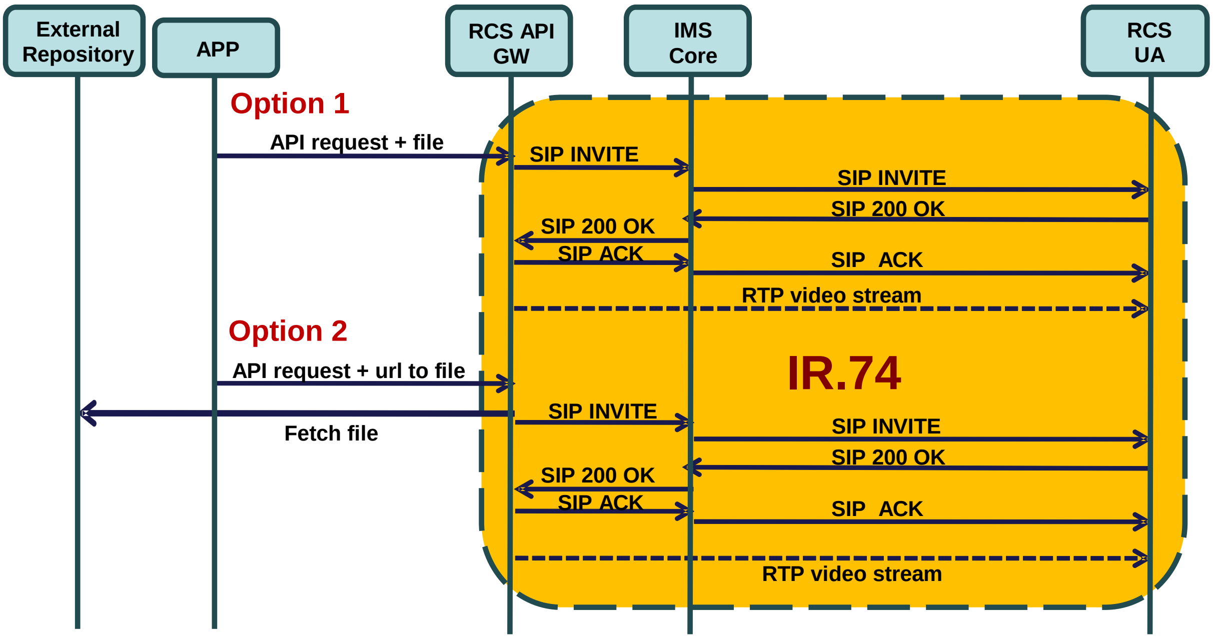


Figure 6 Schematic flow for Video Share Use Case 1

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, 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).

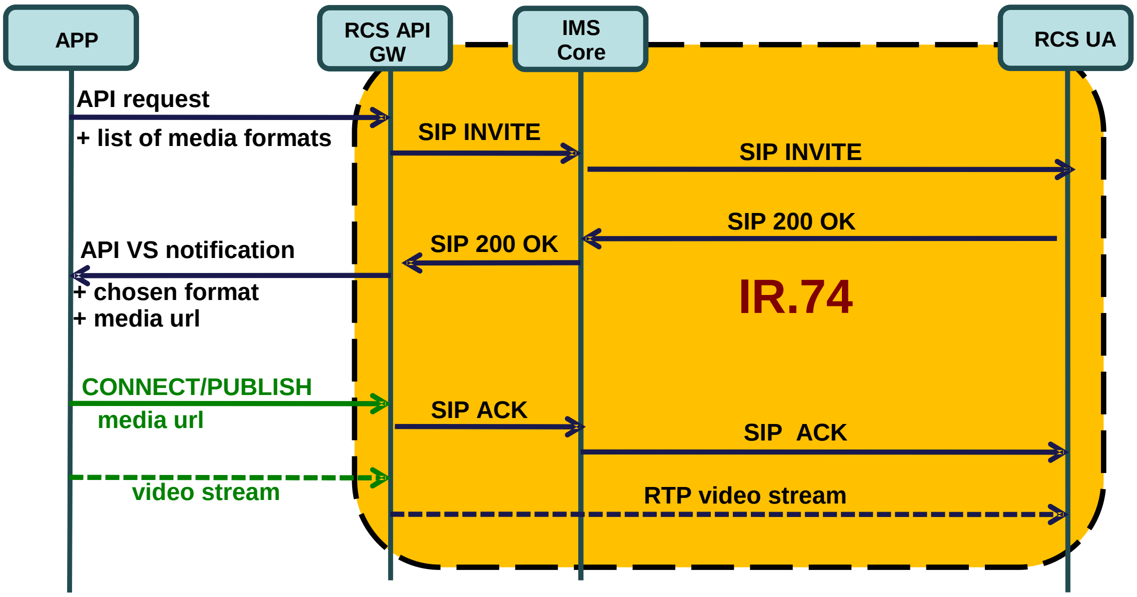


Figure 7 Schematic Flow for Video Share Use Case 2

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 VS 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 VS 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., 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 8).

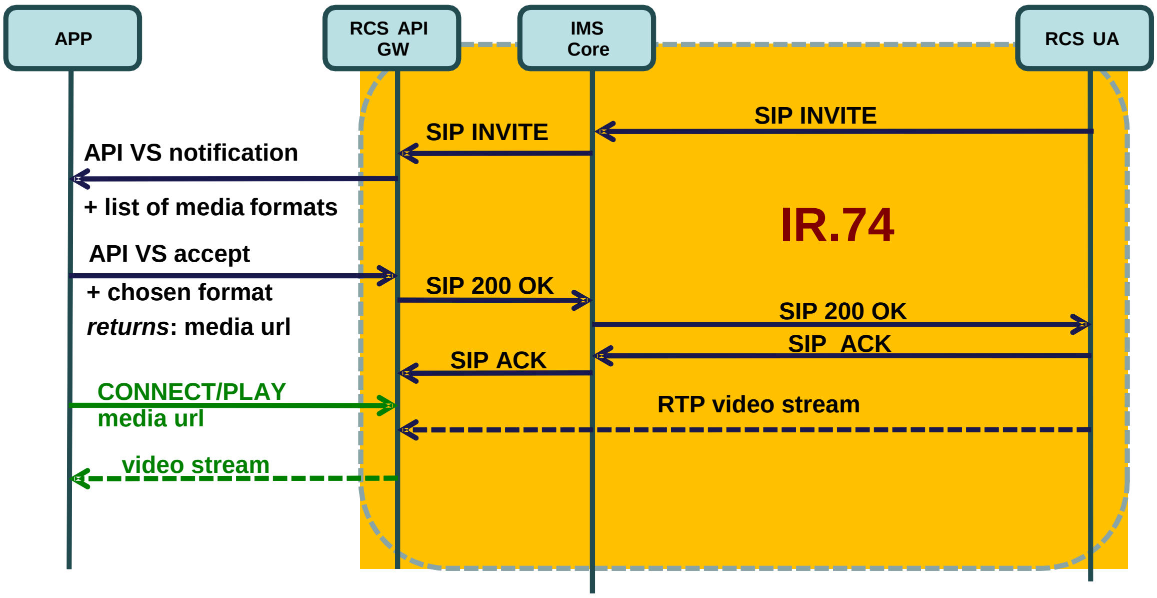


Figure 8 Schematic Flow for Video Share Use Case 3

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

4.10.2 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 that created the invitation can cancel it, and it is only offered 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. In the case 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.

4.10.3 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 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 BYE to originating side.
UNI-VSH-011	The Video Share API SHALL support notifications about “Video Share” (“ended”, “cancelled”, “failed”) 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.

4.11 Image Share UNI API requirements

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

4.11.1 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 IS is started by the application using the API. The application uses an image file as the source of the IS 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.

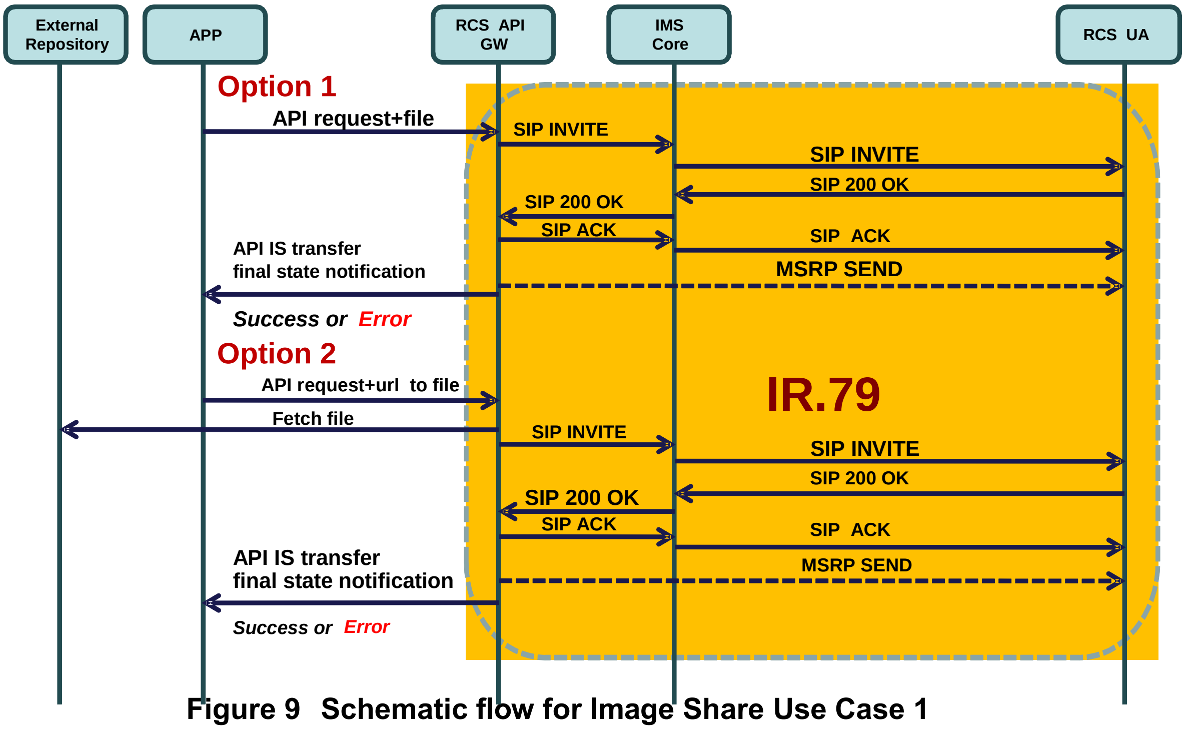


Figure 9 Schematic flow for Image Share Use Case 1

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

   The IS 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 IS session is correctly established, the application will be notified and given a URL in which the file can be downloaded.

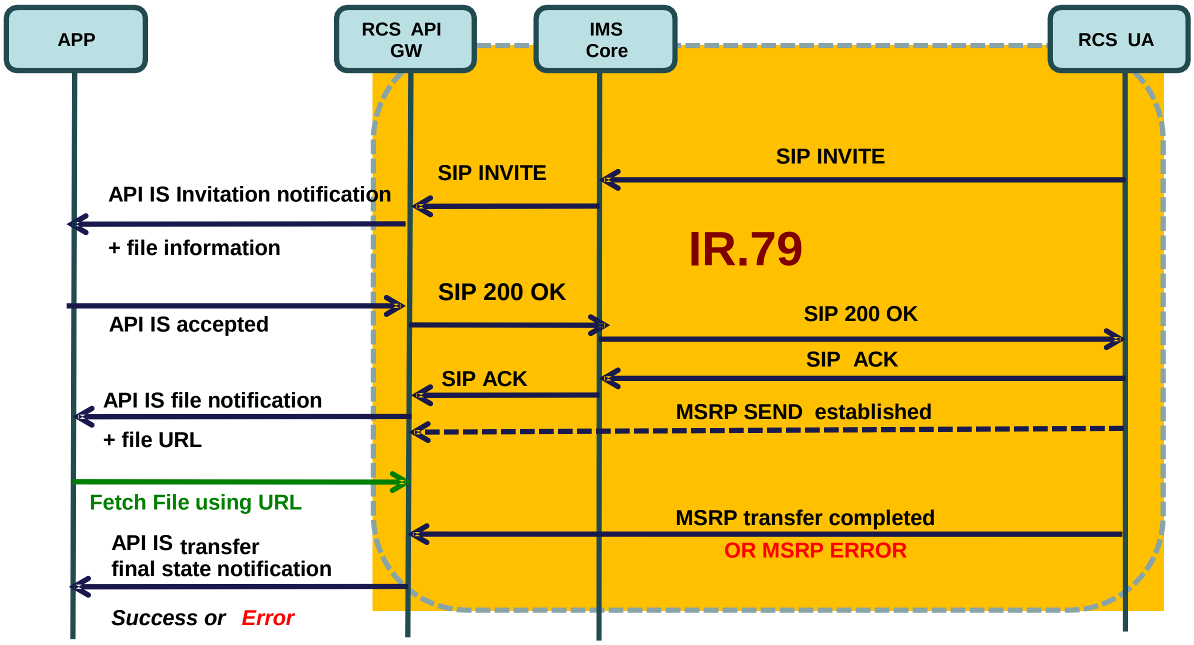


Figure 10 Schematic flow for Image Share use case 2

4.11.2 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 only offered before the session is accepted.
UNI-ISH-005	The Image Share API		The final set of applicable

	SHALL support notifications about Image Share ("alerting", "accepted", "ended", "declined", "failed")		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: Application on originating side stops Image Share. A SIP BYE is sent to the remote end.

4.11.3 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 a Image Share invitation.	Inviting contact Reference to an ongoing call (for IR.79)	Use case: Application on terminating side receives 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 a Image Share by the terminating side.	oauth_token={access-token}	Use case: Application on terminating side accepts Image Share. Triggers sending a SIP 200 (if accepted) or a suitable rejection cause (if declined) to originating side.
UNI-ISH-010	The Image Share API SHALL support ending a Image Share by the terminating side.	oauth_token={access-token}	Use case: Application on terminating side ends Image Share. Triggers sending BYE to 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 the image share content is available for download	url={img url}	The gateway will send this notification to the client with url to download the image. 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 when the whole file has been received. In any case the notification SHALL be sent before the final state notification is sent.

4.11.4 Capability Query UNI API requirements

Refer to section 4.3.3 (Services capabilities).

4.12 Location Pull

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

The Location Pull API should be agnostic to the underlying location based service technology (i.e. SUPL or control plane) used in the location query.

The Location PULL API requirement herein is based on the UNI specification of RCS 5.0; therefore, additional parameters or information available from the OMA Terminal Location API are outside the scope of this specification.

References: [RCSR5] Section 3.10.1.2 Geolocation PULL feature

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
	SHALL support the request of positioning accuracy in meters.
		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. Application may optionally use other available contactID attribute (ACR) to request pulling the location of a given contact in address book.
			The requested accuracy of the positioning result is in meters.

			Typically, a request for higher positioning accuracy may take longer to retrieve than a request for coarse accuracy.

5 Annex 1: RCS API Authentication & Authorization – Use Cases

5.1 Overview

Use case examples and flows for detailing requirements on

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

Using MSISDN for user authentication and OAuth for application authorization

Type of application: network-side web application, illustrated with two variant, both of them following the OAuth Authorization 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 RCS Set Tagline web app
- The end-user accesses to RCS Set Tagline web app from any browser

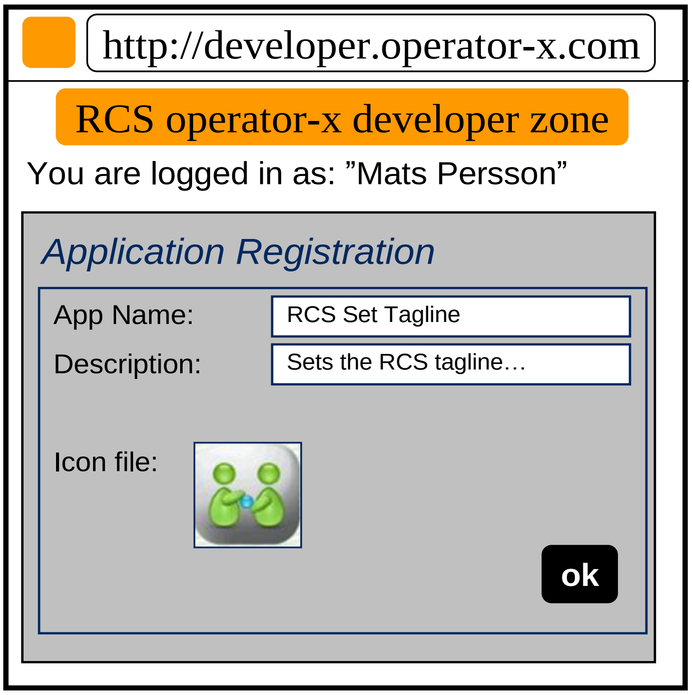
(B) “App on Facebook”

- 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

5.2 Application registration – Developer view

5.2.1 (A) General

- The developer (”Mats Persson”) 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”

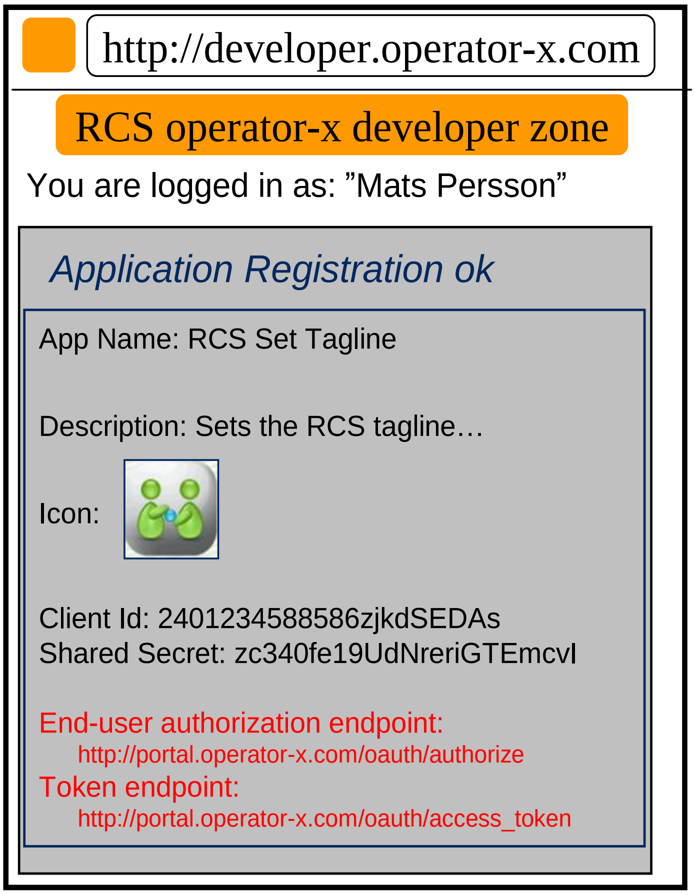
Application Registration

App Name: RCS Set Tagline

Description: Sets the RCS tagline…

Icon file:

- 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 Authorization Server (end-user authorization endpoint and token endpoint).
- The Application is then deployed in 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…

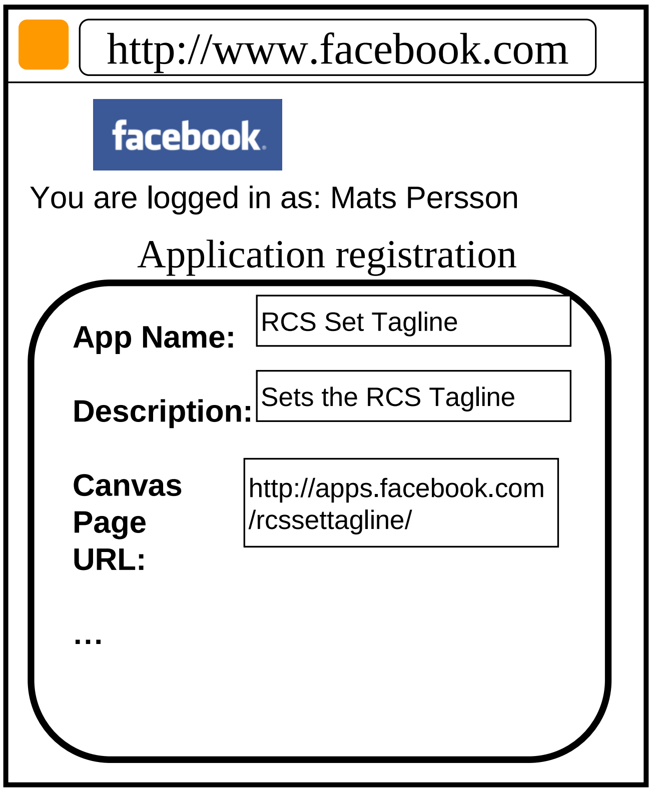
Icon:

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

5.2.2 (B) Additional step in case of Facebook variant

- The developer (”Mats Persson”) wants to publish his ”RCS Set Tag Line” web app as an ”App on Facebook”.
- The developer logs in to his 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 besides assign to this app some OAuth 2.0 credentials; however they are only used 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

You are logged in as: Mats Persson

Application registration

App Name: RCS Set Tagline

Description: Sets the RCS Tagline

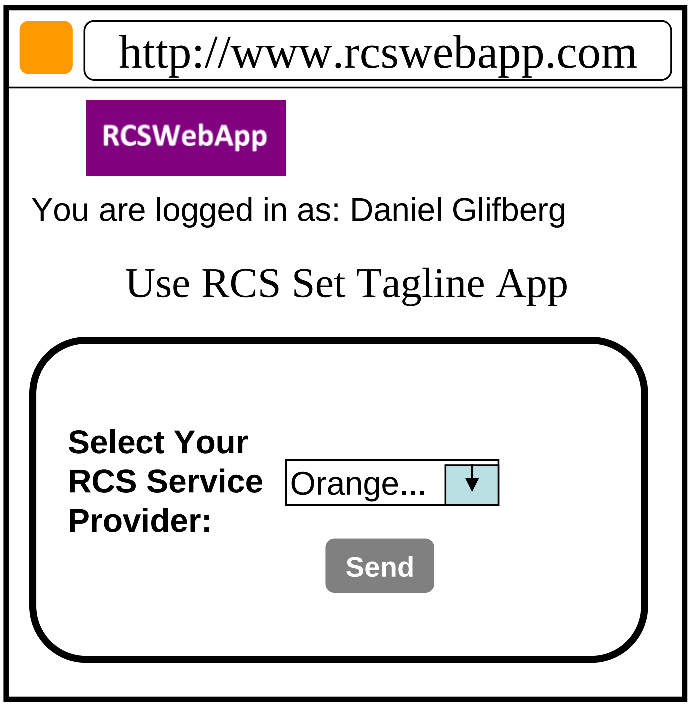
Canvas Page URL: http://apps.facebook.com/rcssettagline/

…

5.3 Application authorization – User view

5.3.1 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. As an 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 “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 authorization request, the application provides Application ID, target RCS resources (scope), and Redirect URI.



http://www.rcswebapp.com

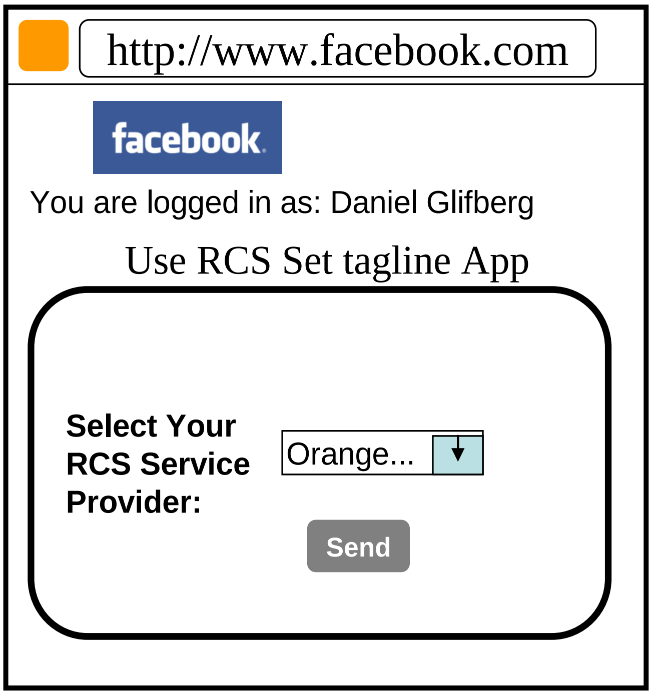
You are logged in as: Daniel Glifberg

Use RCS Set Tagline App

Select Your RCS Service Provider: Orange...

5.3.2 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 authorization request, the application provides Application ID, target RCS resources (scope), and Redirect URI.



http://www.facebook.com

You are logged in as: Daniel Glifberg

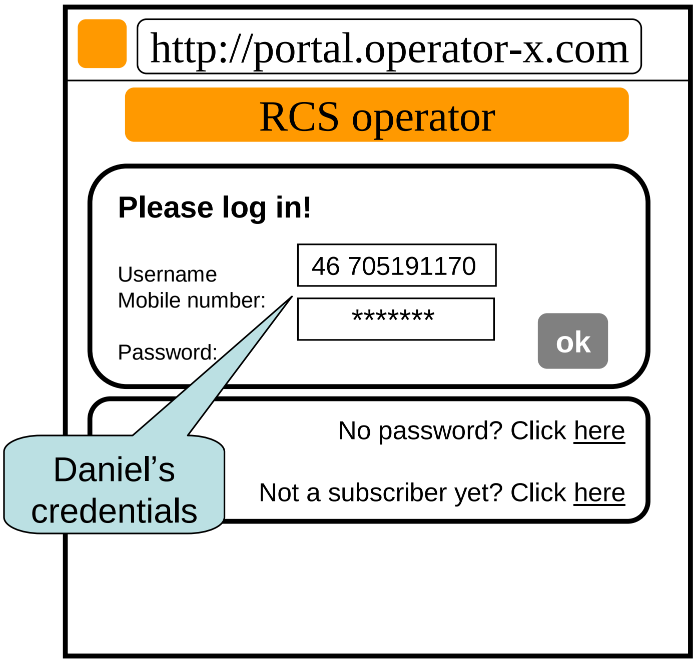
Use RCS Set tagline App

Select Your RCS Service Provider: Orange...

5.3.3 User Authentication (informative)

User authentication is out of the scope of RCS API requirements. Following is an example included for completeness.

- At the user's home operator portal, the user has to log in providing his 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.



http://portal.operator-x.com

RCS operator

Please log in!

Username Mobile number: 46 705191170

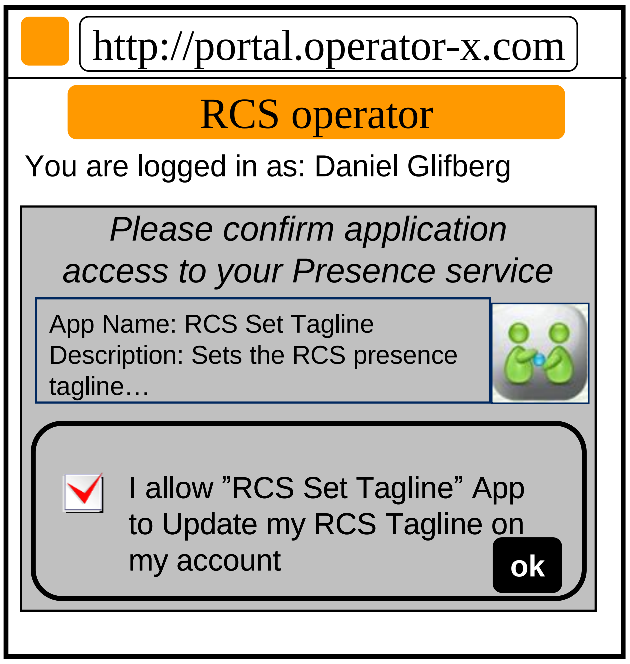
Password: *******

No password? Click here

Not a subscriber yet? Click here

5.3.4 Application authorization - (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 Authorization Dialog is constructed from client_id and scope values supplied in the Authorization Request previously sent to operator portal.
  - The client_id, which identifies the application, was obtained from this operator in previous application registration.
  - The scope value(s), which identifies a set of access permissions on resource(s), is typically found by the developer in API documentation and coded in the app.
  - The Authorization Dialog may be tailored according to end-user's preferred language and device/browser type.
- After granting access, the user is redirected back to original page, passing an authorization code to the app.
  - The portal/GW stores the binding between user identity, scope, authorization code and application credentials.
  - The web app can authenticate to the portal/GW to obtain an access token from the authorization code.
- The application authorization can also be for example time-limited or [to be standardized] based 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

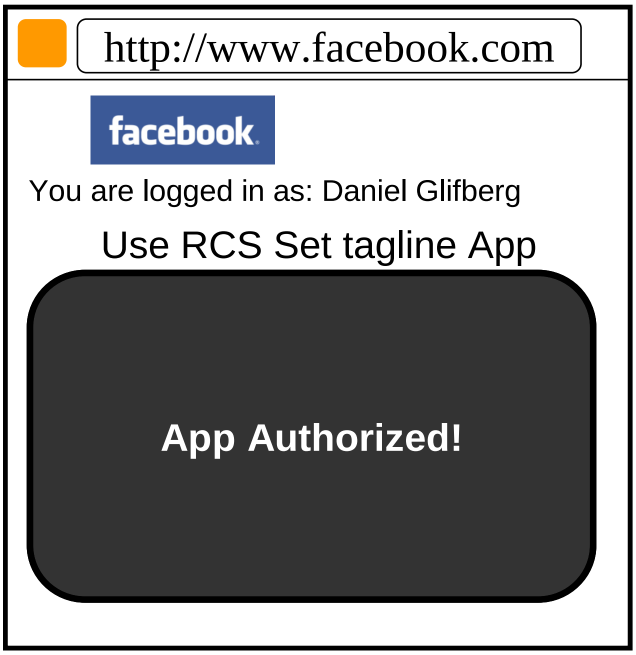
App Name: RCS Set Tagline
Description: Sets the RCS presence tagline…

☑ I allow ”RCS Set Tagline” App to Update my RCS Tagline on my account

Authorization Dialog

- 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).



http://www.facebook.com

facebook

You are logged in as: Daniel Glifberg

Use RCS Set tagline App

App Authorized!

Note: Generic Web App variant is similar.

5.3.5 Application Authorization - (C): Native Application on SMS-capable Device

In the case of Native application, the return of the Authorization Code from the user agent (browser) to the application may not be possible depending on the characteristics of the application and device OS. In order to overcome this issue it is possible to deliver the Authorization 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 from the OAuth “Authorization Code flow” used in the Facebook App and Generic Web App cases at the Authorization Response step. In this case, the Authorization Server does not redirect the User Agent to the OAuth Client in order to provide the Authorization 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 port.

It is for further study at the technical specification phase the means by which the application and the Authorization Server agree on the delivery of the Authorization 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 Authorization 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 only valid for applications installed in the RCS primary device.

The following picture depicts the Authorization mechanism for Native applications described above.

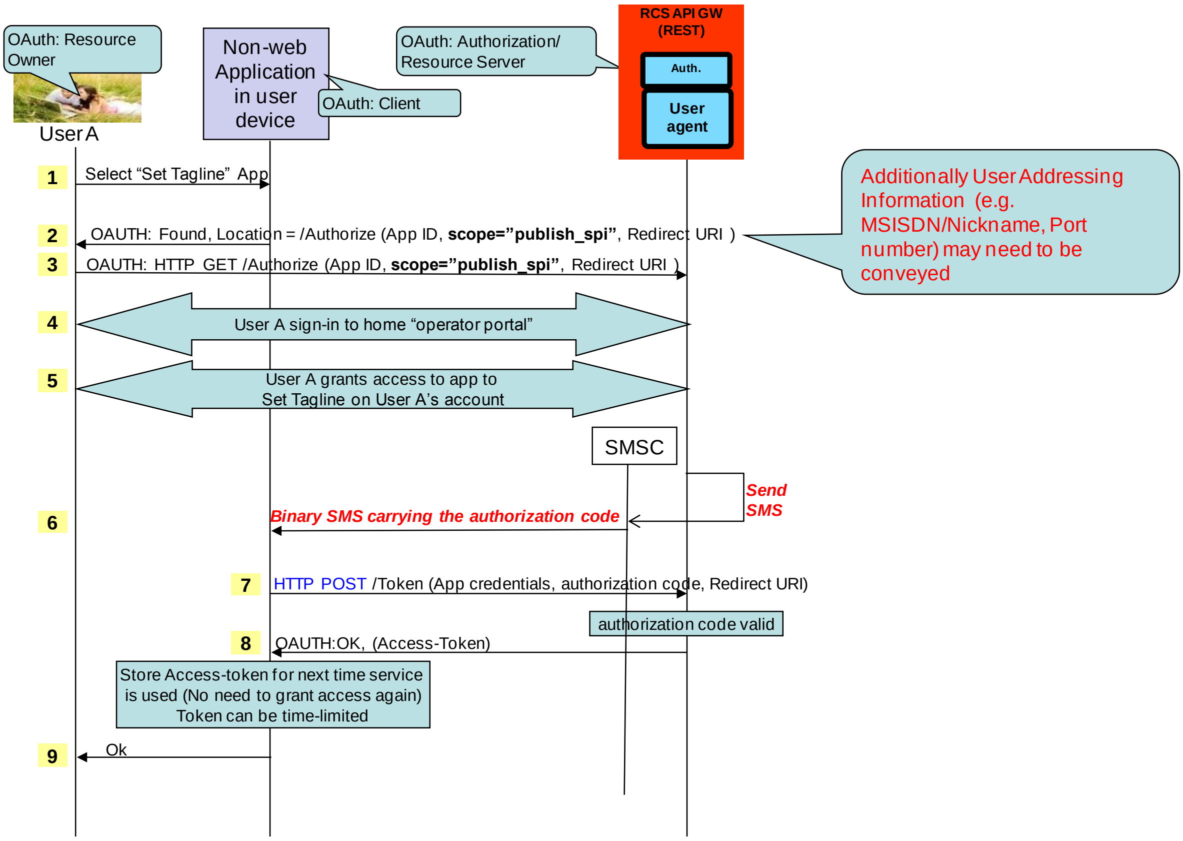
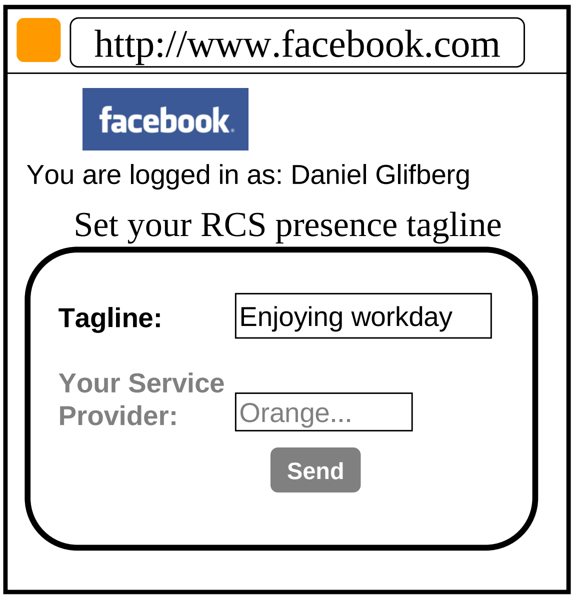

OAuth: Resource Owner
User A
Non-web Application in user device
OAuth: Client
OAuth: Authorization/ Resource Server
RCS API GW (REST)
Auth.
User agent
Additionally User Addressing Information (e.g. MSISDN/Nickname, Port number) may need to be conveyed
1
Select "Set Tagline" App
2
OAUTH: Found, Location = /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 Authorization – Native Application on SMS Capable Device

5.4 Application usage – User view

- The (Facebook) user can now use the "RCS Set Tagline" application.
- As the application has now a valid authorization (connected to the users RCS Service Provider), the user will no longer be asked to authorize the application to Set Tagline on his account.
- The user does thus neither need to select his Service Provider again.
- The application is granted a priori to access the user's RCS account.
- 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).



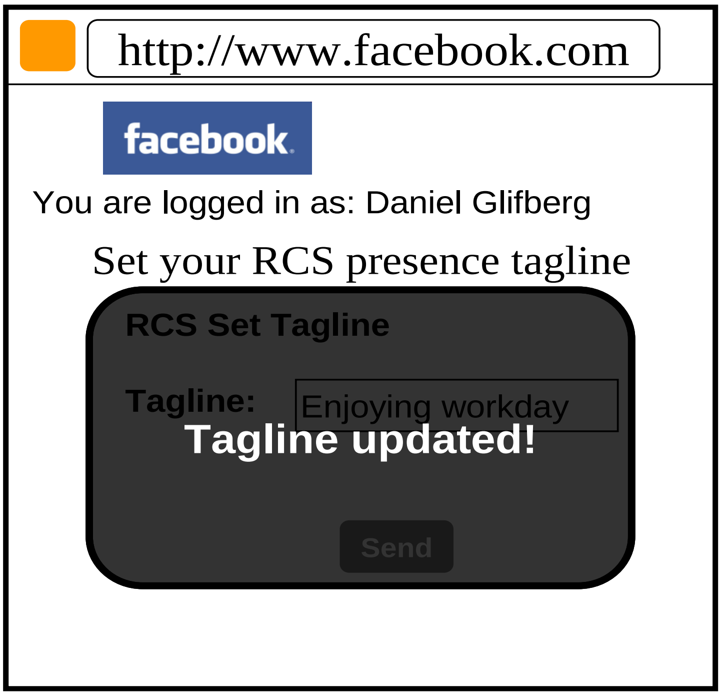
http://www.facebook.com

You are logged in as: Daniel Glifberg

Set your RCS presence tagline

Tagline: Enjoying workday

Your Service Provider: Orange...



http://www.facebook.com

You are logged in as: Daniel Glifberg

Set your RCS presence tagline

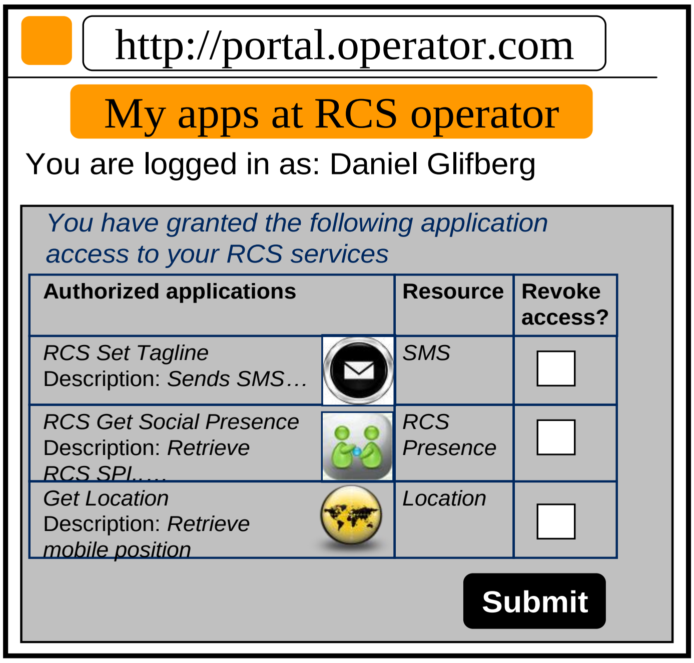
RCS Set Tagline

Tagline: Enjoying workday

Tagline updated!

5.5 Application authorization control – User view

- The user is managing which applications he has granted access to.
- The user can log on to his operator portal and get a list of applications he has 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*	□

6 Document Management

6.1 Document History

Version	Date	Brief Description of Change	Approval Authority	Editor / Company
Draft 0.1	28 Dec 2011	Merged RCS API 1.1 and RCS-e 1.0 requirement documents		Sergio Garcia Telefonica
Draft 1.0	11 Apr 2012	Baseline from merged draft document Including RCCAPI Doc CR_001 RCCAPI Doc CR_002		Jose M Recio Solaiemes
Draft 1.1	12 Apr 2012	Changes after RCCAPI #3		Jose M Recio Solaiemes
Draft 1.2	20 Apr 2012	Changes after RCCAPI #4 and email discussions Including RCCAPI Doc CR 003 rev 1 RCCAPI Doc CR 004 rev 1 RCCAPI Doc CR 005 rev 1 RCCAPI Doc CR 007 rev 1.2		Jose M Recio Solaiemes
Draft 1.3	24 Apr 2012	Editorial changes after email discussions		Jose M Recio Solaiemes
Draft 1.4	24 Apr 2012	Cleaning up figures		Jose M Recio Solaiemes
Draft 1.5	29 Apr 2012	Changes after RCCTF#25 Including RCCAPI Doc CR 006 rev 3 RCCAPI Doc CR 008 rev 1 RCCAPI Doc CR 009 rev 2 RCCAPI Doc CR 010 rev 1		Jose M Recio Solaiemes
Draft 1.6	21 May 2012	Changes after email discussion and comments from NSN, Ericsson, Interop, Verizon and other contributors		Jose M Recio Solaiemes
Draft 1.7	26 May 2012	Editorials after email discussion		Jose M Recio Solaiemes
2.0	26 May 2012	Generated from Draft 1.7		Jose M Recio Solaiemes
2.1	5 July 2012	Editorial corrections in reqs UNI-CHT-002b and UNI-CHT-001		Jose M Recio Solaiemes

6.2 Other Information

Type	Description
Document owner	Rich Communication Suite Programme
Editor/company	Jose M Recio / Solaiemes