



RESTful Network API for Notification Channel

Candidate Version 1.0 – 22 Dec 2015

Open Mobile Alliance
OMA-TS-REST_NetAPI_NotificationChannel-V1_0-20151222-C

Use of this document is subject to all of the terms and conditions of the Use Agreement located at <http://www.openmobilealliance.org/UseAgreement.html>.

Unless this document is clearly designated as an approved specification, this document is a work in process, is not an approved Open Mobile Alliance™ specification, and is subject to revision or removal without notice.

You may use this document or any part of the document for internal or educational purposes only, provided you do not modify, edit or take out of context the information in this document in any manner. Information contained in this document may be used, at your sole risk, for any purposes. You may not use this document in any other manner without the prior written permission of the Open Mobile Alliance. The Open Mobile Alliance authorizes you to copy this document, provided that you retain all copyright and other proprietary notices contained in the original materials on any copies of the materials and that you comply strictly with these terms. This copyright permission does not constitute an endorsement of the products or services. The Open Mobile Alliance assumes no responsibility for errors or omissions in this document.

Each Open Mobile Alliance member has agreed to use reasonable endeavors to inform the Open Mobile Alliance in a timely manner of Essential IPR as it becomes aware that the Essential IPR is related to the prepared or published specification. However, the members do not have an obligation to conduct IPR searches. The declared Essential IPR is publicly available to members and non-members of the Open Mobile Alliance and may be found on the “OMA IPR Declarations” list at <http://www.openmobilealliance.org/ipr.html>. The Open Mobile Alliance has not conducted an independent IPR review of this document and the information contained herein, and makes no representations or warranties regarding third party IPR, including without limitation patents, copyrights or trade secret rights. This document may contain inventions for which you must obtain licenses from third parties before making, using or selling the inventions. Defined terms above are set forth in the schedule to the Open Mobile Alliance Application Form.

NO REPRESENTATIONS OR WARRANTIES (WHETHER EXPRESS OR IMPLIED) ARE MADE BY THE OPEN MOBILE ALLIANCE OR ANY OPEN MOBILE ALLIANCE MEMBER OR ITS AFFILIATES REGARDING ANY OF THE IPR'S REPRESENTED ON THE “OMA IPR DECLARATIONS” LIST, INCLUDING, BUT NOT LIMITED TO THE ACCURACY, COMPLETENESS, VALIDITY OR RELEVANCE OF THE INFORMATION OR WHETHER OR NOT SUCH RIGHTS ARE ESSENTIAL OR NON-ESSENTIAL.

THE OPEN MOBILE ALLIANCE IS NOT LIABLE FOR AND HEREBY DISCLAIMS ANY DIRECT, INDIRECT, PUNITIVE, SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR EXEMPLARY DAMAGES ARISING OUT OF OR IN CONNECTION WITH THE USE OF DOCUMENTS AND THE INFORMATION CONTAINED IN THE DOCUMENTS.

© 2015 Open Mobile Alliance Ltd. All Rights Reserved.

Used with the permission of the Open Mobile Alliance Ltd. under the terms set forth above.

Contents

| | |
|---|-----------|
| 1. SCOPE | 8 |
| 2. REFERENCES | 9 |
| 2.1 NORMATIVE REFERENCES | 9 |
| 2.2 INFORMATIVE REFERENCES | 9 |
| 3. TERMINOLOGY AND CONVENTIONS | 11 |
| 3.1 CONVENTIONS | 11 |
| 3.2 DEFINITIONS | 11 |
| 3.3 ABBREVIATIONS | 11 |
| 4. INTRODUCTION | 13 |
| 4.1 VERSION 1.0 | 13 |
| 5. NOTIFICATION CHANNEL API DEFINITION | 14 |
| 5.1 RESOURCES SUMMARY | 16 |
| 5.2 DATA TYPES | 20 |
| 5.2.1 XML Namespaces..... | 20 |
| 5.2.2 Structures..... | 20 |
| 5.2.2.1 <i>Type: NotificationChannelList</i> | 20 |
| 5.2.2.2 <i>Type: NotificationChannel</i> | 20 |
| 5.2.2.3 <i>Type: NotificationList</i> | 22 |
| 5.2.2.4 <i>Type: LargePollingNotification</i> | 22 |
| 5.2.2.5 <i>Type: ChannelData</i> | 23 |
| 5.2.2.6 <i>Type: LongPollingData</i> | 23 |
| 5.2.2.7 <i>Type: OMAPushData</i> | 23 |
| 5.2.2.8 <i>Type: LargeDataPolling</i> | 24 |
| 5.2.2.9 <i>Type: LongPollingRequestParameters</i> | 24 |
| 5.2.2.10 <i>Type: WebSocketsData</i> | 25 |
| 5.2.2.11 <i>Type: ConnCheck</i> | 25 |
| 5.2.2.12 <i>Type: ConnAck</i> | 25 |
| 5.2.2.13 <i>Type: NotificationChannelLifetime</i> | 26 |
| 5.2.3 Enumerations..... | 26 |
| 5.2.3.1 <i>Enumeration: ChannelType</i> | 26 |
| 5.2.4 Values of the Link “rel” attribute..... | 26 |
| 5.3 SEQUENCE DIAGRAMS | 27 |
| 5.3.1 Create Notification Channel (Long Polling Method)..... | 27 |
| 5.3.2 Notifications delivered to application using Long Polling..... | 28 |
| 5.3.3 Long Polling request timeout response..... | 29 |
| 5.3.4 Multiple notifications delivered to application in response to the Long Polling request..... | 30 |
| 5.3.5 Max number of notifications reached during the Long Polling..... | 31 |
| 5.3.6 Max wait time or max number of notifications reached during the Long Polling..... | 32 |
| 5.3.7 Create Notification Channel (OMA Push Method)..... | 35 |
| 5.3.8 Notifications delivered to application using OMA Push..... | 35 |
| 5.3.9 Create Notification Channel (OMA Push method with Large Data Polling enabled)..... | 37 |
| 5.3.10 Notifications delivered to application using OMA Push while Large Data Polling is enabled..... | 38 |
| 5.3.11 Create Notification Channel (WebSockets)..... | 40 |
| 5.3.12 Notifications delivered to application using WebSockets..... | 41 |
| 5.3.13 Refreshing a Notification Channel..... | 42 |
| 5.3.14 Client-initiated ConnCheck/ConnAck..... | 43 |
| 5.3.15 Successful server-initiated ConnCheck/ConnAck..... | 44 |
| 5.3.16 Unsuccessful server-initiated ConnCheck/ConnAck..... | 45 |
| 5.3.17 Notifications delivered to application using Long Polling..... | 45 |
| 6. DETAILED SPECIFICATION OF THE RESOURCES | 47 |
| 6.1 RESOURCE: NOTIFICATION CHANNELS | 47 |
| 6.1.1 Request URL variables..... | 47 |
| 6.1.2 Response Codes and Error Handling..... | 48 |

| | | |
|------------|---|-----------|
| 6.1.3 | GET | 48 |
| 6.1.3.1 | Example: Retrieve active Notification Channels (Informative) | 48 |
| 6.1.3.1.1 | Request | 48 |
| 6.1.3.1.2 | Response | 48 |
| 6.1.4 | PUT | 49 |
| 6.1.5 | POST | 49 |
| 6.1.5.1 | Example: Create Notification Channel (Long Polling method), using tel URI (Informative) | 49 |
| 6.1.5.1.1 | Request | 49 |
| 6.1.5.1.2 | Response | 49 |
| 6.1.5.2 | Example: Create Notification Channel (OMA Push method), using tel URI (Informative) | 50 |
| 6.1.5.2.1 | Request | 50 |
| 6.1.5.2.2 | Response | 50 |
| 6.1.5.3 | Example: Create Notification Channel (OMA Push method with largeDataPolling), using tel URI (Informative) | 50 |
| 6.1.5.3.1 | Request | 50 |
| 6.1.5.3.2 | Response | 51 |
| 6.1.5.4 | Example: Create Notification Channel (OMA Push method with LargeDataPolling) not supported (Informative) | 51 |
| 6.1.5.4.1 | Request | 51 |
| 6.1.5.4.2 | Response | 52 |
| 6.1.5.5 | Example: Create Notification Channel (Long Polling method), using ACR (Informative) | 52 |
| 6.1.5.5.1 | Request | 52 |
| 6.1.5.5.2 | Response | 53 |
| 6.1.5.6 | Example: Create Notification Channel (WebSockets method), using tel URI (Informative) | 53 |
| 6.1.5.6.1 | Request | 53 |
| 6.1.5.6.2 | Response | 53 |
| 6.1.5.7 | Example: Attempt to create Notification Channel of unsupported type (Informative) | 54 |
| 6.1.5.7.1 | Request | 54 |
| 6.1.5.7.2 | Response | 54 |
| 6.1.6 | DELETE | 55 |
| 6.2 | RESOURCE: INDIVIDUAL NOTIFICATION CHANNEL | 55 |
| 6.2.1 | Request URL variables | 55 |
| 6.2.2 | Response Codes and Error Handling | 55 |
| 6.2.3 | GET | 55 |
| 6.2.3.1 | Example: Retrieve individual Notification Channel (Informative) | 55 |
| 6.2.3.1.1 | Request | 55 |
| 6.2.3.1.2 | Response | 55 |
| 6.2.4 | PUT | 56 |
| 6.2.5 | POST | 56 |
| 6.2.6 | DELETE | 56 |
| 6.2.6.1 | Example: Removing Notification Channel (Informative) | 56 |
| 6.2.6.1.1 | Request | 56 |
| 6.2.6.1.2 | Response | 56 |
| 6.3 | RESOURCE: NOTIFICATION LIST | 56 |
| 6.3.1 | Request URL variables | 57 |
| 6.3.2 | Response Codes and Error Handling | 57 |
| 6.3.3 | GET | 57 |
| 6.3.4 | PUT | 57 |
| 6.3.5 | POST | 57 |
| 6.3.5.1 | Example 1: Single notification delivered in a NotificationList (Informative) | 57 |
| 6.3.5.1.1 | Request | 57 |
| 6.3.5.1.2 | Response | 57 |
| 6.3.5.2 | Example 2: Multiple notifications delivered (Informative) | 58 |
| 6.3.5.2.1 | Request | 58 |
| 6.3.5.2.2 | Response | 58 |
| 6.3.5.3 | Example 3: Server timeout (Informative) | 59 |
| 6.3.5.3.1 | Request | 59 |
| 6.3.5.3.2 | Response | 59 |
| 6.3.5.4 | Example 4: Single notification delivered outside a NotificationList (Informative) | 60 |
| 6.3.5.4.1 | Request | 60 |
| 6.3.5.4.2 | Response | 60 |
| 6.3.6 | DELETE | 60 |
| 6.4 | RESOURCE: NOTIFICATION CHANNEL LIFETIME | 60 |

| | | |
|--------------------|--|-----------|
| 6.4.1 | Request URL variables | 60 |
| 6.4.2 | Response Codes and Error Handling | 61 |
| 6.4.3 | GET | 61 |
| 6.4.3.1 | Example: Retrieve remaining Notification Channel lifetime (Informative)..... | 61 |
| 6.4.3.1.1 | Request..... | 61 |
| 6.4.3.1.2 | Response..... | 61 |
| 6.4.4 | PUT | 61 |
| 6.4.4.1 | Example: Update Notification Channel lifetime (Informative)..... | 61 |
| 6.4.4.1.1 | Request..... | 61 |
| 6.4.4.1.2 | Response..... | 62 |
| 6.4.5 | POST..... | 62 |
| 6.4.6 | DELETE | 62 |
| 7. | FAULT DEFINITIONS..... | 63 |
| 7.1 | SERVICE EXCEPTIONS..... | 63 |
| 7.1.1 | SVC1012: Simultaneous channel requests not supported..... | 63 |
| 7.2 | POLICY EXCEPTIONS | 63 |
| 7.2.1 | POL1023: Notification channel type not supported..... | 63 |
| APPENDIX A. | CHANGE HISTORY (INFORMATIVE)..... | 64 |
| A.1 | APPROVED VERSION HISTORY | 64 |
| A.2 | DRAFT/CANDIDATE VERSION 1.0 HISTORY | 64 |
| APPENDIX B. | STATIC CONFORMANCE REQUIREMENTS (NORMATIVE)..... | 68 |
| B.1 | SCR FOR REST.NC SERVER..... | 68 |
| B.1.1 | SCR for REST.NC.Channels Server | 68 |
| B.1.2 | SCR for REST.NC.IndividualChannel Server | 69 |
| B.1.3 | SCR for REST.NC.LongPolling Server | 69 |
| B.1.4 | SCR for REST.NC.OMAPush Server..... | 69 |
| B.1.5 | SCR for REST.NC.Refresh Server | 69 |
| B.1.6 | SCR for REST.NC.WebSockets Server..... | 69 |
| APPENDIX C. | APPLICATION/X-WWW-FORM-URLENCODED REQUEST FORMAT FOR POST OPERATIONS (NORMATIVE) | 70 |
| C.1 | CREATING A NOTIFICATION CHANNEL | 70 |
| C.1.1 | Example 1: Create Notification Channel (Long Polling method), using tel URI (Informative) | 71 |
| C.1.1.1 | Request..... | 71 |
| C.1.1.2 | Response | 71 |
| C.1.2 | Example 2: Create Notification Channel (OMA Push method), using tel URI (Informative) | 72 |
| C.1.2.1 | Request..... | 72 |
| C.1.2.2 | Response | 72 |
| C.1.3 | Example 3: Create Notification Channel, using ACR (Informative) | 72 |
| C.1.3.1 | Request..... | 72 |
| C.1.3.2 | Response | 73 |
| C.2 | RETRIEVING NOTIFICATIONS FROM THE NOTIFICATION SERVER | 73 |
| C.2.1 | Example 1: Single notification delivered in a NotificationList (Informative) | 73 |
| C.2.1.1 | Request..... | 73 |
| C.2.1.2 | Response | 74 |
| C.2.2 | Example 2: Single notification delivered outside a NotificationList (Informative) | 74 |
| C.2.2.1 | Request..... | 74 |
| C.2.2.2 | Response | 74 |
| APPENDIX D. | JSON EXAMPLES (INFORMATIVE) | 76 |
| D.1 | RETRIEVE ACTIVE NOTIFICATION CHANNELS (SECTION 6.1.3.1) | 76 |
| D.2 | CREATE NOTIFICATION CHANNEL (LONG POLLING METHOD), USING TEL URI (SECTION 6.1.5.1)..... | 77 |
| D.3 | CREATE NOTIFICATION CHANNEL (OMA PUSH METHOD), USING TEL URI (SECTION 6.1.5.2) | 78 |
| D.4 | CREATE NOTIFICATION CHANNEL (OMA PUSH METHOD WITH LARGEDATAPOLLING), USING TEL URI (SECTION 6.1.5.3)..... | 78 |
| D.5 | CREATE NOTIFICATION CHANNEL (OMA PUSH METHOD WITH LARGEDATAPOLLING) NOT SUPPORTED (SECTION 6.1.5.4)..... | 79 |
| D.6 | CREATE NOTIFICATION CHANNEL (LONG POLLING METHOD), USING ACR (SECTION 6.1.5.5)..... | 80 |

| | | |
|-------------|---|-----|
| D.7 | CREATE NOTIFICATION CHANNEL (WEBSOCKETS METHOD), USING TEL URI (SECTION 6.1.5.6) | 81 |
| D.8 | EXAMPLE: ATTEMPT TO CREATE NOTIFICATION CHANNEL OF UNSUPPORTED TYPE (SECTION 6.1.5.7) | 82 |
| D.9 | RETRIEVE INDIVIDUAL NOTIFICATION CHANNEL (SECTION 6.2.3.1) | 82 |
| D.10 | REMOVING NOTIFICATION CHANNEL (SECTION 6.2.6.1) | 83 |
| D.11 | SINGLE NOTIFICATION DELIVERED IN A NOTIFICATIONLIST (SECTION 6.3.5.1) | 83 |
| D.12 | MULTIPLE NOTIFICATIONS DELIVERED (SECTION 6.3.5.2) | 84 |
| D.13 | SERVER TIMEOUT (SECTION 6.3.5.3) | 85 |
| D.14 | SINGLE NOTIFICATION DELIVERED IN A NOTIFICATIONLIST (SECTION 6.3.5.4) | 85 |
| D.15 | RETRIEVE REMAINING NOTIFICATION CHANNEL LIFETIME (SECTION 6.4.3.1) | 86 |
| D.16 | UPDATE NOTIFICATION CHANNEL LIFETIME (SECTION 6.4.4.1) | 86 |
| APPENDIX E. | OPERATIONS MAPPING TO A PRE-EXISTING BASELINE SPECIFICATION (INFORMATIVE) | 87 |
| APPENDIX F. | LIGHT-WEIGHT RESOURCES (INFORMATIVE) | 88 |
| APPENDIX G. | AUTHORIZATION ASPECTS (NORMATIVE) | 89 |
| G.1 | USE WITH OMA AUTHORIZATION FRAMEWORK FOR NETWORK APIS | 89 |
| G.1.1 | Scope values | 89 |
| G.1.1.1 | Definitions | 89 |
| G.1.1.2 | Downscoping | 89 |
| G.1.1.3 | Mapping with resources and methods | 90 |
| G.1.2 | Use of 'acr:auth' | 92 |
| APPENDIX H. | NOTIFICATION SERVER - PUSH ENABLER INTERACTION (INFORMATIVE) | 93 |
| APPENDIX I. | NOTIFICATION DELIVERY USING WEBSOCKETS (NORMATIVE) | 97 |
| I.1 | DELIVERY MECHANISM | 97 |
| I.2 | SUBPROTOCOL REGISTRATION | 97 |
| I.3 | CONNECTION CHECKING AND KEEP-ALIVE | 97 |
| I.4 | NOTIFICATION PAYLOAD EXAMPLES – XML FORMAT (INFORMATIVE) | 98 |
| I.4.1 | Example: Single notification delivered in a NotificationList | 98 |
| I.4.2 | Example: Multiple notifications delivered | 98 |
| I.4.3 | Example: Single notification delivered outside a NotificationList | 99 |
| I.5 | NOTIFICATION PAYLOAD EXAMPLES – JSON (INFORMATIVE) | 100 |
| I.5.1 | Single notification delivered in a NotificationList | 100 |
| I.5.2 | Multiple notifications delivered | 100 |
| I.5.3 | Single notification delivered in a NotificationList | 101 |

Figures

| | | |
|-----------|--|----|
| Figure 1 | Resource structure defined by this specification | 17 |
| Figure 2 | Create Notification Channel | 27 |
| Figure 3 | Notifications delivered to application | 28 |
| Figure 4 | Request timeout | 29 |
| Figure 5 | Multiple notifications delivered to application in response | 30 |
| Figure 6 | Maximum number of notifications in the response to the Long Polling | 31 |
| Figure 7 | Max wait time or max number of notifications criterion used to respond to the Long Polling | 33 |
| Figure 8 | Create Notification Channel (OMA Push Method) | 35 |
| Figure 9 | Notifications delivered to application using OMA Push | 36 |
| Figure 10 | Create Notification Channel (OMA Push method with Large Data Polling enabled) | 37 |

| | |
|--|----|
| Figure 11 Notifications delivered to application using OMA Push | 38 |
| Figure 12 Create Notification Channel (WebSockets)..... | 40 |
| Figure 13 Notifications delivered to application using WebSockets..... | 41 |
| Figure 14 Notification Channel refresh | 42 |
| Figure 15 Client-initiated ConnCheck/ConnAck for session keep-alive | 43 |
| Figure 16 Successful server-initiated ConnCheck/ConnAck for session keep-alive | 44 |
| Figure 17 Unsuccessful server-initiated ConnCheck/ConnAck for session keep-alive | 45 |

Tables

| | |
|--|----|
| Table 1: Scope values for RESTful Notification Channel API | 89 |
| Table 2: Required scope values for: Management of Notification Channel | 91 |
| Table 3: Required scope values for: Retrieval of notifications from Notification Server..... | 91 |

1. Scope

This specification defines a RESTful API for Notification Channel using HTTP protocol bindings.

2. References

2.1 Normative References

- [OMA_PUSH] “OMA Push 2.3” Open Mobile Alliance™, OMA-ERP-Push-V2_3
URL:<http://www.openmobilealliance.org/>
- [REST_NetAPI_ACR] “RESTful Network API for Anonymous Customer Reference Management”, Open Mobile Alliance™, OMA-TS-REST_NetAPI_ACR-V1_0, URL: <http://www.openmobilealliance.org/>
- [REST_NetAPI_Common] “Common definitions for RESTful Network APIs”, Open Mobile Alliance™, OMA-TS-REST_NetAPI_Common-V1_0, URL:<http://www.openmobilealliance.org/>
- [REST_SUP_NotificationChannel] “XML schema for the RESTful Network API for Notification Channel”, Open Mobile Alliance™, OMA-SUP-XSD_rest_netapi_notificationchannel-V1_0, URL:<http://www.openmobilealliance.org/>
- [RFC2119] “Key words for use in RFCs to Indicate Requirement Levels”, S. Bradner, March 1997, URL:<http://www.ietf.org/rfc/rfc2119.txt>
- [RFC3261] “SIP: Session Initiation Protocol”, J. Rosenberg et al., June 2002, URL:<http://www.rfc-editor.org/rfc/rfc3261.txt>
- [RFC3966] “The tel URI for Telephone Numbers”, H.Schulzrinne, December 2004, URL:<http://www.ietf.org/rfc/rfc3966.txt>
- [RFC3986] “Uniform Resource Identifier (URI): Generic Syntax”, R. Fielding et. al, January 2005, URL:<http://www.ietf.org/rfc/rfc3986.txt>
- [RFC6455] “The WebSocket Protocol”, I. Fette and A. Melnikov, December 2011, URL:<http://www.ietf.org/rfc/rfc6455.txt>
- [RFC7159] “The JavaScript Object Notation (JSON) Data Interchange Format”, T. Bray, Ed., March 2014, URL:<http://tools.ietf.org/html/rfc7159.txt>
- [RFC7231] “Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content”, R. Fielding, Ed., J.Raschke, Ed., June 2014, URL:<http://tools.ietf.org/html/rfc7231.txt>
- [SCRRULES] “SCR Rules and Procedures”, Open Mobile Alliance™, OMA-ORG-SCR_Rules_and_Procedures, URL:<http://www.openmobilealliance.org/>
- [W3C_URLENC] HTML 4.01 Specification, Section 17.13.4 Form content types, The World Wide Web Consortium, URL:<http://www.w3.org/TR/html401/interact/forms.html#h-17.13.4.1>
- [XMLSchema1] W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures, W3C Recommendation 5 April 2012, URL: <http://www.w3.org/TR/xmlschema11-1/>
- [XMLSchema2] W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes, W3C Recommendation 5 April 2012, URL: <http://www.w3.org/TR/xmlschema11-2/>

2.2 Informative References

- [IANA] Protocol Assignments, Internet Assigned Numbers Authority, URL:<http://www.iana.org>
- [OMADICT] “Dictionary for OMA Specifications”, Version 2.9, Open Mobile Alliance™, OMA-ORG-Dictionary-V2_9, URL:<http://www.openmobilealliance.org/>
- [PushOTA] “Push Over-the-Air”, Open Mobile Alliance™, OMA-TS-PushOTA-V2_3, URL:<http://www.openmobilealliance.org/>
- [PushPAP] “Push Access Protocol”, Open Mobile Alliance™, OMA-TS-PAP-V2_3, URL:<http://www.openmobilealliance.org/>
- [PushREST] “RESTful Network API for OMA Push”, Open Mobile Alliance™, OMA-TS-REST_NetAPI_Push-V1_0, URL:<http://www.openmobilealliance.org/>
- [REST_WP] “Guidelines for RESTful Network APIs”, Open Mobile Alliance™, OMA-WP-Guidelines_for_RESTful_Network_APIs, URL:<http://www.openmobilealliance.org/>

- [RFC6202] “Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP”, April 2011, URL:<http://tools.ietf.org/rfc/rfc6202.txt>
- [W3C_WebSock] “The WebSocket API”, W3C Candidate Recommendation 20 September 2012, Ian Hickson, ed., URL:<http://www.w3.org/TR/websockets/>

3. Terminology and Conventions

3.1 Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

All sections and appendixes, except “Scope” and “Introduction”, are normative, unless they are explicitly indicated to be informative.

3.2 Definitions

For the purpose of this TS, all definitions from the OMA Dictionary apply [OMADICT].

| | |
|-------------------------------------|--|
| Client-side Notification URL | An HTTP URL exposed by a client, on which it is capable of receiving notifications and that can be used by the client when subscribing to notifications. |
| Long Polling | A variation of the traditional polling technique, where the server does not reply to a request unless a particular event, status or timeout has occurred. Once the server has sent a response, it closes the connection, and typically the client immediately sends a new request. This allows the emulation of an information push from a server to a client. |
| Notification Channel | A channel created on the request of the client and used to deliver notifications from a server to a client. The channel is represented as a resource and provides means for the server to post notifications and for the client to receive them via specified delivery mechanisms. For example in the case of Long Polling the channel resource is defined by a pair of URLs. One of the URLs is used by the client as a callback URL when subscribing for notifications. The other URL is used by the client to retrieve notifications from the Notification Server. |
| Notification Server | A server that is capable of creating and maintaining Notification Channels. |
| Server-side Notification URL | An HTTP URL exposed by a Notification Server, that identifies a Notification Channel and that can be used by a client when subscribing to notifications. |

3.3 Abbreviations

| | |
|-------------|---------------------------------------|
| ACR | Anonymous Customer Reference |
| API | Application Programming Interface |
| HTTP | HyperText Transfer Protocol |
| JSON | JavaScript Object Notation |
| MIME | Multipurpose Internet Mail Extensions |
| OMA | Open Mobile Alliance |
| PAP | Push Access Protocol |
| PPG | Push Proxy Gateway |
| REST | REpresentational State Transfer |
| SCR | Static Conformance Requirements |
| SIP | Session Initiation Protocol |
| TS | Technical Specification |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| WP | White Paper |

XML eXtensible Markup Language
XSD XML Schema Definition

4. Introduction

The Technical Specification for the RESTful Network API for Notification Channel contains HTTP protocol bindings for Notification Channel, using the REST architectural style. The specification provides resource definitions, the HTTP verbs applicable for each of these resources, and the element data structures, as well as support material including flow diagrams and examples using the various supported message body formats (i.e. XML, JSON, and application/x-www-form-urlencoded).

This specification defines Pull methods and Push methods to deliver the notifications to the client.

4.1 Version 1.0

Version 1.0 of this specification supports the following operations:

- Manage Notification Channel
- Retrieve asynchronous notifications from the Notification Server via Long Polling (a Pull method)
- Receive asynchronous notifications from the Notification Server via OMA Push (a Push method)
- Receive asynchronous notifications from the Notification Server via WebSockets (a Push method)

In addition, this specification provides:

- Support for scope values used with authorization framework defined in [Autho4API_10]
- Support for Anonymous Customer Reference (ACR) as an end user identifier
- Support for “acr:auth” as a reserved keyword in a resource URL variable that identifies an end user

5. Notification Channel API definition

This section is organized to support a comprehensive understanding of the Notification Channel API design. It specifies the definition of all resources, definition of all data structures, and definitions of all operations permitted on the specified resources.

This specification introduces methods for a client (e.g. a browser or a native application) to receive asynchronous notifications from a Notification Server about the events the client has subscribed to with one or more Enabler servers. The notification delivery methods specified in this document fall into two groups: Pull and Push methods. For Pull, the notification delivery method specified is based on HTTP requests and often referred as “HTTP Long Polling” [RFC6202]. For Push, two notification delivery methods are defined: WebSockets [RFC6455] and OMA Push [PUSH_ARCH]. For OMA Push delivery, this specification assumes the Notification Server, as a Push Initiator, knows how to interact with PPG using Push Access Protocol (PAP) [OMA PUSH] and as such not in the scope of this document.

For all notification delivery methods, as notifications are conveyed through a Notification Channel, the channel must be created first before any further interaction can be invoked, such as a Long Polling request invoked by the client, or an asynchronous event-push initiated by the channel onto PPG for OMA Push.

A single Notification Channel may handle notifications from several Enabler servers. Note that the client subscriptions to notifications are specific for each Enabler server and they are not in the scope of this specification.

The following applies selectively to the different types of notification channels.

1) Long Polling:

In response to a channel creation request containing `channelType = LongPolling`, the Notification Server will provide two URLs: callback URL and channel URL. The client uses callback URL as the notification endpoint when subscribing to notifications from the Enabler server(s). Thus, each Enabler server will send subsequent notifications using this callback URL referring to the Notification Server. The channel URL is used to retrieve notifications from the Notification Server using the HTTP Long Polling mechanism. When the Notification Server receives a notification from an Enabler server, it possibly groups multiple notifications prior to delivery, and conveys the notification(s) to the client with the response to the pending HTTP Long Polling request.

A Notification Channel has certain time-to-live and therefore in order to continue using it, the channel has to be maintained (“refreshed”) by the client. For the Long Polling delivery method, the channel is refreshed implicitly: With each Long Polling request, the Notification Server will reset the channel life time to its original value.

Clients SHOULD NOT establish more than one simultaneous connection to the channel URL, and servers SHOULD NOT allow more than one simultaneous connection to the channel URL. If another Long Polling request arrives at the Notification Server while a Long Polling request for the same channel URL is still open, the server SHOULD terminate the first request with an appropriate error (e.g., SVC1012 Simultaneous channel requests not supported), and pass notifications to the client in response to the newly arrived Long Polling request.

Clients SHOULD issue a new Long Polling request as quickly as possible after receiving a notification.

2) OMA Push:

In response to a channel creation request containing `channelType = OMAPush`, the Notification Server will only provide a callback URL. That is, for the OMA Push notification delivery method, the notification server does not provide a channel URL as the client application is expected to asynchronously receive notifications via the OMA Push enabler [OMA_PUSH]. As explained earlier above, the client application would use the callback URL as notification endpoint when subscribing to notifications from the Enabler server(s).

Additionally, the request for a channel creation of type OMA Push may contain a unique application Id (`appId`) which is required by the OMA Push infrastructure [OMA_PUSH] to direct the asynchronous push messages to a particular client application on the device. However, if the application Id is not present in the channel creation request, it is assumed that the Notification Server has other means of retrieving the application Id (e.g. through the usage of the available OAuth token in the Notification Channel creation request).

When the Notification Server receives a notification from an Enabler server, it possibly groups multiple notifications prior to delivery, and conveys the notification(s) to the client via the PPG.

When the client expects a high-traffic notifications behaviour from a given backend enabler and due to various reasons (e.g. resources limitation such as battery) cannot use a Long Polling method or cannot deal with the limited notification size of OMA Push delivery method (which requires notifications to be partitioned into many pieces for delivery over SMS which in itself has limits over the number SMSs per user), it can inform the server that it desires to retrieve notifications in a HTTP Polling manner (i.e. “largeDataPolling”) if the number of notifications is more than the specified “OMAPushData.maxNotifications”.

When the “largeDataPolling” mechanism which is a feature of OMA Push channel is enabled (i.e. “largeDataPolling.pollingEnabled” = true) by the client, any time the number of notifications is more than the specified “OMAPushData.maxNotifications” or the notification size is beyond the known limitation (e.g. 2K uncompressed bytes) of OMA Push delivery method or certain server policy is met, the server informs the client via a Push MESSAGE that a large set of notifications is queued up to be retrieved. This Push MESSAGE contains a LargePollingNotification which itself contains a dynamically created “channelURL”.

The “channelURL” is used by the client to retrieve notifications from the Notification Server using a HTTP Polling mechanism which is similar to the Long Polling explained above with the difference that once the queued up notifications have all been fetched from the “channelURL”, the server marks the notificationList as complete (by setting the “ncListComplete” parameter to true) and destroys the “channelURL” accordingly. At this point the dynamically created channelURL is no longer valid (hence the client should stop polling it) while the OMA Push channel continues its normal life cycle.

Depending on the number of awaiting events in the channel and the requested “maxPollingNotifications”, the client may repeat polling on channelURL multiple times in order to retrieve all the awaiting notifications.

A Notification Channel has certain time-to-live and therefore in order to continue using it, the channel has to be maintained (“refreshed”) by the client. For this purpose, a resource is provided that the application can use to explicitly refresh the channel.

3) WebSockets:

In response to a channel creation request containing `channelType = WebSockets`, the Notification Server will provide a callback URL and a channel URL. The client uses the callback URL as notification endpoint when subscribing to notifications from the Enabler server(s). Thus, each Enabler server will send subsequent notifications using this callback URL referring to the Notification Server. The channel URL is used to establish a WebSockets connection to receive notifications from the Notification Server, whereas the transmission of a (set of) notification(s) is initiated by the Notification Server.

When the Notification Server receives a notification from an Enabler server, it possibly groups multiple notifications prior to delivery, and conveys the notification(s) to the client in the server-to-client leg of the bidirectional WebSockets connection. The client-to-server leg of the connection is currently unused except for connectivity checking and keep-alive.

A Notification Channel has certain time-to-live and therefore in order to continue using it, the channel has to be maintained (“refreshed”) by the client. For this purpose, a resource is provided that the application can use to explicitly refresh the channel (see section 6.4). Alternatively, the mechanism for connection checking and keep-alive defined in this specification (Appendix I.3) can be used for refresh.

It should be noted that in order not to disclose underlying network topology, the Notification Server usually sends to the client a mapped version of the real callback URL. Later, when the Enabler server receives such mapped callback URL, it will apply de-mapping of the URL before it can be used. How this mapping and de-mapping is performed on the server is out of scope for this specification.

The remainder of this document is structured as follows:

Section 5 starts with a diagram representing the resources hierarchy, followed by a table listing all the resources (and their URL) used by this API, along with the data structure and the supported HTTP verbs (section 5.1). What follows are the data structures (section 5.2). A sample of typical use cases is included in section 5.3, described as high level flow diagrams.

Section 6 contains detailed specification for each of the resources. Each such subsection defines the resource, the request URL variables that are common for all HTTP commands, the possible HTTP response codes, and the supported HTTP verbs. For each supported HTTP verb, a description of the functionality is provided, along with an example of a request and an example of a response. For each unsupported HTTP verb, the returned HTTP error status is specified, as well as what should be returned in the Allow header.

All examples in section 6 use XML as the format for the message body. `Application/x-www-form-urlencoded` examples are provided in Appendix C, while JSON examples are provided in Appendix D.

Section 7 contains fault definition details such as Service Exceptions and Policy Exceptions. Appendix B provides the Static Conformance Requirements (SCR).

Appendix E provides the operations mapping to a pre-existing baseline specification, where applicable.

Appendix F provides a list of all light-weight resources, where applicable.

Appendix G defines authorization aspects to control access to the resources defined in this specification.

Note: Throughout this document client and application can be used interchangeably.

5.1 Resources Summary

This section summarizes all the resources used by the RESTful Notification Channel API.

The "apiVersion" URL variable SHALL have the value "v1" to indicate that the API corresponds to this version of the specification. See [REST_NetAPI_Common] which specifies the semantics of this variable.

The figure below visualizes the resource structure defined by this specification. Note that those nodes in the resource tree which have associated HTTP methods defined in this specification are depicted by solid boxes.

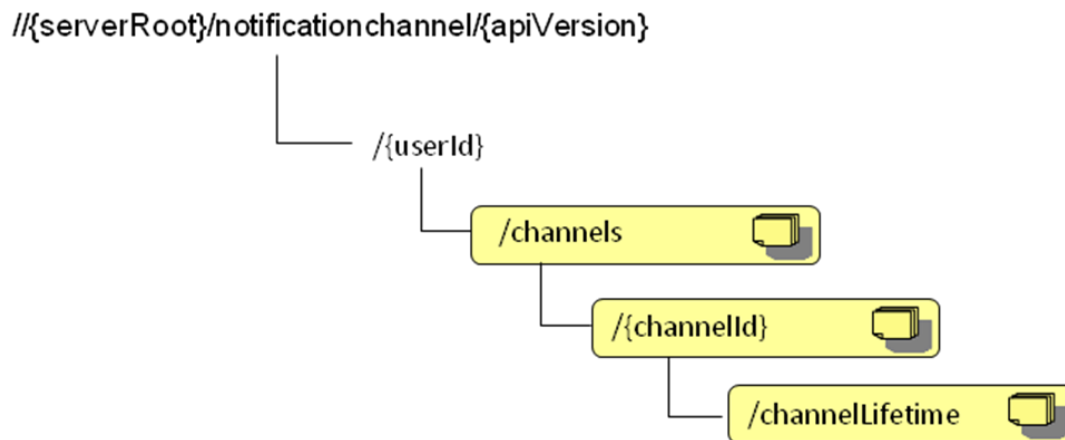


Figure 1 Resource structure defined by this specification

The following tables give a detailed overview of the resources defined in this specification, the data type of their representation and the allowed HTTP methods.

Purpose: To allow the client to manage Notification Channels

| Resource | URL Base URL: http://{serverRoot}/notificationchannel/{apiVersion} | Data Structures | HTTP verbs | | | |
|---------------------------------|--|---|---|--|-------------------------------------|---|
| | | | GET | PUT | POST | DELETE |
| Notification Channels | /userId/channels | NotificationChannelList (used for GET) NotificationChannel (used for POST) | Retrieves a list of Notification Channels. | no | Creates a new Notification Channel. | no |
| Individual Notification Channel | /userId/channels/{channelId} | NotificationChannel (used for GET) | Retrieves an individual Notification Channel. | no | no | Removes an individual Notification Channel. |
| Notification Channel lifetime | /userId/channels/{channelId}/channelLifetime | NotificationChannelLifetime | Retrieves the lifetime of a Notification Channel. | Updates ("refreshes") the lifetime of a Notification Channel | no | no |

Purpose: To allow the client to retrieve notifications from the Notification Server by using Long Polling

| Resource | URL: <specified by the server> | Data Structures | HTTP verbs | | | |
|-------------------|---|--|------------|-----|---|--------|
| | | | GET | PUT | POST | DELETE |
| Notification list | < Resource URL is received with "channelURL" in response from the server when a Long Polling Notification Channel is created> | LongPollingRequestParameters (used for POST request) NotificationList or notification as defined by individual enabler specification (used in response to the Long Polling POST request) | no | no | Retrieves pending notifications from the identified Long Polling Notification Channel. At the same time the channel life time is reset to its original value. | no |

Note: The URL of this resource is used by WebSockets-based notification channels to create the WebSockets connection through which the server can send notifications to the client. When using WebSockets, this is however not a resource in the RESTful sense; therefore, WebSockets-based notification channels are not mentioned in the table above.

5.2 Data Types

5.2.1 XML Namespaces

The XML namespace for the Notification Channel data types is:

urn:oma:xml:rest:netapi:notificationchannel:1

The 'xsd' namespace prefix is used in the present document to refer to the XML Schema data types defined in XML Schema [XMLSchema1, XMLSchema2]. The 'common' namespace prefix is used in the present document to refer to the data types defined in [REST_NetAPI_Common]. The use of namespace prefixes such as 'xsd' is not semantically significant.

The XML schema for the data structures defined in the section below is given in [REST_SUP_NotificationChannel].

5.2.2 Structures

The subsections of this section define the data structures used in the Notification Channel API.

Some of the structures can be instantiated as so-called root elements.

5.2.2.1 Type: NotificationChannelList

This type defines a list of Notification Channels.

| Element | Type | Optional | Description |
|---------------------|---------------------------------------|----------|--|
| notificationChannel | NotificationChannel [0..unbounded] | Yes | May contain an array of Notification Channels. |
| resourceURL | xsd:anyURI | No | Self referring URL |

A root element named notificationChannelList of type NotificationChannelList is allowed in response bodies.

5.2.2.2 Type: NotificationChannel

This type defines a single Notification Channel.

| Element | Type | Optional | Description |
|------------------|------------|----------|---|
| clientCorrelator | xsd:string | Yes | <p>A correlator that the client can use to tag this particular resource representation during a request to create a resource on the server.</p> <p>This element SHOULD be present. Note: this allows the client to recover from communication failures during resource creation and therefore avoids duplicate channel creation in such situations.</p> <p>In case the field is present, the server SHALL not alter its value, and SHALL provide it as part of the representation of this resource. In case the field is not present, the server SHALL NOT generate it.</p> |
| applicationTag | xsd:string | Yes | <p>A tag that the client MAY use to tag this particular resource on the server. In case the field is present, the server SHALL not alter its value, and SHALL provide it as part of the representation of this resource. In case the field is not present, the server SHALL NOT generate it.</p> |

| | | | |
|-----------------|-------------|-----|---|
| channelType | ChannelType | No | Specifies the type of Notification Channel to be used and thereby defines the method that will be used to receive new notifications on the channel. |
| channelData | ChannelData | Yes | <p>Contains specific information for the Notification Channel type specified in channelType.</p> <p>The channelData MUST be included in the response to the request for the creation of Notification Channel for Long Polling, OMA Push or WebSockets.</p> <p>Note that for Long Polling, the channel data is defined in the type LongPollingData (see 5.2.2.6). For OMA Push, the channel data is defined in the type OMAPushData (see 5.2.2.7). For WebSockets, the channel data is defined in the type WebSocketsData (see 5.2.2.10). All these data types are derived from ChannelData.</p> <p>In XML implementation for channelData, LongPollingData, OMAPushData or WebSocket Data, the type is identified by the xsi:type attribute.</p> |
| channelLifetime | xsd:int | Yes | <p>Lifetime (duration) of Notification Channel in seconds.</p> <p>Client can specify desired lifetime of Notification Channel in POST request when creating Notification Channel, however the server in the response to the request may change the desired lifetime according to its server policy.</p> <p>If the element is not present in the POST request, a default channel lifetime specified by server policy will apply.</p> <p>The server SHALL always include the channel lifetime in the response.</p> |
| callbackURL | xsd:anyURI | Yes | Specified by the server. Contains a callback URL used when establishing subscriptions for notifications from the respective Enabler server (not part of this specification). The callbackURL SHALL NOT be included in POST request to create the Notification Channel resource. MUST be included in responses to the channel creation and any HTTP method that returns an entity body. |
| resourceURL | xsd:anyURI | Yes | Self referring URL. The resourceURL SHALL NOT be included in POST requests by the client, but MUST be included in POST requests representing notifications by the server to the client, when a complete representation of the resource is embedded in the notification. The resourceURL MUST be also included in responses to any HTTP method that returns an entity body, and in PUT requests. |

A root element named notificationChannel of type NotificationChannel is allowed in request and/or response bodies.

5.2.2.3 Type: NotificationList

This type defines a list of notifications that are being delivered to the client.

| Element | Type | Optional | Description |
|---|--|----------|---|
| <Element is defined by respective Enabler server API> | <Type is defined by respective enabler API> [0..unbounded] | Yes | Contains a list (array) of notifications. The notification types are defined by the different OMA RESTful Network APIs. The list does not impose any further restriction on its content, i.e. notifications of a particular type can occur 0 or more times in the list. |
| ncListComplete | xsd:boolean | Yes | <p>Specified by the server only for the OMA Push channel type, on the Polling channel and if the client has set "largeDataPolling.pollingEnabled" to true as part of the OMA Push channel creation.</p> <p>This parameter SHALL be set to true by the server when the notificationList is complete (i.e. no more notifications are left in the "channelURL") and the "channelURL" has accordingly been destroyed by the server.</p> <p>When this parameter is true, the client SHOULD stop Polling the "channelURL" which was previously reported to it through OMA Push "LargePollingNotification". See section 5.2.2.7 for further information.</p> |

A root element named notificationList of type NotificationList is allowed in request and/or response bodies.

5.2.2.4 Type: LargePollingNotification

This type represents a wakeup call notification.

| Element | Type | Optional | Description |
|---------------|-------------------|----------|--|
| channelURL | xsd:anyURI | No | Specified by the server. Contains the URL used to retrieve the events from a dynamically created Polling Channel. |
| channelExpiry | xsd:dateTimeStamp | No | <p>Specified by the server. The time at which the channelURL will expire if the channel stays inactive (i.e. if channelURL is not polled by the client before expiry time).</p> <p>Once, the client polls the channelURL, the channelExpiry SHALL be extended appropriately by the server until the client retrieves all the events in a HTTP Polling fashion at which point the server reports the end of the notificationList to the client (see section 5.2.2.3).</p> <p>If the channelURL expires due to client's inactivity (i.e. client doesn't perform repeated polling request to retrieve the remaining events) the accumulated notifications MAY be deleted based on Notification server's policy.</p> |

A root element named `largePollingNotification` of type `LargePollingNotification` is allowed in notification request bodies.

5.2.2.5 Type: ChannelData

This is an abstract data type that contains no elements. Data type that is used to define specific information for a particular Notification Channel type (`channelData` in 5.2.2.2), SHALL be derived from this data type.

5.2.2.6 Type: LongPollingData

This type is derived from `ChannelData` and it defines specific data for the Long Polling mechanism that is used on the Notification Channel. It is used inside the ‘`channelData`’ element when a channel is created, and it is identified by `xsi:type` attribute. The `xsi:type` attribute SHALL be included in XML instances, and SHALL NOT be included in JSON instances.

| Element | Type | Optional | Description |
|-------------------------------|-------------------------|----------|---|
| <code>channelURL</code> | <code>xsd:anyURI</code> | Yes | Specified by the server. Contains the URL used to retrieve new events. The <code>channelURL</code> SHALL NOT be included in POST request to create the Notification Channel resource, but MUST be included in the response to the channel creation and any HTTP method that returns an entity body. |
| <code>maxNotifications</code> | <code>xsd:int</code> | Yes | Defines the maximum number of notifications that may be delivered in a notification list. If not specified, a default value specified by the server policy will apply, and the server SHOULD include that value in the response to the client. |
| <code>maxWaitTime</code> | <code>xsd:int</code> | Yes | Defines the maximum wait time in seconds, the client is willing to wait before it is notified of awaiting events at the notification server. If there are awaiting events, the server MUST notify the client of the awaiting events if either the <code>maxNotifications</code> or <code>maxWaitTime</code> criterion has been reached. If not specified, the server default value SHALL take effect. Setting the <code>maxWaitTime</code> to zero indicates the client’s intent to get notifications (not to exceed <code>maxNotifications</code> in a single response) as soon as there is a notification at the notification server. See section 5.3.6 for further information. |

5.2.2.7 Type: OMAPushData

This type is derived from `ChannelData` and it defines specific data for the OMAPush mechanism that is used on the Notification Channel. It is used inside the ‘`channelData`’ element when a channel is created, and it is identified by `xsi:type` attribute. The `xsi:type` attribute SHALL be included in XML instances, and SHALL NOT be included in JSON instances.

| Element | Type | Optional | Description |
|-------------------------------|-------------------------|----------|--|
| <code>appld</code> | <code>xsd:string</code> | Yes | <code>appld</code> is a required data parameter by OMA Push enabler for routing the Push Message to the appropriate application on the target device/MSISDN. |
| <code>maxNotifications</code> | <code>xsd:int</code> | Yes | Defines the maximum number of notifications that may be delivered in a notification list. Note: the actual deliverable notifications may be limited by the capabilities of the Push-OTA bearer, e.g. up to a particular total size of the notification data. |

| | | | |
|------------------|------------------|-----|--|
| | | | If not specified, a default value specified by the server policy will apply, and the server SHOULD include that value in the response to the client. |
| largeDataPolling | LargeDataPolling | Yes | Used by the client if it wishes to enable the LargeDataPolling mechanism which enables it to retrieve large notifications in a HTTP Polling fashion as opposed to receiving them over OMA Push channel. See the OMA Push description in section 5 for further information. |

5.2.2.8 Type: LargeDataPolling

This type defines parameters for LargeDataPolling.

| Element | Type | Optional | Description |
|-------------------------|-------------|----------|--|
| pollingEnabled | xsd:boolean | Yes | <p>If set to true and the number of notifications to be delivered over the OMA Push channel are more than the specified “OMAPushData.maxNotifications” or beyond the known limitation of OMA Push delivery method or certain server policy is met then the server SHALL dynamically create a “channelURL” and inform the client via the “LargePollingNotification” asynchronous event (see 5.2.2.4). The client is then able to use the “channelURL” and retrieve all the notifications in a HTTP Polling manner.</p> <p>If the element is not present or set to false, then the events are only reported via the OMA Push asynchronous delivery method (i.e. the client is not provided with the “LargePollingNotification”).</p> |
| maxPollingNotifications | xsd:int | Yes | <p>Defines the maximum number of notifications (in the notificationList) that may be delivered over the dynamically created “channelURL”.</p> <p>If not specified, a default value specified by the server policy will apply, and the server SHOULD include that value in the response to the client.</p> |

5.2.2.9 Type: LongPollingRequestParameters

This type defines parameters for Long Polling request.

| Element | Type | Optional | Description |
|---------|------|----------|---|
| (empty) | | | In the current version of this specifications, this type is empty |

A root element named longPollingRequestParameters of type LongPollingRequestParameters is allowed in request bodies.

5.2.2.10 Type: WebSocketsData

This type is derived from ChannelData and it defines specific data for a WebSockets-based Notification Channel. It is used inside the 'channelData' element when a channel is created, and it is identified by xsi:type attribute. The xsi:type attribute SHALL be included in XML instances, and SHALL NOT be included in JSON instances.

| Element | Type | Optional | Description |
|------------------|------------|----------|--|
| channelURL | xsd:anyURI | Yes | Specified by the server. Contains the URL used to open a WebSockets connection to receive event notification. The channelURL SHALL NOT be included in POST request to create the Notification Channel resource, but MUST be included in the response to the channel creation and any HTTP method that returns an entity body. |
| maxNotifications | xsd:int | Yes | Defines the maximum number of notifications that may be delivered in a notification list. If not specified, a default value specified by the server policy will apply, and the server SHOULD include that value in the response to the client. |

5.2.2.11 Type: ConnCheck

This type defines a message for WebSockets-based notification channels, see I.3.

| Element | Type | Optional | Description |
|--------------------|---------|----------|--|
| checkInterval | xsd:int | Yes | Time interval in seconds after which the sender of the ConnCheck message intends to send the next ConnCheck message. |
| newChannelLifetime | xsd:int | Yes | Offered new channel lifetime (duration) of Notification Channel in seconds. This new channel lifetime starts once the connAck message from the client corresponding to this connCheck message arrives at the server. MUST be instantiated by the server, and MUST NOT be instantiated by the client. |

A root element named connCheck of type ConnCheck is allowed in WebSockets messages.

5.2.2.12 Type: ConnAck

This type defines a message for WebSockets-based notification channels, see I.3.

| Element | Type | Optional | Description |
|-----------------|---------|----------|--|
| channelLifetime | xsd:int | Yes | Lifetime (duration) of Notification Channel in seconds. MUST be instantiated by the server, and MUST NOT be instantiated by the client. |

A root element named connAck of type ConnAck is allowed in WebSockets messages.

5.2.2.13 Type: NotificationChannelLifetime

This type defines the lifetime of a Notification Channel.

| Element | Type | Optional | Description |
|-----------------|---------|----------|--|
| channellifetime | xsd:int | Yes | <p>Remaining lifetime (duration) of Notification Channel in seconds.</p> <p>The client can specify the desired lifetime of the Notification Channel in PUT request when “refreshing” a Notification Channel, however the server in the response to the request may change the desired lifetime according to its server policy.</p> <p>If the element is not present in the request, a default channel lifetime specified by server policy will apply.</p> <p>The server SHALL always include the channel lifetime in the response.</p> |

A root element named notificationChannelLifetime of type NotificationChannelLifetime is allowed in request and/or response bodies.

5.2.3 Enumerations

The subsections of this section define the enumerations used in the Notification Channel API.

5.2.3.1 Enumeration: ChannelType

| Enumeration | Description |
|-------------|--|
| LongPolling | Indicates that the HTTP Long Polling mechanism is to be used on the Notification Channel to retrieve notifications from the Notification Server. |
| OMAPush | Indicates that the OMA Push mechanism is to be used by the Notification Server to asynchronously notify the client of events. |
| WebSockets | Indicates that a WebSockets connection is to be used by the Notification Server to asynchronously notify the client of events. |

5.2.4 Values of the Link “rel” attribute

The “rel” attribute of the Link element is a free string set by the server implementation, to indicate a relationship between the current resource and an external resource. The following are possible strings (list is non-exhaustive, and can be extended):

- NotificationChannelList
- NotificationChannel

These values indicate the kind of resource that the link points to.

5.3 Sequence Diagrams

The following subsections describe the resources, methods and steps involved in typical scenarios.

Note that signalling sequences between the Notification Server and Enabler servers X (e.g. Presence server) and Y (e.g. Messaging server), as well as the signalling sequences between the application and the Enabler servers X and Y (depicted in grey colour) are not part of this specifications; those sequences in the flows are shown for completeness only.

Upon creation of a Notification Channel, the application is required to inform the Notification Server as to the desired notification delivery mechanism. The following three notification delivery mechanisms are supported:

1. Long Polling
2. OMA Push
3. WebSockets

5.3.1 Create Notification Channel (Long Polling Method)

This figure below shows a scenario for creation of a Notification Channel by an application using the Long Polling notification delivery mechanism.

The resources:

- To create Notification Channel:
http://{serverRoot}/notificationchannel/{apiVersion}/{userId}/channels
- To retrieve new notifications:
The resource to be used is provided in the response to the channel creation.

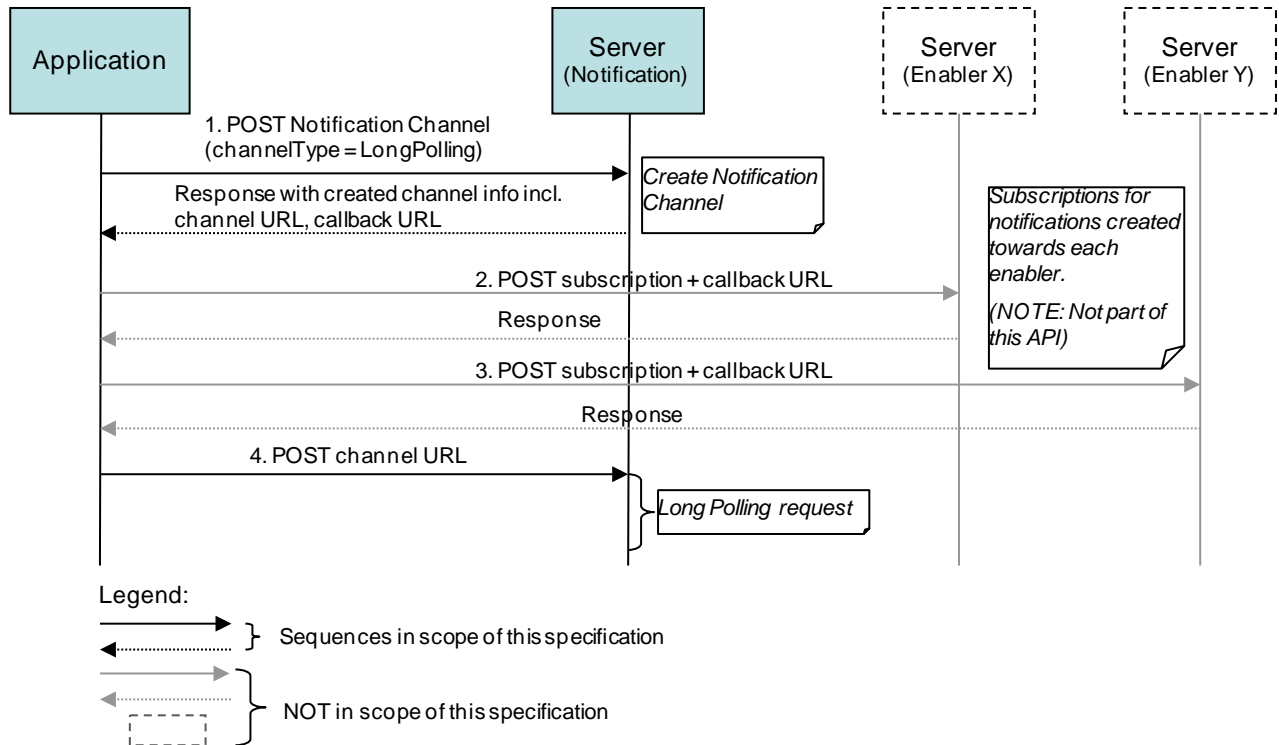


Figure 2 Create Notification Channel

Outline of the flows:

1. Application creates a Notification Channel by sending a POST request to the Notification Server indicating the desire to use the Long Polling notification delivery method by setting the channelType = LongPolling (the request may include a limit to the number of notifications that the application can receive in the responses).

A successful response includes a body containing a unique channel URL which is to be used when issuing the Long Polling request and callback URL which is to be used when subscribing for notifications to a particular Enabler server.

2. Application creates a subscription for notifications from Enabler X server. The included callback URL instructs the Enabler X server to send notifications to the Notification Server (this operation is not part of this API).

The Enabler server returns a response (this operation is not part of this API).

3. Application creates a subscription for notifications from Enabler Y server. The included callback URL instructs the Enabler server to send notifications to the Notification Server (this operation is not part of this API).

The Enabler Y server returns a response (this operation is not part of this API).

4. Application initiates a Long Polling request using the channel URL received in the response to POST in step 1 and waits for a new event.

5.3.2 Notifications delivered to application using Long Polling

This figure below shows a scenario where two notifications are delivered to the application, generated by two different servers.

The resource used by the application for the Long Polling requests is provided by the Notification Server (e.g. received in the response to creation of the Notification Channel, see section 5.3.1).

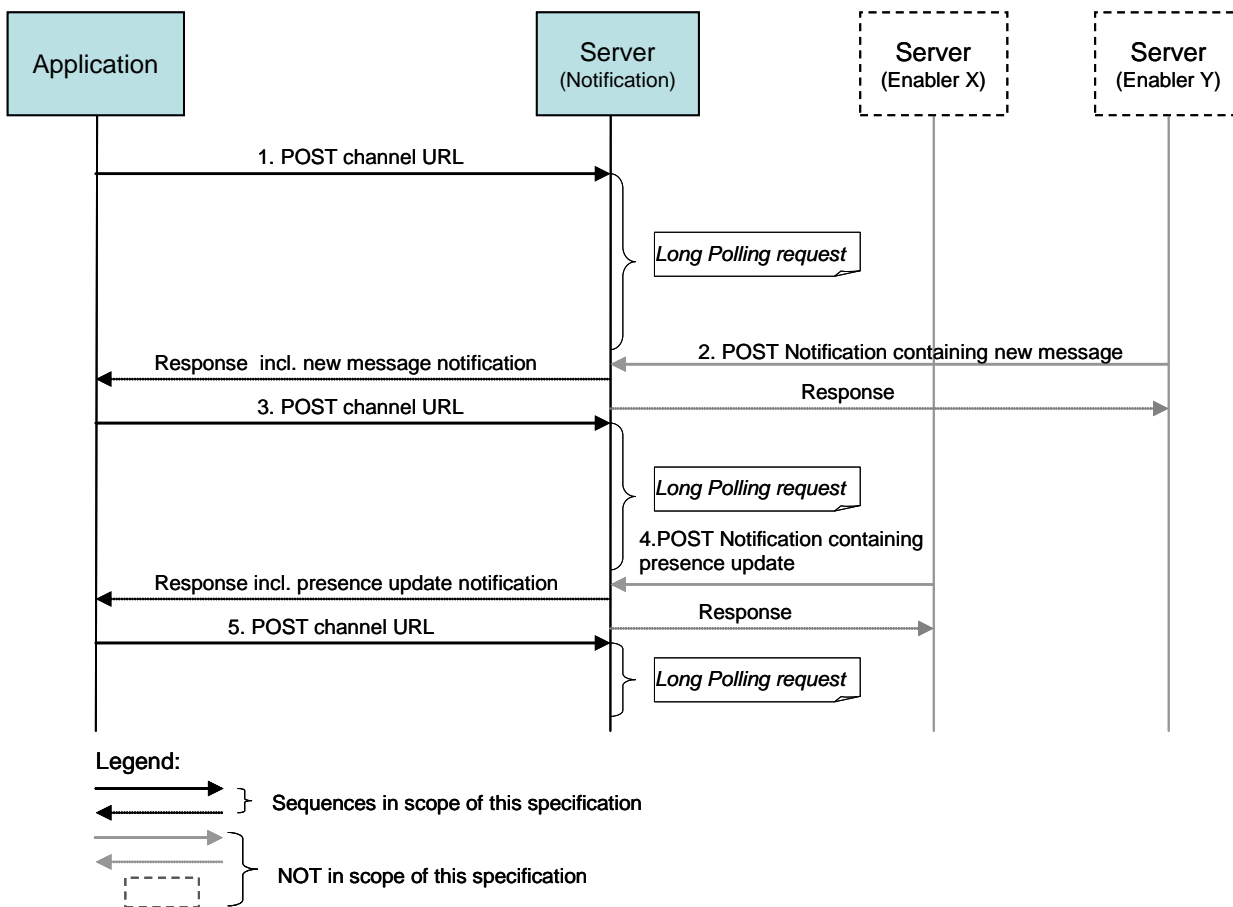


Figure 3 Notifications delivered to application

Outline of the flows:

1. Application initiates a Long Polling request using the channel URL received when the Notification Channel was created.
2. A new message is received, which triggers a notification being sent from the Enabler Y server to the Notification Server using the callback URL provided when the Notification Channel was created (this operation is not part of this API).

A response to the Long Polling request in step 1 is delivered to the application including the new message.

A response to the notification received in step 2 is sent to Enabler Y server after the response is delivered to the application (this operation is not part of this API).

3. Application immediately initiates a new Long Polling request.

4. A new event occurs; in this case a presence update notification is received in the Notification Server using the callback URL provided when the Notification Channel was created (this operation is not part of this API).

A response to the Long Polling request in step 3 is delivered to the application including the presence update.

A response to the notification received in step 4 is sent to Enabler X server after the response is delivered to the application (this operation is not part of this API).

5. Application immediately initiates a new Long Polling request and waits for a new event.

5.3.3 Long Polling request timeout response

This figure below shows a scenario where a Long Polling request times out and a new Long Polling request is sent.

Note that the timeout mentioned below is a value specific to the Long Polling implementation, and not the “channelLifetime” as defined in section 5.2.2.2.

The resource used by the application for the Long Polling requests is provided by the Notification Server (e.g. received in the response to creation of the Notification Channel, see section 5.3.1).

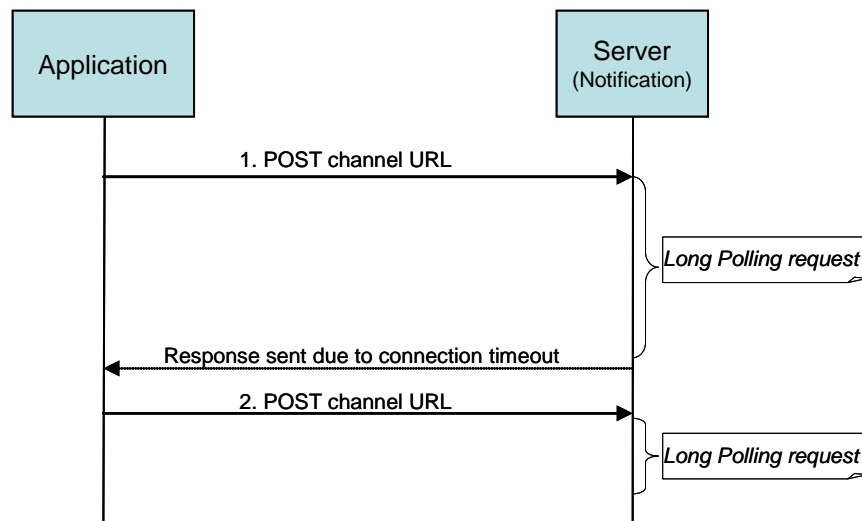


Figure 4 Request timeout

Outline of the flows:

1. Application initiates a Long Polling request using the channel URL received when the Notification Channel was created.
No new event is received within a given time limit causing the request to timeout. An empty response is returned to the application.
2. Application immediately initiates a new Long Polling request and waits for a new event.

5.3.4 Multiple notifications delivered to application in response to the Long Polling request

This figure below shows a scenario where two notifications are delivered to the application in the same response.

The resource used by the application for the Long Polling requests is provided by the Notification Server (e.g. received in the response to creation of the Notification Channel, see section 5.3.1).

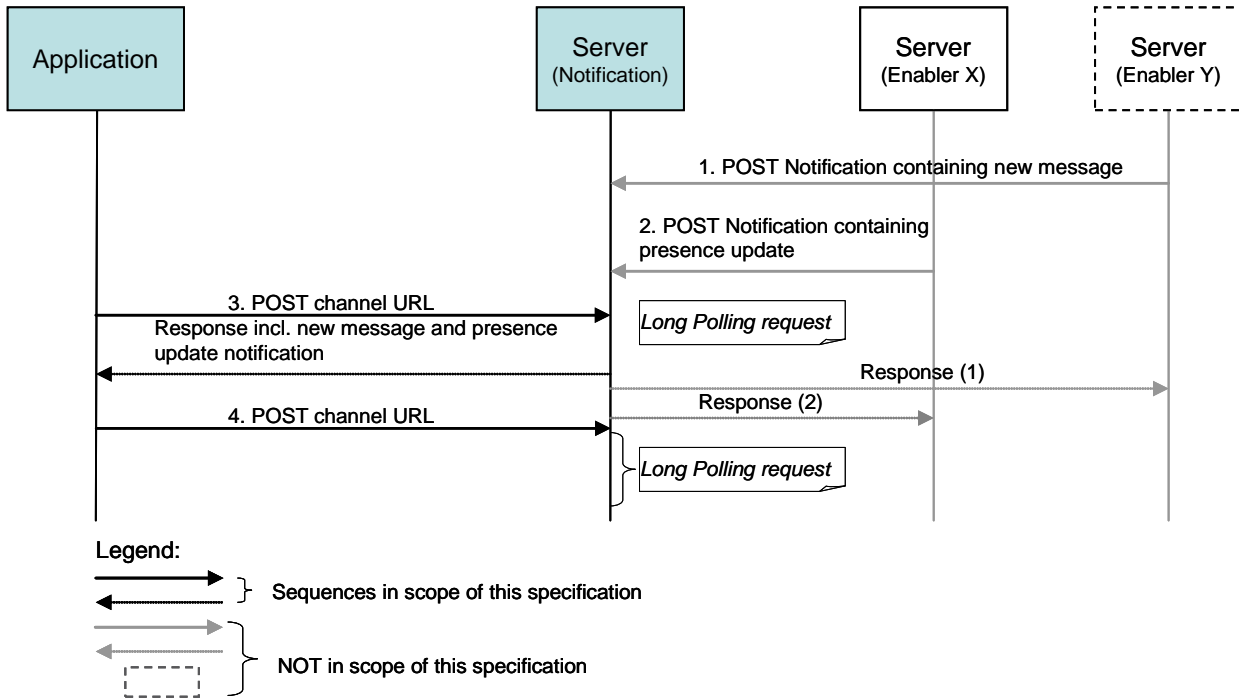


Figure 5 Multiple notifications delivered to application in response

Outline of the flows:

1. A new message is received but in this case there is no outstanding Long Polling request from the application so the notification will be pending in the Notification Server (this operation is not part of this API).
2. A new event occurs; in this case a presence update notification is received. As there is no outstanding Long Polling request from the application the notification will be pending in the Notification Server (this operation is not part of this API).
3. Application initiates a Long Polling request using the channel URL received when the Notification Channel was created. A response to the Long Polling request in step 3 is delivered to the application including the new message and the presence update notification (assuming that the application allowed multiple notifications in the response when the Notification Channel was created).
A response to the notification received in step 1 is sent to Enabler Y server after the response is delivered to the application (this operation is not part of this API).
A response to the notification received in step 2 is sent to Enabler X server after the response is delivered to the application (this operation is not part of this API).
4. Application immediately initiates a new Long Polling request and waits for a new event.

5.3.5 Max number of notifications reached during the Long Polling

This figure below shows a scenario where the limit for the number of notifications in the response to the application (in this example, 3 notifications) has been reached, which triggered response back to the application.

The resource used by the application for the Long Polling requests is provided by the Notification Server (e.g. received in the response to creation of the Notification Channel, see section 5.3.1).

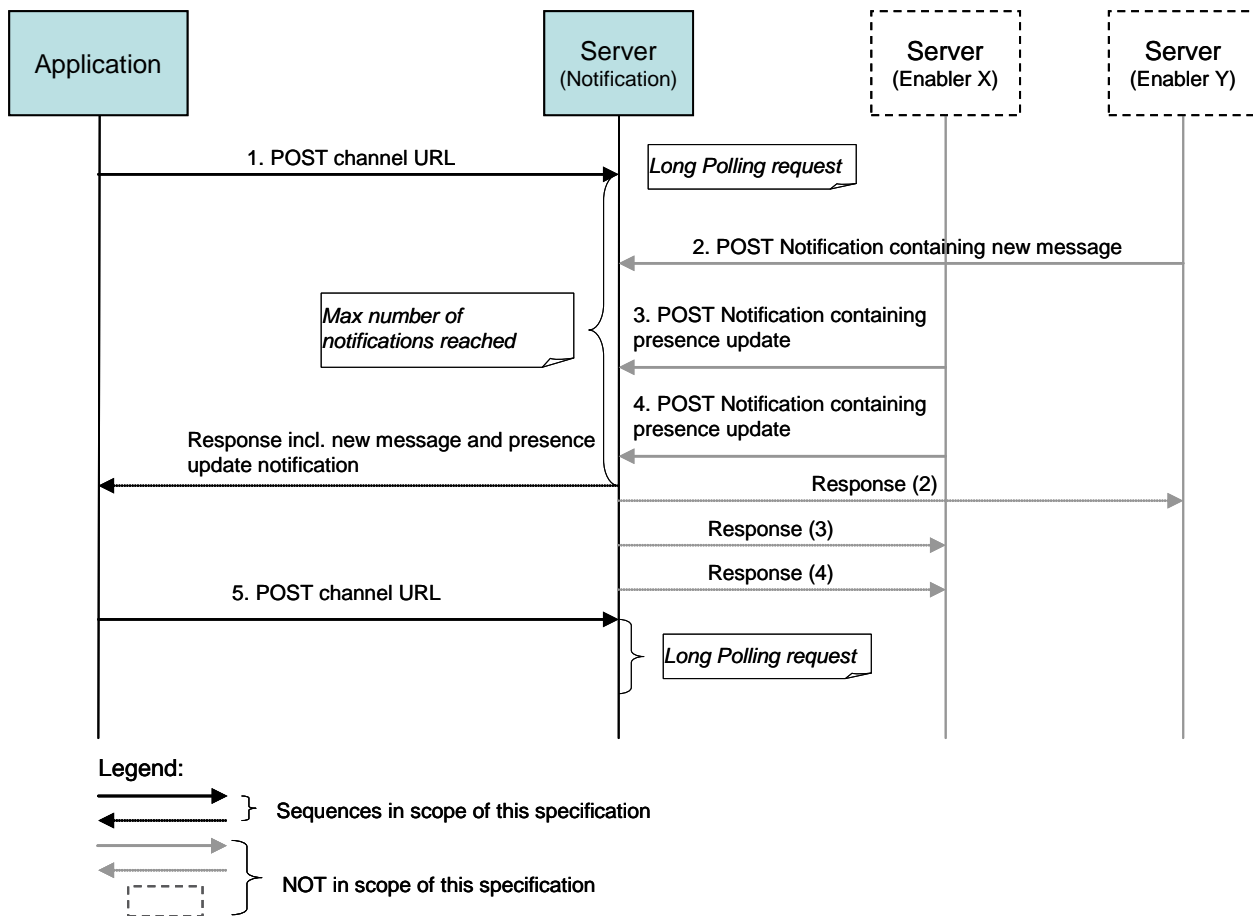


Figure 6 Maximum number of notifications in the response to the Long Polling

Outline of the flows:

1. Application initiates a Long Polling request using the channel URL received when the Notification Channel was created.
2. A new message has been received and the Notification Server is notified (this operation is not part of this API). Since the maxNotifications limit is not yet reached no response to the Long Polling request is sent back to the application.
3. A new event occurs; in this case a presence update notification is received at the Notification Server (this operation is not part of this API). The maxNotifications limit is still not reached.
4. A new event occurs; in this case another presence update notification is received at the Notification Server (this operation is not part of this API).

The maximum number of notifications allowed in the response has been reached and the response to the Long Polling request in step 1 is sent to the application. The response includes the new message and the two presence update notifications.

A response to the notification received in step 2 is sent to Enabler Y server after the response is delivered to the application (this operation is not part of this API).

A response to the notification received in step 3 is sent to Enabler X server after the response is delivered to the application (this operation is not part of this API).

A response to the notification received in step 4 is sent to Enabler X server after the response is delivered to the application (this operation is not part of this API).

5. Application immediately initiates a new Long Polling request.

5.3.6 Max wait time or max number of notifications reached during the Long Polling

This figure below shows a scenario where the limit for the maximum wait time or maximum number of notifications in the response to the application has been reached, which triggers a response back to the application. In the example below the following assumptions have been used:

- Client application at Long Polling Notification Channel creation has set `maxWaitTime = 5 second` and `maxNotifications = 3 events`.
- Server's Long Polling connection timeout = 45 seconds (i.e. Long Polling request to the server times out in 45 seconds if there is no event received within 45 seconds. The client needs to immediately send a new Long Polling request upon receiving an empty response to a prior Long Polling request).

The resource used by the application for the Long Polling requests is provided by the Notification Server (e.g. received in the response to creation of the Notification Channel, see section 5.3.1).

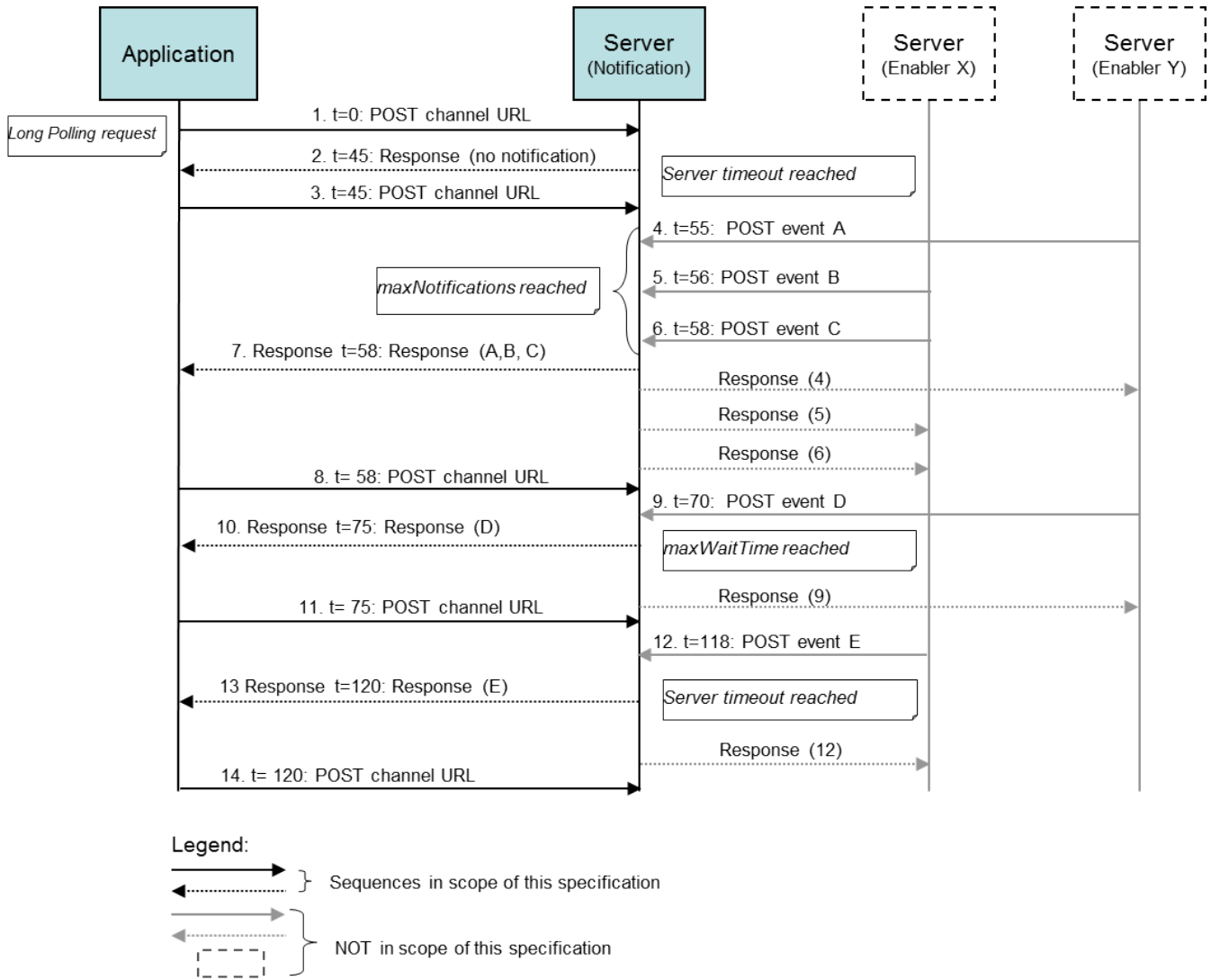


Figure 7 Max wait time or max number of notifications criterion used to respond to the Long Polling

Outline of the flows (where t is time in seconds):

1. Application initiates a Long Polling request at t= 0 using the channel URL received when the Notification Channel was created.
2. No new event is received within a given time limit (45 seconds in this example) causing the request to timeout. An empty response is returned to the application at t=45.
3. Application immediately initiates a new Long Polling request at t= 45
4. At t=55 event “A” arrives at the Notification Server. Since the maxWaitTime (from the time this event has arrived) nor maxNotifications has been reached, the server holds onto the event

5. At t=56 event “B” arrives at the Notification Server. Since the maxWaitTime nor maxNotifications has been reached, the server holds onto the event
6. At t=58 event “C” arrives at the Notification Server. The maximum number of notifications allowed (maxNotifications =3 events) in the response has been reached
7. The response to the Long Polling request in step 3 is sent to the application at t=58. The response includes events “A”, “B” and “C”.
8. Application immediately initiates a new Long Polling request at t= 58
9. At t=70 event “D” arrives at the Notification Server. Since the maxWaitTime (from the time this event has arrived) nor maxNotifications has been reached, the server holds onto the event
Time passes by and at t=75, maxWaitTime = 5 seconds (from the time the first event arrived) limit is reached.
10. The response to the Long Polling request in step 8 is sent to the application at t=75. The response includes events “D”.
11. Application immediately initiates a new Long Polling request at t= 75
12. At t=118 event “E” arrives at the Notification Server. Since the maxWaitTime nor maxNotifications has been reached, the server holds onto the event
Time passes by and at t=120, server’s connection timeout kicks in
13. The response to the Long Polling request in step 11 is sent to the application at t=120. The response includes events “E”.
14. Application immediately initiates a new Long Polling request at t= 120

5.3.7 Create Notification Channel (OMA Push Method)

This figure below shows a scenario for creation of a Notification Channel by an application using the OMA Push notification delivery mechanism.

The resources:

- To create Notification Channel:
http://{serverRoot}/notificationchannel/{apiVersion}/{userId}/channels

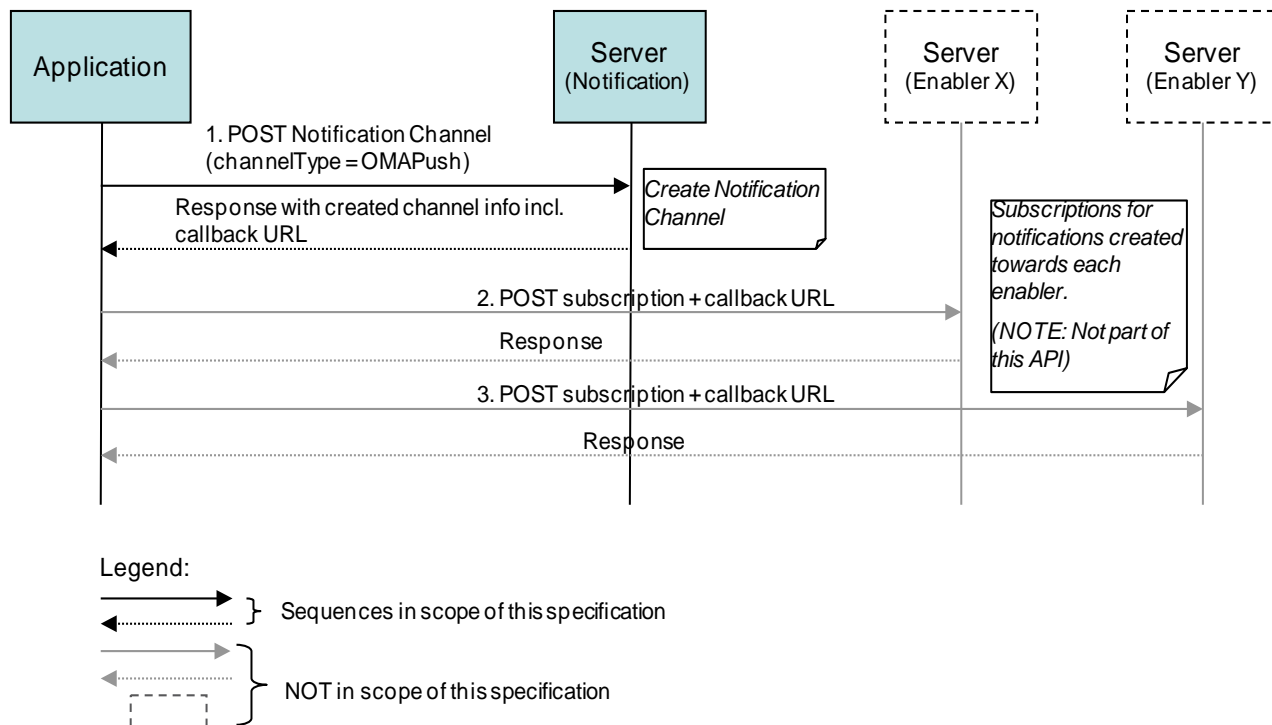


Figure 8 Create Notification Channel (OMA Push Method)

Outline of the flows:

1. Application creates a Notification Channel by sending a POST request to the Notification Server indicating the desire to use the OMA Push notification delivery method by setting the channelType = OMAPush. The request may include a limit to the number of notifications that the application can receive in the asynchronous notification list. Additionally, the request may contain an appId which uniquely identify the application to the OMA Push Enabler.

A successful response includes a body containing a callback URL which is to be used when subscribing for notifications to a particular Enabler server.
2. Application creates a subscription for notifications from Enabler X server. The included callback URL instructs the Enabler X server to send notifications to the Notification Server (this operation is not part of this API).

The Enabler server returns a response (this operation is not part of this API).
3. Application creates a subscription for notifications from Enabler Y server. The included callback URL instructs the Enabler server to send notifications to the Notification Server (this operation is not part of this API).

The Enabler Y server returns a response (this operation is not part of this API).

5.3.8 Notifications delivered to application using OMA Push

This figure below shows a scenario where two notifications generated by two different servers are delivered to the application

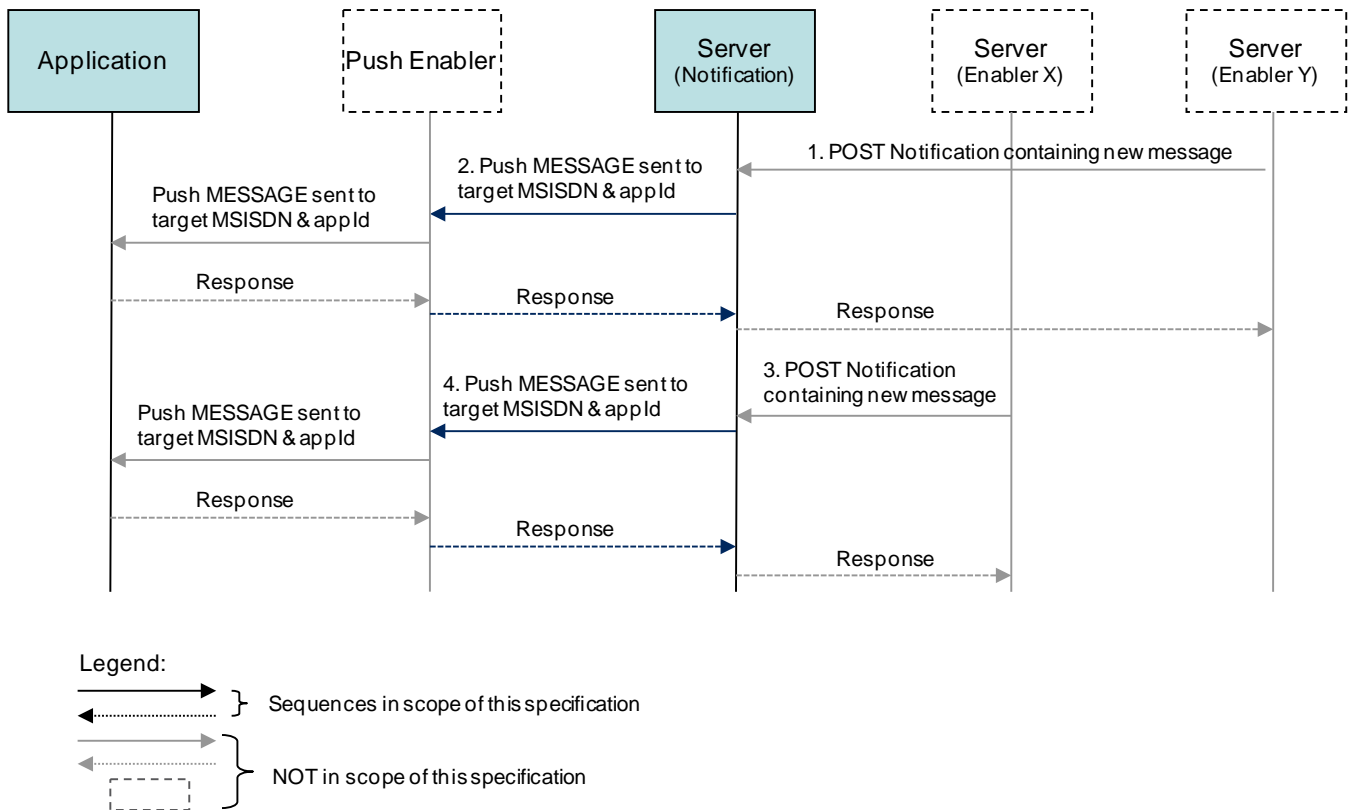


Figure 9 Notifications delivered to application using OMA Push

Outline of the flows:

1. An event occurs which triggers a notification being sent from Enabler Y server to the Notification Server using the callback URL provided when the Notification Channel was created (this operation is not part of this API).
2. The Notification Server maps the callback URL at which it received the event to the associated MSISDN and appId which it had previously captured as part of the channel creation process. A Push MESSAGE containing the new event is then sent from the Notification Server to the Push Enabler targeting the appropriate MSISDN and appId (this operation is not part of this API. See Appendix H for further information regarding Notification Server and Push Enabler interaction).

Note: In advance configuration of the Notification Server with the appropriate Push Enabler (e.g. PPG) address is outside the scope of this document.

In turn, Push Enabler passes the Push MESSAGE containing the new event to the application on the device via the Push client residing on the device (this operation is not part of this API).

If requested by the Notification Server, the Push client or application may provide a delivery confirmation, which is forwarded to the Notification Server by the Push Enabler (this operation is not part of this API).

A response to the notification received in step 1 is sent to Enabler Y server after the response is delivered to the application (this operation is not part of this API).

3. The same process as explain in step 1 above involving Enabler X.
4. The same process as explain in step 2 above involving Enabler X.

5.3.9 Create Notification Channel (OMA Push method with Large Data Polling enabled)

This figure (below) shows a scenario for creation of a Notification Channel by an application using the OMA Push notification delivery mechanism with “LargeDataPolling” feature enabled.

The resources:

- To create Notification Channel:

http://{serverRoot}/notificationchannel/{apiVersion}/{userId}/channels

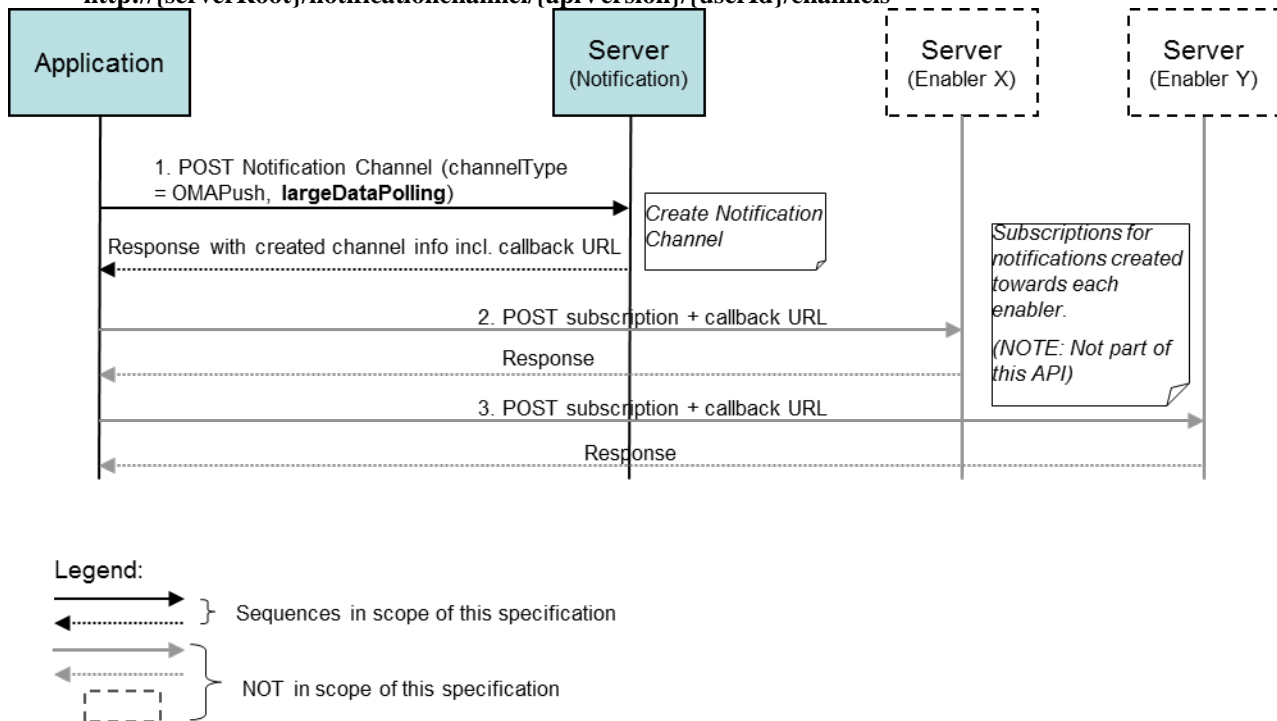


Figure 10 Create Notification Channel (OMA Push method with Large Data Polling enabled)

Outline of the flows:

1. Application creates a Notification Channel by sending a POST request to the Notification Server indicating the desire to use the OMA Push notification delivery method with the “largeDataPolling” feature enabled (i.e. channelType = OMAPush and “largeDataPolling.pollingEnabled” = true). The request includes maxNotifications indicating the number of notifications that the application can receive over the OMA Push delivery method and if the number of messages exceed the maxNotifications limit (or notification size is beyond the known limitation), then the desire to receive the events over a polling channel (i.e. LargeDataPolling channelURL). Additionally, the request may contain an appId which uniquely identify the application to the OMA Push Enabler.

A successful response includes a body containing a callback URL which is to be used when subscribing for notifications to a particular Enabler server.

2. Application creates a subscription for notifications from Enabler X server. The included callback URL instructs the Enabler X server to send notifications to the Notification Server (this operation is not part of this API).

The Enabler server returns a response (this operation is not part of this API).

3. Application creates a subscription for notifications from Enabler Y server. The included callback URL instructs the Enabler server to send notifications to the Notification Server (this operation is not part of this API).

The Enabler Y server returns a response (this operation is not part of this API).

5.3.10 Notifications delivered to application using OMA Push while Large Data Polling is enabled

This figure (below) shows a scenario where notifications are delivered to the application over a dynamically generated Large Data Polling channel URL when certain conditions are met (e.g. number of notifications are more than maxNotifications).

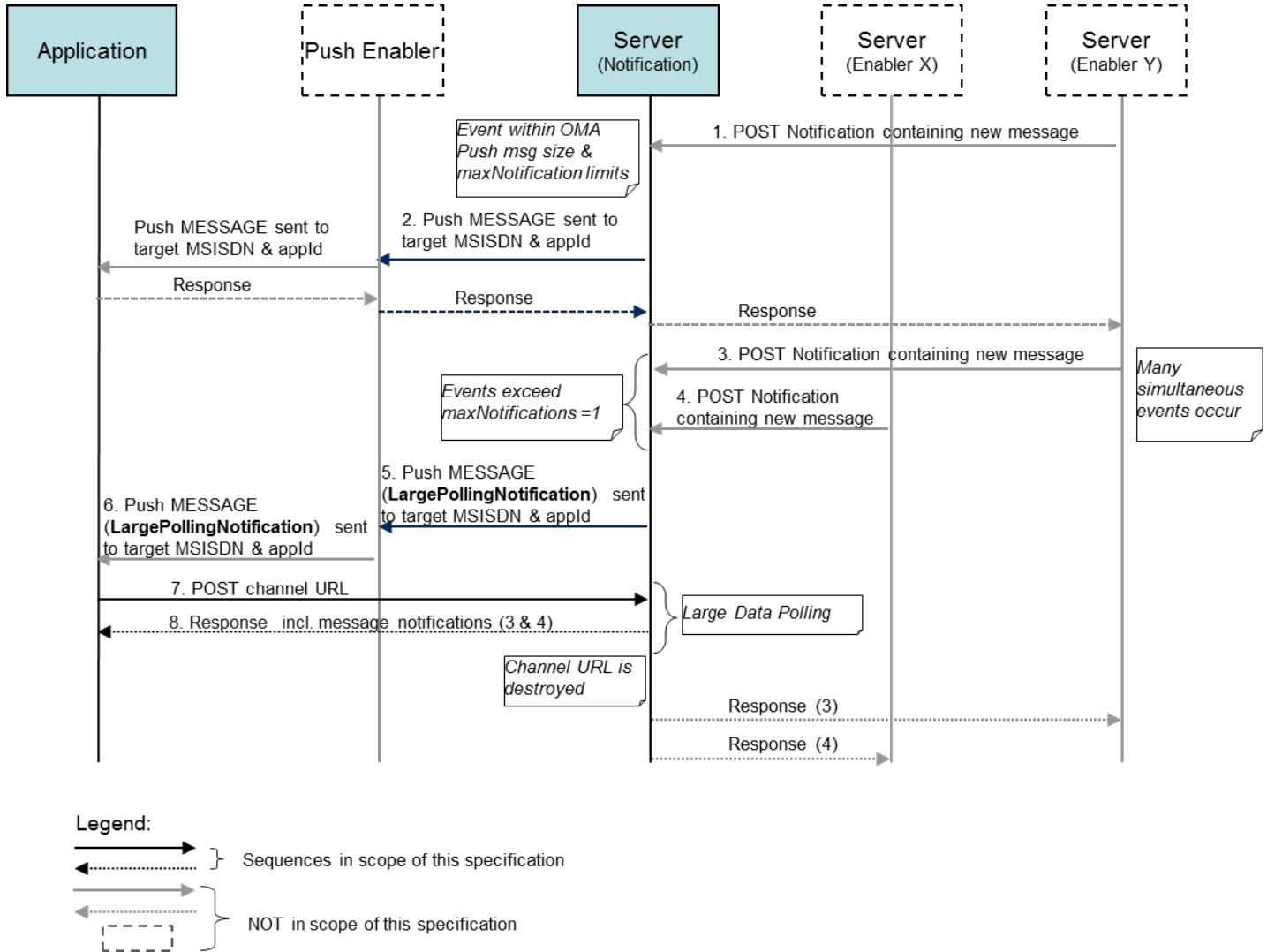


Figure 11 Notifications delivered to application using OMA Push

Outline of the flows:

1. An event occurs which triggers a notification being sent from Enabler Y server to the Notification Server using the callback URL provided when the Notification Channel was created (this operation is not part of this API).
2. The Notification Server maps the callback URL at which it received the event to the associated MSISDN and appId which it had previously captured as part of the channel creation process. Since, the notification size and the number of notifications does not meet the conditions requiring a dynamic Channel URL generation (i.e. the event can be delivered through OMA Push delivery method), a Push MESSAGE containing the new event (from Enabler Y) is then sent from the Notification Server to the Push Enabler targeting the appropriate MSISDN and appId (this operation is not part of this API. See Appendix H for further information regarding Notification Server and Push Enabler interaction).

Note: In advance configuration of the Notification Server with the appropriate Push Enabler (e.g. PPG) address is outside the scope of this document.

In turn, Push Enabler passes the Push MESSAGE containing the new event to the application on the device via the Push client residing on the device (this operation is not part of this API).

If requested by the Notification Server, the Push client or application may provide a delivery confirmation, which is forwarded to the Notification Server by the Push Enabler (this operation is not part of this API).

A response to the notification received in step 1 is sent to Enabler Y server after the response is delivered to the application (this operation is not part of this API).

3. After some time, an event occurs which triggers a notification being sent from Enabler Y server to the Notification Server (as explain in step 1 above).
4. Concurrent to step 3, another event occurs in Enabler X which triggers a notification being sent to the Notification Server (as explain in step 1 above).

The Notification Server realizes that, as per client's request, it needs to inform the client that it should poll the awaiting notifications (i.e. number of notifications arrived at the Notification Server is more than maxNotification =1).

5. The Notification Server, sends a Push MESSAGE (to the appropriate MSISDN and appId) containing a "LargePollingNotification" which itself contains a dynamically created "channelURL" (this operation is not part of this API. See Appendix H for further information regarding Notification Server and Push Enabler interaction).
6. In turn, Push Enabler passes the Push MESSAGE containing the "LargePollingNotification" to the application on the device via the Push client residing on the device (this operation is not part of this API).
7. The client invokes a POST onto the "channelURL" which it extracted from the "LargePollingNotification".
8. The Notification server responds with a "NotificationList" containing events which it received from Enabler Y and X (as shown step 3 & 4). The Server indicates the end of NotificationList to the client by setting "ncListComplete" = true. The Notification server also destroys the dynamically created channel URL (which was reported through LargePollingNotification in step 5).

A response to the notifications received in step 3 and 4 are sent to Enabler Y and X server after the notifications are delivered to the application (this operation is not part of this API).

5.3.11 Create Notification Channel (WebSockets)

This figure below shows a scenario for creation of a Notification Channel by an application using WebSockets as the delivery mechanism.

The resources:

- To create a Notification Channel:
http://{serverRoot}/notificationchannel/{apiVersion}/{userId}/channels

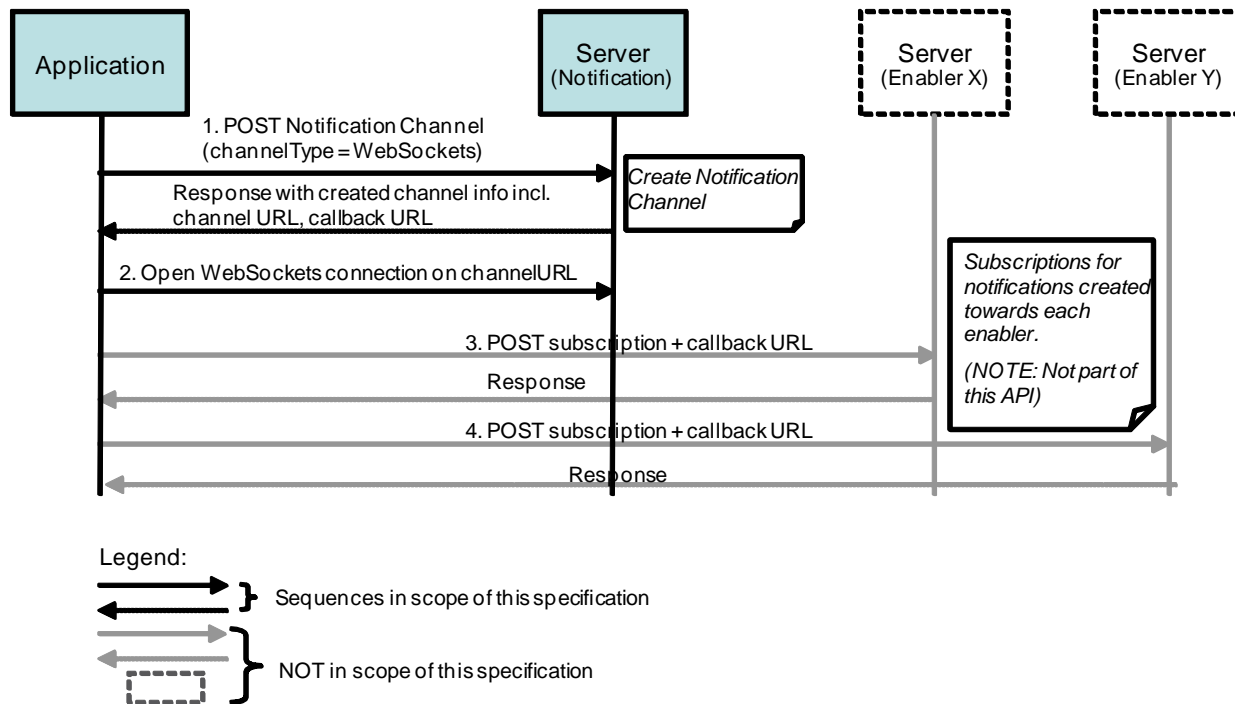


Figure 12 Create Notification Channel (WebSockets)

Outline of the flows:

1. The application creates a Notification Channel by sending a POST request to the Notification Server indicating the desire to use the WebSockets notification delivery method by setting channelType = WebSockets (the request may include a limit to the number of notifications that the application can receive in one batch).
 A successful response includes a body containing a unique channel URL which is to be used to open the WebSockets connection, and a callback URL which is to be used when subscribing for notifications to a particular Enabler server.
2. The application opens a WebSockets connection on the channel URL received in the response to POST in step 1 and waits for notifications arriving via this connection.
3. The application creates a subscription for notifications from Enabler X server. The included callback URL instructs the Enabler X server to send notifications to the Notification Server (this operation is not part of this API).
 The Enabler server returns a response (this operation is not part of this API).
4. The application creates a subscription for notifications from Enabler Y server. The included callback URL instructs the Enabler server to send notifications to the Notification Server (this operation is not part of this API).
 The Enabler Y server returns a response (this operation is not part of this API).

5.3.12 Notifications delivered to application using WebSockets

This figure below shows a scenario where two notifications generated by two different servers are delivered to the application using WebSockets,

The resources:

- There is no HTTP resource involved in delivering the notifications, as these are received via the WebSockets connection.

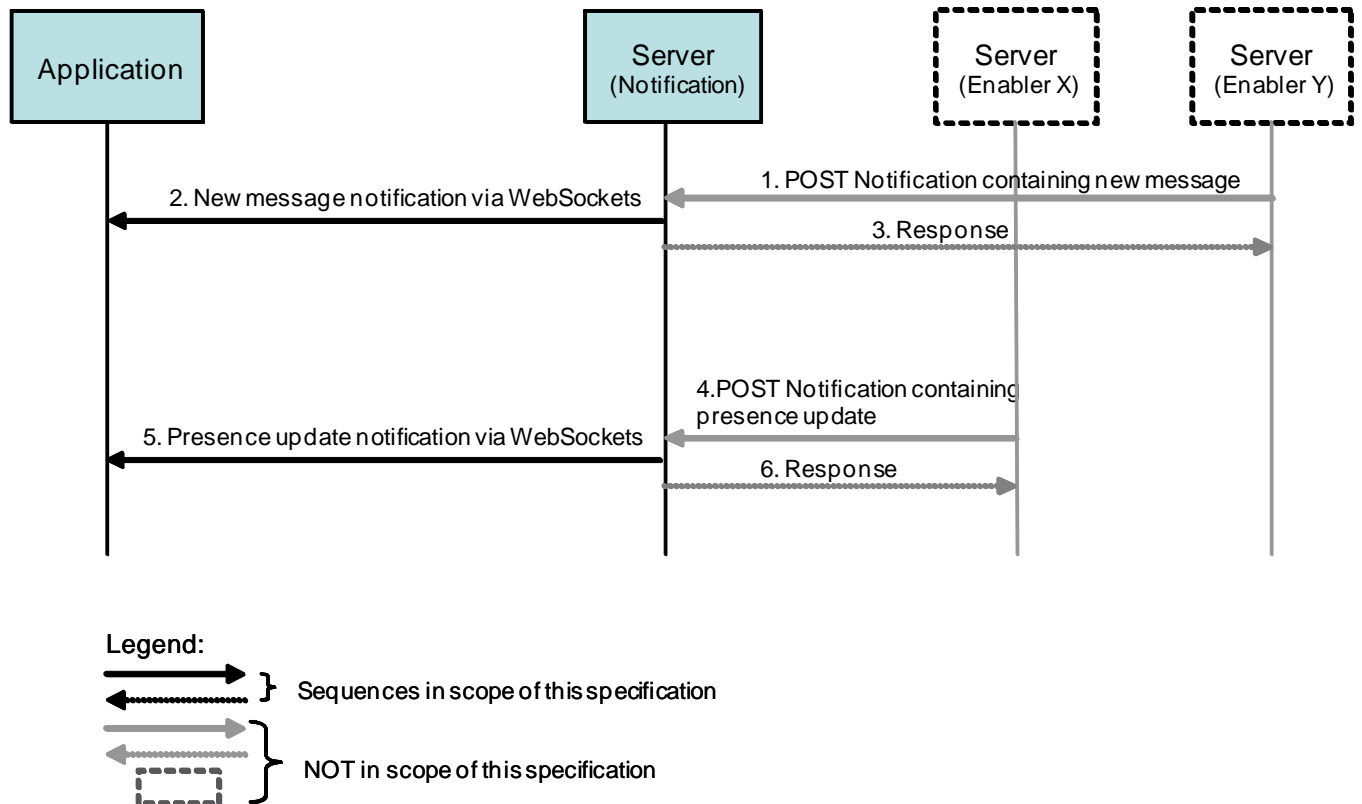


Figure 13 Notifications delivered to application using WebSockets

Outline of the flows:

1. A new message is received by the Enabler Y server, which triggers a notification being sent from the Enabler Y server to the Notification Server (this operation is not part of this API).
2. A notification including the new message is delivered to the application via the WebSockets connection.
3. A response to the notification received in step 1 is sent to Enabler Y server after the response has been delivered to the application (this operation is not part of this API).
4. A new event occurs; in this case a presence update notification is received in the Notification Server from Enabler X server (this operation is not part of this API).
5. A notification regarding the presence update is delivered to the application via the WebSockets connection.
6. A response to the notification received in step 4 is sent to Enabler X server after the response has been delivered to the application (this operation is not part of this API).

5.3.13 Refreshing a Notification Channel

This figure below shows how the application can refresh a Notification Channel.

The resources:

- To refresh a Notification Channel:
`http://{serverRoot}/notificationchannel/{apiVersion}/{userId}/channels/{channelId}/channelLifetime`

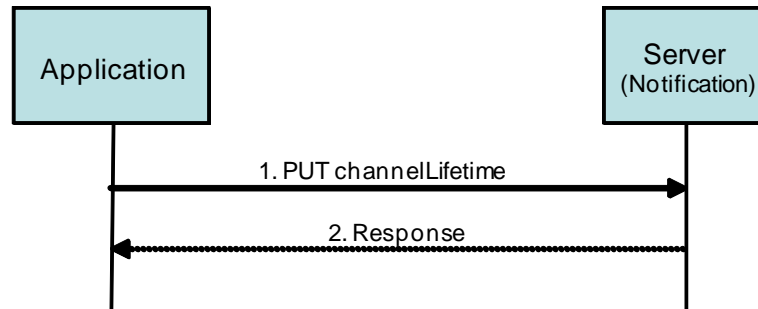


Figure 14 Notification Channel refresh

Outline of the flows:

1. To refresh a notification channel, the application uses the PUT method to set a new value of the resource representing the channel lifetime.
2. The server returns a response confirming the update, possibly returning a different channel lifetime value than the one in the request if that one was modified due to a policy.

5.3.14 Client-initiated ConnCheck/ConnAck

This figure below shows how the application can use ConnCheck/ConnAck for session keep-alive.

This flow is only relevant for Websockets-based notification channels.

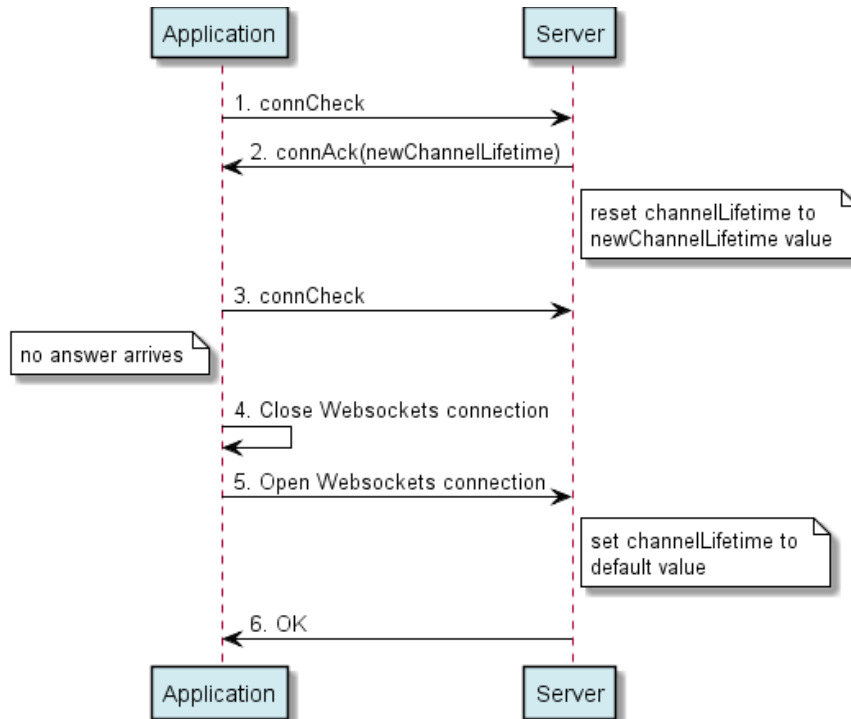


Figure 15 Client-initiated ConnCheck/ConnAck for session keep-alive

Outline of the flows:

1. The application sends a connCheck message to the server
2. The server responds with a connAck message, indicating the lifetime of the channel after sending the connAck. The server resets the channel lifetime to the value indicated to the application.
3. After a certain time interval, the application sends another connCheck message to the server. No answer arrives during a time interval in which an answer is expected (note that the server is usually responding to a connCheck immediately).
4. The application considers the Websockets connection defunct and closes it.
5. The application opens a new Websockets connection with the server. The server resets the channel lifetime to a default value depending on operator policies.
6. The server confirms the successful creation of the Websockets connection.

5.3.15 Successful server-initiated ConnCheck/ConnAck

This figure below shows how the server can use ConnCheck/ConnAck for session keep-alive.

This flow is only relevant for Websockets-based notification channels.

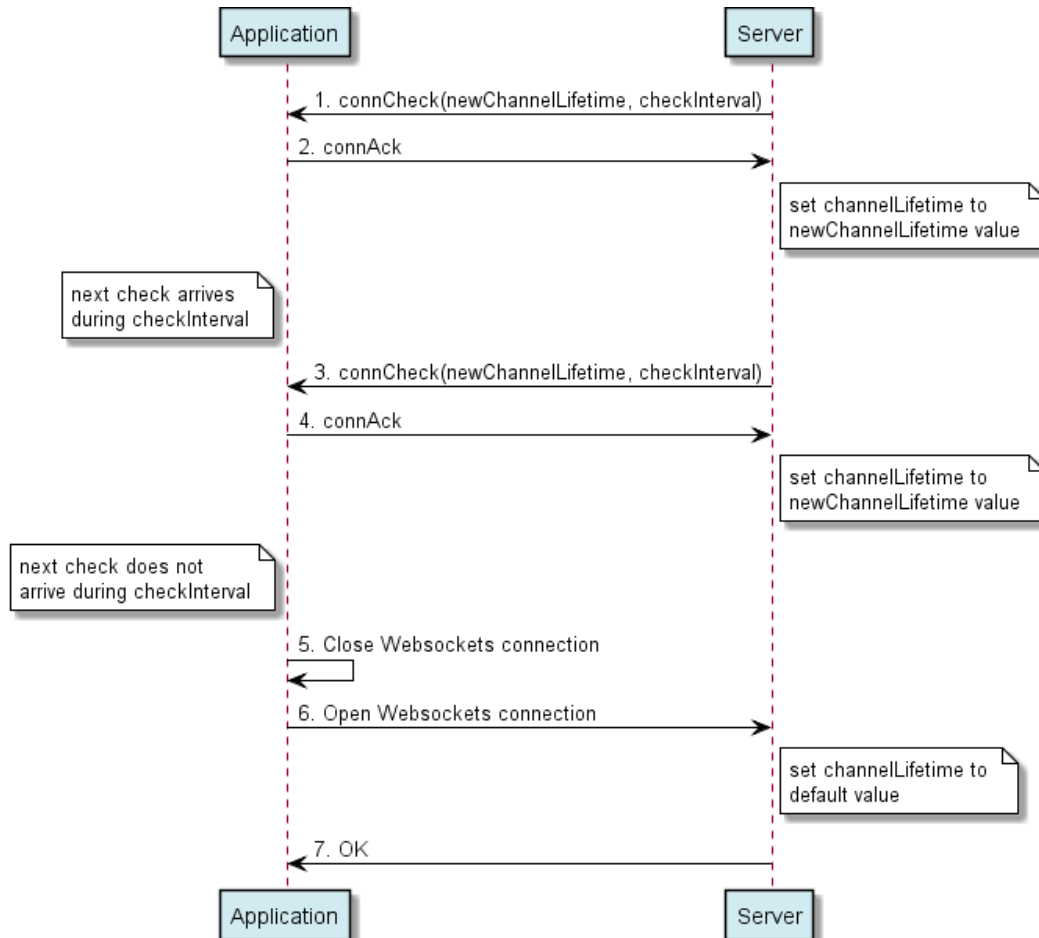


Figure 16 Successful server-initiated ConnCheck/ConnAck for session keep-alive

Outline of the flows:

1. The server sends to the application a `connCheck` message, indicating the lifetime of the channel it intends to use after it will receive the `connAck`, and further indicating a time interval after which it intends to send the next check.
2. The application responds with a `connAck` message. Upon receiving this message, the server resets the channel lifetime to the value indicated in the `connCheck` message.
3. Before the expiry of the time interval announced in the “`checkInterval`” parameter in the previous `connCheck` message, the server sends another `connCheck` message to the application.
4. The application responds with a `connAck` message. Upon receiving this message, the server resets the channel lifetime to the value indicated in the `connCheck` message.
5. The application expects to receive another `connCheck` message within the interval announced in the “`checkInterval`” parameter of the previous `connCheck` message, however such message does not arrive. The application therefore considers the Websockets connection defunct and closes it.
6. No answer arrives during a time interval in which an answer is expected (note that the server is usually responding to a `connCheck` immediately).

7. The application considers the Websockets connection defunct and closes it.
8. The application opens a new Websockets connection with the server. The server resets the channel lifetime to a default value depending on operator policies.
9. The server confirms the successful creation of the Websockets connection.

5.3.16 Unsuccessful server-initiated ConnCheck/ConnAck

This figure below shows a flow of an unsuccessful server-initiated session keep-alive, due to the client failing to respond. This flow is only relevant for Websockets-based notification channels.

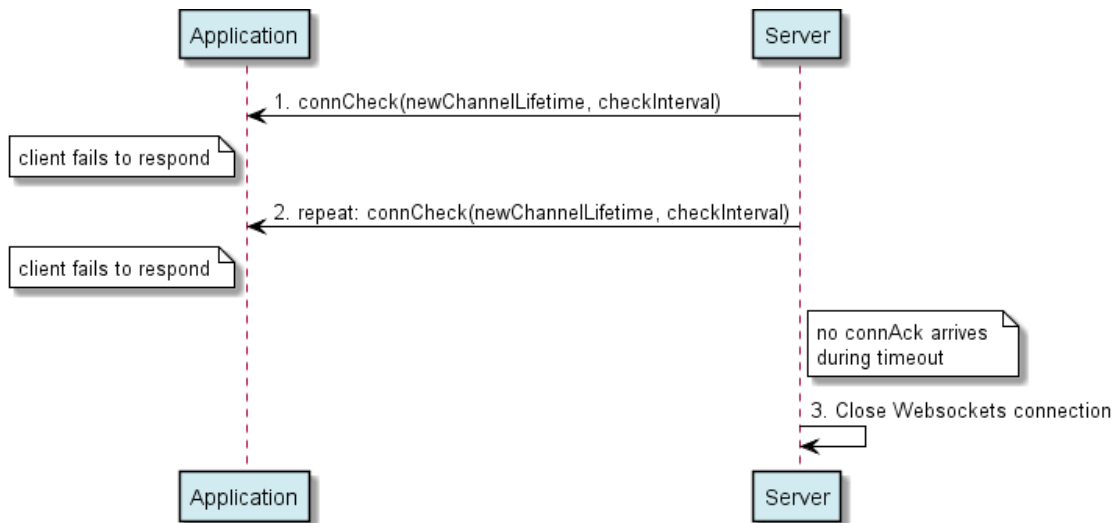


Figure 17 Unsuccessful server-initiated ConnCheck/ConnAck for session keep-alive

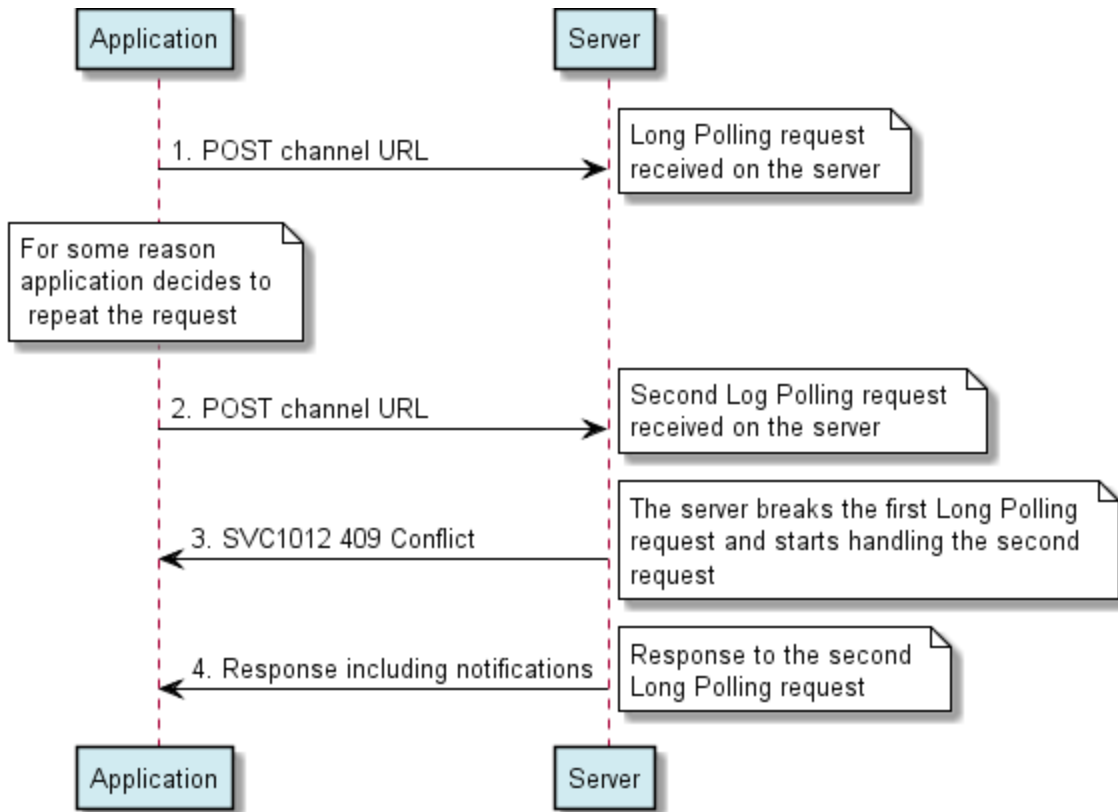
Outline of the flows:

1. The server sends to the application a connCheck message, indicating the lifetime of the channel it intends to use after it will receive the connAck, and further indicating a time interval after which it intends to send the next check.
2. The application fails to respond. The server might send additional connCheck messages until an implementation-specific timeout occurs.
3. After that timeout has occurred, the server closes the Websockets connection, since the application has failed to respond to any of the previous connCheck messages.

5.3.17 Notifications delivered to application using Long Polling

This figure below shows a scenario where the Long Polling request suffers a connection failure causing the client to reconnect. On receiving the request the server rejects the initial (failed) request and continues with the second.

The resource used by the application for the Long Polling requests is provided by the Notification Server (e.g. received in the response to creation of the Notification Channel, see section 5.3.1).



Outline of the flows:

1. Application initiates a Long Polling request using the channel URL received when the Notification Channel was created.
2. For some reason the application decides to re-initiate a Long Polling request using the same channel URL as (1).
3. Notification Server detects multiple Long Polling requests for the same channel URL. It rejects the first (i.e., the one initiated in (1)), but establishes the second.
4. Notification Server responds to the second Long Polling request (i.e., the one initiated in (2)) with notifications received.

6. Detailed specification of the resources

The following applies to all resources defined in this specification regardless of the representation format (i.e. XML, JSON, application/x-www-form-urlencoded):

- Reserved characters in URL variables (parts of a URL denoted below by a name in curly brackets) **MUST** be percent-encoded according to [RFC3986]. Note that this always applies, no matter whether the URL is used as a Request URL or inside the representation of a resource (such as in “resourceURL” and “link” elements).
- If a user identifier (e.g. address, userId, etc) of type anyURI is in the form of an MSISDN, it **MUST** be defined as a global number according to [RFC3966] (e.g. tel:+19585550100). The use of characters other than digits and the leading “+” sign **SHOULD** be avoided in order to ensure uniqueness of the resource URL. This applies regardless of whether the user identifier appears in a URL variable or in a parameter in the body of an HTTP message.
- If a user identifier (e.g. address, userId, etc) of type anyURI is in the form of a SIP URI, it **MUST** be defined according to [RFC3261].
- If a user identifier (e.g. address, userId, etc) of type anyURI is in the form of an Anonymous Customer Reference (ACR), it **MUST** be defined according to Appendix H of [REST_NetAPI_ACR].
 - The ACR ‘auth’ is a supported reserved keyword, and **MUST NOT** be assigned as an ACR to any particular end user. See G.1.2 for details regarding the use of this reserved keyword.
- For requests and responses that have a body, the following applies: in the requests received, the server **SHALL** support JSON and XML encoding of the parameters in the body, and **MAY** support application/x-www-form-urlencoded parameters in the body. The Server **SHALL** return either JSON or XML encoded parameters in the response body, according to the result of the content type negotiation as specified in [REST_NetAPI_Common]. In notifications to the Client, the server **SHALL** use either XML or JSON encoding, depending on which format the client has specified in the related subscription. The generation and handling of the JSON representations **SHALL** follow the rules for JSON encoding in HTTP Requests/Responses as specified in [REST_NetAPI_Common].

6.1 Resource: Notification channels

The resource used is:

http://{serverRoot}/notificationchannel/{apiVersion}/{userId}/channels

This resource is used for create a new Notification Channel as well as to obtain a list of active Notification Channels for the specified user.

6.1.1 Request URL variables

The following request URL variables are common for all HTTP commands:

| Name | Description |
|------------|---|
| serverRoot | Server base url: hostname+port+base path. Port and base path are OPTIONAL. Example: example.com/exampleAPI |
| apiVersion | Version of the API client wants to use. The value of this variable is defined in section 5.1. |
| userId | User identifier. Examples: tel:+19585550100, acr:pseudonym123 |

See section 6 for a statement on the escaping of reserved characters in URL variables.

6.1.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to Notification Channel, see section 7.

6.1.3 GET

This operation is used for retrieval of active Notification Channels.

6.1.3.1 Example: Retrieve active Notification Channels (Informative)

6.1.3.1.1 Request

```
GET /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels HTTP/1.1
Host: example.com
Accept: application/xml
```

6.1.3.1.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: nnnn
Date: Thu, 04 Jun 2009 02:51:59 GMT

<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationChannelList xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<notificationChannel>
  <clientCorrelator>123</clientCorrelator>
  <applicationTag>myApp</applicationTag>
  <channelType>LongPolling</channelType>
  <channelData xsi:type="nc:LongPollingData">
    <channelURL>http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123/notifications
    </channelURL>
    <maxNotifications>1</maxNotifications>
  </channelData>
  <channelLifetime>7200</channelLifetime>
  <callbackURL>http://example.com/callbackUrl/cbu111</callbackURL>
  <resourceURL>http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123</resourceURL>
</notificationChannel>
<notificationChannel>
  <clientCorrelator>987</clientCorrelator>
  <applicationTag>someOtherApp</applicationTag>
  <channelType>OMAPush</channelType>
  <channelData xsi:type="nc:OMAPushData">
    <appld>x-wap-application:wml.ua</appld>
    <maxNotifications>5</maxNotifications>
  </channelData>
  <channelLifetime>3600</channelLifetime>
  <callbackURL>http://example.com/callbackUrl/cbu222</callbackURL>
  <resourceURL>http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch987</resourceURL>
</notificationChannel>
<resourceURL>http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels</resourceURL>
</nc:notificationChannelList>
```


6.1.4 PUT

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: GET, POST' field in the response as per sections 6.5.5 and 7.4.1 of [RFC7231].

6.1.5 POST

This operation is used for creation of a Notification Channel in order to receive notifications from an Enabler server to which the client has subscribed for notifications.

6.1.5.1 Example: Create Notification Channel (Long Polling method), using tel URI (Informative)

6.1.5.1.1 Request

```
POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels HTTP/1.1
```

```
Host: example.com
```

```
Accept: application/xml
```

```
Content-Type: application/xml
```

```
Content-Length: nnnn
```

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationChannel xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <clientCorrelator>123</clientCorrelator>
  <applicationTag>myApp</applicationTag>
  <channelType>LongPolling</channelType>
  <channelData xsi:type="nc:LongPollingData">
    <maxNotifications>1</maxNotifications>
  </channelData>
  <channelLifetime>7200</channelLifetime>
</nc:notificationChannel>
```

6.1.5.1.2 Response

```
HTTP/1.1 201 Created
```

```
Location: http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123
```

```
Date: Thu, 04 Jun 2009 02:51:59 GMT
```

```
Content-Type: application/xml
```

```
Content-Length: nnnn
```

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationChannel xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <clientCorrelator>123</clientCorrelator>
  <applicationTag>myApp</applicationTag>
  <channelType>LongPolling</channelType>
  <channelData xsi:type="nc:LongPollingData">
    <channelURL> http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123/notifications
    </channelURL>
    <maxNotifications>1</maxNotifications>
  </channelData>
  <channelLifetime>7200</channelLifetime>
  <callbackURL>http://example.com/callbackUrl/cbu111</callbackURL>
  <resourceURL>http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123</resourceURL>
</nc:notificationChannel>
```

6.1.5.2 Example: Create Notification Channel (OMA Push method), using tel URI (Informative)

6.1.5.2.1 Request

```
POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels HTTP/1.1
Host: example.com
Accept: application/xml
Content-Type: application/xml
Content-Length: nnnn

<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationChannel xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <clientCorrelator>987</clientCorrelator>
  <applicationTag>myApp</applicationTag>
  <channelType>OMAPush</channelType>
  <channelData xsi:type="nc:OMAPushData">
    <appld>x-wap-application:wml.ua</appld>
    <maxNotifications>1</maxNotifications>
  </channelData>
  <channelLifetime>7200</channelLifetime>
</nc:notificationChannel>
```

6.1.5.2.2 Response

```
HTTP/1.1 201 Created
Location: http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch987
Date: Thu, 04 Jun 2009 02:51:59 GMT
Content-Type: application/xml
Content-Length: nnnn

<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationChannel xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <clientCorrelator>987</clientCorrelator>
  <applicationTag>myApp</applicationTag>
  <channelType>OMAPush</channelType>
  <channelData xsi:type="nc:OMAPushData">
    <appld>x-wap-application:wml.ua</appld>
    <maxNotifications>1</maxNotifications>
  </channelData>
  <channelLifetime>7200</channelLifetime>
  <callbackURL>http://example.com/callbackUrl/cbu222</callbackURL>
  <resourceURL>http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch987</resourceURL>
</nc:notificationChannel>
```

6.1.5.3 Example: Create Notification Channel (OMA Push method with largeDataPolling), using tel URI (Informative)

6.1.5.3.1 Request

```
POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels HTTP/1.1
Host: example.com
Accept: application/xml
Content-Type: application/xml
```

Content-Length: nnnn

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationChannel xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <clientCorrelator>987</clientCorrelator>
  <applicationTag>myApp</applicationTag>
  <channelType>OMAPush</channelType>
  <channelData xsi:type="nc:OMAPushData">
    <appld>x-wap-application:wml.ua</appld>
    <maxNotifications>1</maxNotifications>
    <largeDataPolling>
      <pollingEnabled>true</pollingEnabled>
      <maxPollingNotifications>10</maxPollingNotifications>
    </largeDataPolling>
  </channelData>
  <channelLifetime>7200</channelLifetime>
</nc:notificationChannel>
```

6.1.5.3.2 Response

HTTP/1.1 201 Created
 Location: http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch987
 Date: Thu, 04 Jun 2015 02:51:59 GMT
 Content-Type: application/xml
 Content-Length: nnnn

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationChannel xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <clientCorrelator>987</clientCorrelator>
  <applicationTag>myApp</applicationTag>
  <channelType>OMAPush</channelType>
  <channelData xsi:type="nc:OMAPushData">
    <appld>x-wap-application:wml.ua</appld>
    <maxNotifications>1</maxNotifications>
    <largeDataPolling>
      <pollingEnabled>true</pollingEnabled>
      <maxPollingNotifications>10</maxPollingNotifications>
    </largeDataPolling>
  </channelData>
  <channelLifetime>7200</channelLifetime>
  <callbackURL>http://example.com/callbackUrl/cbu222</callbackURL>
  <resourceURL>http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch987</resourceURL>
</nc:notificationChannel>
```

6.1.5.4 Example: Create Notification Channel (OMA Push method with LargeDataPolling) not supported (Informative)

6.1.5.4.1 Request

POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels HTTP/1.1
 Host: example.com
 Accept: application/xml
 Content-Type: application/xml

Content-Length: nnnn

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationChannel xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <clientCorrelator>987</clientCorrelator>
  <applicationTag>myApp</applicationTag>
  <channelType>OMAPush</channelType>
  <channelData xsi:type="nc:OMAPushData">
    <appld>x-wap-application:wml.ua</appld>
    <maxNotifications>1</maxNotifications>
    <largeDataPolling>
      <pollingEnabled>true</pollingEnabled>
      <maxPollingNotifications>10</maxPollingNotifications>
    </largeDataPolling>
  </channelData>
  <channelLifetime>7200</channelLifetime>
</nc:notificationChannel>
```

6.1.5.4.2 Response

HTTP/1.1 403 Forbidden
 Date: Thu, 04 Jun 2015 02:51:59 GMT
 Content-Type: application/xml
 Content-Length: nnnn

```
<?xml version="1.0" encoding="UTF-8"?>
<common:requestError xmlns:common="urn:oma:xml:rest:netapi:common:1">
  <policyException>
    <messageId>POL2006</messageId>
    <text>Requested feature %1 is not available</text>
    <variables>LargeDataPolling</variables>
  </policyException>
</common:requestError>
```

6.1.5.5 Example: Create Notification Channel (Long Polling method), using ACR (Informative)

6.1.5.5.1 Request

POST /exampleAPI/notificationchannel/v1/acr%3Apseudonym123/channels HTTP/1.1
 Host: example.com
 Accept: application/xml
 Content-Type: application/xml
 Content-Length: nnnn

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationChannel xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <clientCorrelator>123</clientCorrelator>
  <applicationTag>myApp</applicationTag>
  <channelType>LongPolling</channelType>
  <channelData xsi:type="nc:LongPollingData">
    <maxNotifications>1</maxNotifications>
  </channelData>
```

```
<channelLifetime>7200</channelLifetime>
</nc:notificationChannel>
```

6.1.5.5.2 Response

HTTP/1.1 201 Created

Location: http://example.com/exampleAPI/notificationchannel/v1/acr%3A%2B19585550100/channels/ch123

Date: Thu, 04 Jun 2009 02:51:59 GMT

Content-Type: application/xml

Content-Length: nnnn

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationChannel xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <clientCorrelator>123</clientCorrelator>
  <applicationTag>myApp</applicationTag>
  <channelType>LongPolling</channelType>
  <channelData xsi:type="nc:LongPollingData">
    <channelURL> http://example.com/exampleAPI/notificationchannel/v1/acr%3A%2B19585550100/channels/ch123/notifications
    </channelURL>
    <maxNotifications>1</maxNotifications>
  </channelData>
  <channelLifetime>7200</channelLifetime>
  <callbackURL>http://example.com/callbackUrl/cbu111</callbackURL>
  <resourceURL>http://example.com/exampleAPI/notificationchannel/v1/acr%3A%2B19585550100/channels/ch123</resourceURL>
</nc:notificationChannel>
```

6.1.5.6 Example: Create Notification Channel (WebSockets method), using tel URI (Informative)

6.1.5.6.1 Request

POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels HTTP/1.1

Host: example.com

Accept: application/xml

Content-Type: application/xml

Content-Length: nnnn

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationChannel xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <clientCorrelator>987</clientCorrelator>
  <applicationTag>myApp</applicationTag>
  <channelType>WebSockets</channelType>
  <channelData xsi:type="nc:WebSocketsData">
    <maxNotifications>5</maxNotifications>
  </channelData>
  <channelLifetime>7200</channelLifetime>
</nc:notificationChannel>
```

6.1.5.6.2 Response

HTTP/1.1 201 Created

Location: http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch378

Date: Thu, 28 Jun 2013 02:51:59 GMT

Content-Type: application/xml
Content-Length: nnnn

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationChannel xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <clientCorrelator>987</clientCorrelator>
  <applicationTag>myApp</applicationTag>
  <channelType>WebSockets</channelType>
  <channelData xsi:type="nc:WebSocketsData">
    <channelURL>ws://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch378/ws</channelURL>
    <maxNotifications>5</maxNotifications>
  </channelData>
  <channelLifetime>3600</channelLifetime>
  <callbackURL>http://example.com/callbackUrl/cbu112</callbackURL>
  <resourceURL>http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch378</resourceURL>
</nc:notificationChannel>
```

6.1.5.7 Example: Attempt to create Notification Channel of unsupported type (Informative)

6.1.5.7.1 Request

POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels HTTP/1.1
Host: example.com
Accept: application/xml
Content-Type: application/xml
Content-Length: nnnn

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationChannel xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <clientCorrelator>123</clientCorrelator>
  <applicationTag>myApp</applicationTag>
  <channelType>LongPolling</channelType>
  <channelData xsi:type="nc:LongPollingData">
    <maxNotifications>1</maxNotifications>
  </channelData>
  <channelLifetime>7200</channelLifetime>
</nc:notificationChannel>
```

6.1.5.7.2 Response

HTTP/1.1 400 Bad Request
Date: Thu, 28 Jun 2013 02:51:59 GMT
Content-Type: application/xml
Content-Length: nnnn

```
<?xml version="1.0" encoding="UTF-8"?>
<common:requestError xmlns:common="urn:oma:xml:rest:netapi:common:1">
  <policyException>
    <messageId>POL1023</messageId>
    <text>Notification channel type %1 not supported. Supported types: %2.</text>
    <variables>LongPolling</variables>
    <variables>OMAPush, WebSockets</variables>
```

```
</policyException>
</common:requestError>
```

6.1.6 DELETE

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: GET, POST' field in the response as per sections 6.5.5 and 7.4.1 of [RFC7231].

6.2 Resource: Individual Notification Channel

The resource used is:

http://{serverRoot}/notificationchannel/{apiVersion}/{userId}/channels/{channelId}

This resource is used for management of an individual Notification Channel, operations such as: to retrieve information of the Notification Channel or to remove (terminate) Notification Channel.

6.2.1 Request URL variables

The following request URL variables are common for all HTTP commands:

| Name | Description |
|------------|---|
| serverRoot | Server base url: hostname+port+base path. Port and base path are OPTIONAL. Example: example.com/exampleAPI |
| apiVersion | Version of the API client wants to use. The value of this variable is defined in section 5.1. |
| userId | User identifier. Examples: tel:+19585550100, acr:pseudonym123 |
| channelId | Channel identifier. Example: ch456 |

See section 6 for a statement on the escaping of reserved characters in URL variables.

6.2.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to Notification Channel, see section 7.

6.2.3 GET

This operation is used for retrieval of an individual Notification Channel.

6.2.3.1 Example: Retrieve individual Notification Channel (Informative)

6.2.3.1.1 Request

```
GET /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch456 HTTP/1.1
Host: example.com
Accept: application/xml
```

6.2.3.1.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/xml
```

Content-Length: nnnn

Date: Thu, 04 Jun 2009 02:51:59 GMT

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationChannel xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <clientCorrelator>456</clientCorrelator>
  <applicationTag>someOtherApp</applicationTag>
  <channelType>LongPolling</channelType>
  <channelData xsi:type="nc:LongPollingData">
    <channelURL>http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch456/notifications
    </channelURL>
    <maxNotifications>5</maxNotifications>
  </channelData>
  <channelLifetime>7200</channelLifetime>
  <callbackURL>http://example.com/callbackUrl/cbu333</callbackURL>
  <resourceURL>http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch456</resourceURL>
</nc:notificationChannel>
```

6.2.4 PUT

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: GET, DELETE' field in the response as per sections 6.5.5 and 7.4.1 of [RFC7231].

6.2.5 POST

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: GET, DELETE' field in the response as per sections 6.5.5 and 7.4.1 of [RFC7231].

6.2.6 DELETE

This operation is used for removing an individual Notification Channel. Any outstanding poll request will immediately be responded with a 404 Not Found.

6.2.6.1 Example: Removing Notification Channel (Informative)

6.2.6.1.1 Request

```
DELETE /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch456 HTTP/1.1
Host: example.com
```

6.2.6.1.2 Response

```
HTTP/1.1 204 No Content
Date: Thu, 04 Jun 2009 02:51:59 GMT
```

6.3 Resource: Notification list

The resource URL is provided by the server (channel URL received when the Long Polling Notification Channel or WebSockets Notification Channel is created) and therefore this specification does not make any assumption about the structure of this URL.

For the Long Polling method, this resource is used for retrieval of new notifications from the Notification Server, for which the application has subscribed from the respective Enabler server. At the same time, the server resets the channel lifetime to its original value.

For the WebSockets method, this resource is used to create a WebSockets connection through which the server can send notifications to the client, and which the client can check for connectivity using the ConnCheck and ConnAck messages. This means it is not a resource used in a RESTful manner for WebSockets-based notification channels.

6.3.1 Request URL variables

Provided by the Notification Server in response to request for creation of a Long Polling Notification Channel, if any.

6.3.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to Notification Channel, see section 7.

6.3.3 GET

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per sections 6.5.5 and 7.4.1 of [RFC7231].

6.3.4 PUT

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per sections 6.5.5 and 7.4.1 of [RFC7231].

6.3.5 POST

This operation is used for retrieval of new notifications from the Notification Server if the Notification Channel involved is of Long Polling type.

6.3.5.1 Example 1: Single notification delivered in a NotificationList (Informative)

In this example a presence update is delivered to the application.

6.3.5.1.1 Request

```
POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123/notifications HTTP/1.1
Host: example.com
Accept: application/xml
Content-Type: application/xml
Content-Length: nnnn

<?xml version="1.0" encoding="UTF-8"?>
<nc:longPollingRequestParameters xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1"/>
```

6.3.5.1.2 Response

```
HTTP/1.1 200 OK
Date: Thu, 04 Jun 2009 02:51:59 GMT
Content-Type: application/xml
Connection: close
Content-Length: nnnn

<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationList xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1">
  <pr:presenceNotification xmlns:pr="urn:oma:xml:rest:netapi:presence:1">
    <presentityUserId>tel:+19585550100</presentityUserId>
    <callbackData>1234</callbackData>
```

```

<resourceStatus>Active</resourceStatus>
<presence>
  <person>
    <mood>
      <moodValue>Happy</moodValue>
    </mood>
  </person>
</presence>
<link rel="PresenceSubscription"
  href="http://example.com/exampleAPI/v1/presence/tel%3A%2B19585550101/subscriptions/presenceSubscriptions/
  tel%3A%2B19585550100/sub001"/>
</pr:presenceNotification>
</nc:notificationList>

```

6.3.5.2 Example 2: Multiple notifications delivered (Informative)

In this example a presence update and message notification are delivered to the application.

6.3.5.2.1 Request

```

POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123/notifications HTTP/1.1
Host: example.com
Accept: application/xml
Content-Type: application/xml
Content-Length: nnnn

<?xml version="1.0" encoding="UTF-8"?>
<nc:longPollingRequestParameters xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1"/>

```

6.3.5.2.2 Response

```

HTTP/1.1 200 OK
Date: Thu, 04 Jun 2009 02:51:59 GMT
Content-Type: application/xml
Connection: close
Content-Length: nnnn

<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationList xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1">
  <pr:presenceNotification xmlns:pr="urn:oma:xml:rest:netapi:presence:1">
    <presentityUserId>tel:+19585550100</presentityUserId>
    <callbackData>1234</callbackData>
    <resourceStatus>Active</resourceStatus>
    <presence>
      <person>
        <mood>
          <moodValue>Happy</moodValue>
        </mood>
      </person>
    </presence>
    <link rel="PresenceSubscription"
      href="http://example.com/exampleAPI/v1/presence/tel%3A%2B19585550101/subscriptions/presenceSubscriptions/
      tel%3A%2B19585550100/sub001"/>
  </pr:presenceNotification>
  <mms:inboundMessageNotification xmlns:mms="urn:oma:xml:rest:netapi:messaging:1">
    <inboundMessage>

```

```

<destinationAddress>tel:+19585550100</destinationAddress>
<senderAddress>tel:+19585550101</senderAddress>
<resourceURL>http://example.com/exampleAPI/v1/messaging/inbound/registrations/reg123/messages/msg123
</resourceURL>
<link rel="Subscription" href="http://example.com/exampleAPI/v1/messaging/inbound/subscriptions/sub123"/>
<messageId>msg123</messageId>
<inboundMMSMessage>
  <subject>Who is RESTing on the beach?</subject>
</inboundMMSMessage>
</inboundMessage>
</mms:inboundMessageNotification>
<mms:inboundMessageNotification xmlns:mms="urn:oma:xml:rest:netapi:messaging:1">
  <inboundMessage>
    <destinationAddress>tel:+19585550100</destinationAddress>
    <senderAddress>tel:+19585550102</senderAddress>
    <resourceURL>http://example.com/exampleAPI/v1/messaging/inbound/registrations/reg123/messages/msg1234
    </resourceURL>
    <link rel="Subscription" href="http://example.com/exampleAPI/v1/messaging/inbound/subscriptions/sub123"/>
    <messageId>msg1234</messageId>
    <inboundMMSMessage>
      <subject>Who is still RESTing on the beach?</subject>
    </inboundMMSMessage>
  </inboundMessage>
</mms:inboundMessageNotification>
</nc:notificationList>

```

6.3.5.3 Example 3: Server timeout (Informative)

In this example a Long Polling request times out in the Notification Server before any new notifications from Enabler servers have been received on the server. The server responds with an empty response.

6.3.5.3.1 Request

```

POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123/notifications HTTP/1.1
Host: example.com
Accept: application/xml
Content-Type: application/xml
Content-Length: nnnn

<?xml version="1.0" encoding="UTF-8"?>
<nc:longPollingRequestParameters xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1"/>

```

6.3.5.3.2 Response

```

HTTP/1.1 200 OK
Date: Thu, 04 Jun 2009 02:51:59 GMT
Content-Type: application/xml
Connection: close
Content-Length: nnnn

<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationList xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1"/>

```

6.3.5.4 Example 4: Single notification delivered outside a NotificationList (Informative)

In this example a presence update is delivered to the application.

6.3.5.4.1 Request

```
POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123/notifications HTTP/1.1
Host: example.com
Accept: application/xml
Content-Type: application/xml
Content-Length: nnnn
```

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:longPollingRequestParameters xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1"/>
```

6.3.5.4.2 Response

```
HTTP/1.1 200 OK
Date: Thu, 04 Jun 2009 02:51:59 GMT
Content-Type: application/xml
Connection: close
Content-Length: nnnn
```

```
<?xml version="1.0" encoding="UTF-8"?>
<pr:presenceNotification xmlns:pr="urn:oma:xml:rest:netapi:presence:1">
  <presentityUserId>tel:+19585550100</presentityUserId>
  <callbackData>1234</callbackData>
  <resourceStatus>Active</resourceStatus>
  <presence>
    <person>
      <mood>
        <moodValue>Happy</moodValue>
      </mood>
    </person>
  </presence>
  <link rel="PresenceSubscription"
    href="http://example.com/exampleAPI/v1/presence/tel%3A%2B19585550101/subscriptions/presenceSubscriptions/
    tel%3A%2B19585550100/sub001"/>
</pr:presenceNotification>
```

6.3.6 DELETE

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per sections 6.5.5 and 7.4.1 of [RFC7231].

6.4 Resource: Notification Channel lifetime

The resource used is:

http://{serverRoot}/notificationchannel/{apiVersion}/{userId}/channels/{channelId}/channelLifetime

This resource is used to retrieve and update (“refresh”) the Notification Channel lifetime.

6.4.1 Request URL variables

The following request URL variables are common for all HTTP commands:

| Name | Description |
|------------|---|
| serverRoot | Server base url: hostname+port+base path. Port and base path are OPTIONAL. Example: example.com/exampleAPI |
| apiVersion | Version of the API client wants to use. The value of this variable is defined in section 5.1. |
| userId | User identifier. Examples: tel:+19585550100, acr:pseudonym123 |
| channelId | Channel identifier. Example: ch456 |

See section 6 for a statement on the escaping of reserved characters in URL variables.

6.4.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to Notification Channel, see section 7.

6.4.3 GET

This operation is used for retrieval of the remaining Notification Channel lifetime.

6.4.3.1 Example: Retrieve remaining Notification Channel lifetime(Informative)

6.4.3.1.1 Request

```
GET /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch456/channelLifetime HTTP/1.1
Host: example.com
Accept: application/xml
```

6.4.3.1.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: nnnn
Date: Thu, 28 Jun 2013 02:51:59 GMT

<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationChannelLifetime xmlns:nc="urn:oma:xm1:rest:netapi:notificationchannel:1">
  <channelLifetime>1724</channelLifetime>
</nc:notificationChannelLifetime>
```

6.4.4 PUT

This operation is used for retrieval of the remaining Notification Channel lifetime, i.e. “refresh” the channel.

6.4.4.1 Example: Update Notification Channel lifetime (Informative)

6.4.4.1.1 Request

```
PUT /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch456/channelLifetime HTTP/1.1
Host: example.com
Accept: application/xml
Content-Type: application/xml
Content-Length: nnnn
```

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationChannelLifetime xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1">
  <channelLifetime>7200</channelLifetime>
</nc:notificationChannelLifetime>
```

6.4.4.1.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: nnnn
Date: Thu, 28 Jun 2013 02:51:59 GMT
```

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationChannelLifetime xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1">
  <channelLifetime>3600</channelLifetime>
</nc:notificationChannelLifetime>
```

6.4.5 POST

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: GET, PUT' field in the response as per sections 6.5.5 and 7.4.1 of [RFC7231].

6.4.6 DELETE

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: GET, PUT' field in the response as per sections 6.5.5 and 7.4.1 of [RFC7231].

7. Fault definitions

7.1 Service Exceptions

For common Service Exceptions refer to [REST_NetAPI_Common].

The following additional Service Exception codes are defined for the Notification Channel API.

7.1.1 SVC1012: Simultaneous channel requests not supported

| Name | Description |
|---------------------|---|
| MessageID | SVC1012 |
| Text | Simultaneous channel requests not supported |
| Variables | None |
| HTTP status code(s) | 409 Conflict |

7.2 Policy Exceptions

For common Policy Exceptions refer to [REST_NetAPI_Common].

The following additional Policy Exception codes are defined for the Notification Channel API.

7.2.1 POL1023: Notification channel type not supported

| Name | Description |
|---------------------|---|
| MessageID | POL1023 |
| Text | Notification channel type %1 not supported. Supported types: %2. |
| Variables | %1 – Type of the notification channel %2 – List of supported channel types |
| HTTP status code(s) | 403 Forbidden |

The variable %1 SHALL contain one of the types as defined in section 5.2.3.1, %2 a comma-separated list of one or more types as defined in section 5.2.3.1.

Appendix A. Change History

(Informative)

A.1 Approved Version History

| Reference | Date | Description |
|-----------|------|------------------|
| n/a | n/a | No prior version |

A.2 Draft/Candidate Version 1.0 History

| Document Identifier | Date | Sections | Description |
|---|--|------------------------------------|--|
| Draft Versions: REST_NetAPI_NotificationChannel-V1_0 | 28 Apr 2011 | Many | This is the first version of the document that is based on agreed contribution OMA-ARC-RC-APIs-2011-0040R03-INP_Proposal_for_Notification_Channel_TS. In addition, the document title is updated to address the issues from ARC-2011-A071. |
| | 25 May 2011 | Many | Implemented CR, OMA-ARC-REST-NetAPI-2011-0008-CR_TS_changes_for_NotificationChannel |
| | 02 Jul 2011 | Many | Implemented CRs: OMA-ARC-REST-NetAPI-2011-0092-CR _TS_NotificationChannel_alignment_with_new_template OMA-ARC-REST-NetAPI-2011-0096-CR _TS_NotificationChannel_channelData_type |
| | 25 Jul 2011 | Many | Implemented CRs: OMA-ARC-REST-NetAPI-2011-0135R01-CR _TS_NC_duration_timer OMA-ARC-REST-NetAPI-2011-0137-CR _TS_NC_Additional_SCRs OMA-ARC-REST-NetAPI-2011-0147R01-CR _TS_NC_Appendix_C_and_D |
| | 08 Sep 2011 | Many | Implemented CRs: OMA-ARC-REST-NetAPI-2011-0210R01-CR_NC_XML_examples_for_channel_duration OMA-ARC-REST-NetAPI-2011-0224-CR_NC_telURI_resourceURL_changes |
| | 21 Sep 2011 | Many | Implemented CRs: OMA-ARC-REST-NetAPI-2011-0241-CR_NC_TS_ACR_changes OMA-ARC-REST-NetAPI-2011-0253R02-CR_NC_TS_clarifications_and_tidy_ups |
| | 03 Nov 2011 | Many | Implemented CR: OMA-ARC-REST-NetAPI-2011-0330R02-CR_NC_TS_CONRR_fixing_editorial_comments |
| | 20 Dec 2011 | Many | Implemented CRs: OMA-ARC-REST-NetAPI-2011-0451-CR_NC_TS_CONRR_technical_comments_resolution OMA-ARC-REST-NetAPI-2011-0454-CR_NC_TS_Appendix_G |
| | Candidate Version: REST_NetAPI_NotificationChannel-V1_0 | 17 Jan 2012 | n/a |
| Draft Versions: REST_NetAPI_NotificationChannel-V1_0 | 24 Jul 2012 | 5, 6.1.2, 6.2.2, 6.3.2, 7, G.1.1.3 | Incorporated CR: OMA-TS-REST_NetAPI_NotificationChannel-V1_0-20120117-C_changes_CR0162 Editorial changes |
| | 24 Aug 2012 | 5.2.2.2, C.1 | Incorporated CR: OMA-ARC-REST-NetAPI-2012-0233-CR_NC_TS_issue_20_clientCorrelator_resolution Editorial changes |

| Document Identifier | Date | Sections | Description |
|--|-------------|--|--|
| | 15 Oct 2012 | 2.1, 2.2, 3.3, 4.1, 5, 5.1, 5.2.2.2, 5.2.2.6, 5.2.3.1, 5.3, 5.3.1, 5.3.5, 5.3.6, 5.3.7, 6.1.3.1.2, 6.1.5.1, 6.1.5.1.2, 6.1.5.2, 6.1.5.3, 6.1.5.3.2, 6.3, 6.3.1, 6.3.5, 7.2, B.1.1, B.1.4, C.1.1, C.1.2, C.1.3, C.2, D.1, D.2, D.3, D.4., D.5, G.1.1.1, G.1.1.2, G.1.1.3 | Incorporated CR: OMA-ARC-REST-NetAPI-2012-0254R01- CR_Notification_Channel_support_for_OMA_Push Editorial changes |
| | 08 Nov 2012 | 2.2, 3.2, 5.1, 5.2.2, 5.2.3.1, 6.1.1, 6.2.1, 7.2.1, B.1, B.1.1, B.1.2, B.1.3, B.1.4, C.1, C.2, D.9, F | Incorporated CR: OMA-ARC-REST-NetAPI-2012-0273R01- CR_NC_TS_NotificationList_fixing_element_description Editorial changes |
| | 19 Nov 2012 | 6.3.5.2.2, D.9 | Incorporated CR: OMA-ARC-REST-NetAPI-2012-0276- CR_Notification_Channel_fixing_and_extending_examples |
| | 13 Dec 2012 | 4.1, 6, B, G.1.1.1, G.1.1.3, G.1.2, | Incorporated CR: OMA-ARC-REST-NetAPI-2012-0291- CR_NC_TS_implementing_blueprint_for_authorization Template changed to OMA-TEMPLATE- TS_RESTful_Network_API-20120813-I Editorial changes |
| | 15 Apr 2013 | 2.1, 2.2, 5.3.7, 7.2.1, H | Incorporated CR: OMA-ARC-REST-NetAPI-2013-0019R01- CR_Notification_Server_Push_Enabler_interaction_info Editorial changes |
| | 15 Jul 2013 | H | Incorporated CR: OMA-ARC-REST-NetAPI-2013-0048- CR_NotifChannel_fixing_xml_example_for_push_pap Editorial changes |
| Candidate Version: REST_NetAPI_NotificationChannel-V1_0 | 30 Jul 2013 | n/a | Status changed to Candidate by TP TP Ref # OMA-TP-2013-0224- INP_REST_NetAPI_NotificationChannel_V1_0_ERP_for_Candidat e_re_approval |
| Draft Versions: REST_NetAPI_NotificationChannel-V1_0 | 27 Sep 2013 | 2, 4, 5, 5.1, 5.2.2.2, 5.2.2.3, 5.2.2.8, 5.2.2.9, 5.2.3.1, 5.2.4, 5.3, 5.3.1, 5.3.3, 5.3.8, 5.3.9, 5.3.10, 6.1.5.4, 6.1.5.5, 6.3.1, 6.4, 7.2.1, B.1.1, B.1.4, B.1.5, C, D.5, D.6, D.12, D.13, G.1.1, I | Incorporated CR: OMA-ARC-REST-2013-0052R01-CR_Notif_WebSockets_TS Editorial changes |

| Document Identifier | Date | Sections | Description |
|---------------------|-------------|---|---|
| | 15 Jan 2014 | 5.2.2.8, 5.2.2.9, 5.2.2.10, 5.2.2.11, 6.3, 6.3.5.1, 6.3.5.2, 6.3.5.4, B.1.1, B.1.3, B.1.6, C.2.1, C.2.2, D.9, D.10, D.12, I | Incorporated CRs: OMA-ARC-REST-NetAPI-2014-0001R01-CR_Single_Notification_delivery OMA-ARC-REST-NetAPI-2014-0003R01-CR_Long_Polling_Optionality_Bugfix_NotifChannel_TS OMA-ARC-REST-NetAPI-2014-0005-CR_Notif_No_Headers_No_Compression OMA-ARC-REST-NetAPI-2014-0007R01-CR_Notif_Ping_Pong Editorial changes |
| | 29 Jan 2014 | G.1.1.3 | Incorporated CR: OMA-ARC-REST-NetAPI-2014-0008-CR_Notif_Protected_access_to_Channel_URL Editorial changes |
| | 13 Feb 2014 | 7.2.1 | Incorporated CR: OMA-ARC-REST-NetAPI-2014-0016-CR_Notif_section_7_HTTP_code_fix |
| | 18 Feb 2014 | 5.1 | Incorporated CR: OMA-ARC-REST-NetAPI-2014-0022-CR_Notif_Explaining_meaning_of_Notification_List_for_Websocket ts |
| | 19 Mar 2014 | 5.2.2.5, 5.2.2.6, 5.2.2.8, D.1-D.7 | Incorporated CR: OMA-ARC-REST-NetAPI-2014-0033-CR_xsi_type_NotifChannel |
| | 08 May 2014 | 5, 5.2.2.9, 5.2.2.10, 5.3.11, 5.3.12, 5.3.13, I.3 | Incorporated CR: OMA-ARC-REST-NetAPI-2014-0039R01-CR_Notif_WS_Connection_Re_Establishment Editorial changes |
| | 09 Jun 2014 | 5, 5.3.14, 6.1.5.5.2, 7.1, D.6, I.1, I.4, I.5 | Incorporated CRs: OMA-ARC-REST-NetAPI-2014-0048R02-CR_NC_Multiple_long_polls OMA-ARC-REST-NetAPI-2014-0051-CR_Notification_Channel_Examples_and_Fixes |
| | 14 Oct 2014 | 2.1, 6 | Incorporated CR: OMA-ARC-REST-NetAPI-2014-0076-CR_ACR_reference_in_TS_NotifChnl |
| | 04 Mar 2015 | 2.1, 6.1.4, 6.1.6, 6.2.4, 6.2.5, 6.3.3, 6.3.4, 6.3.6, 6.4.5, 6.4.6, D, G.1.2 | Incorporated CR: OMA-ARC-REST-NetAPI-2015-0021-CR_NotifChnl_TS_updating_references Editorial changes |
| | 11 Jun 2015 | 5, 5.1, 5.2.2.3, 5.2.2.4, 5.2.2.7, 5.2.2.8, 5.3, 5.3.7, 5.3.8, 5.3.9, 6.1.5.3, 6.1.5.4, D.4, D.5, H | Incorporated CRs: OMA-ARC-REST-NetAPI-2015-0059R02-CR_NotifChannel_Dynamic_Polling_channelURL OMA-ARC-REST-NetAPI-2015-0062-CR_NotifChannel_Dynamic_Polling_XmlJson OMA-ARC-REST-NetAPI-2015-0063-CR_LargePollingNotification_Example OMA-ARC-REST-NetAPI-2015-0064R01-CR_NotifChannel_LargeDataPolling_SeqDiagrams |
| | 21 Oct 2015 | 5.2.2.6, 5.3.6 | Incorporated CR: OMA-ARC-REST-NetAPI-2015-0071-CR_Notification_Channel_maxWaitTime Editorial changes |
| | 03 Dec 2015 | 6.3.5.1.2, 6.3.5.2.2, 6.3.5.4.2, A.2, C.2.1.2, C.2.2.2, D.1, H, I.4 | Incorporated CR: OMA-ARC-REST-NetAPI-2015-0095-CR_Notification_Channel_fixing_validation_errors Editorial changes |

| Document Identifier | Date | Sections | Description |
|--|-------------|----------|---|
| Candidate Version: REST_NetAPI_NotificationChannel-V1_0 | 22 Dec 2015 | n/a | Status changed to Candidate by TP TP Ref # OMA-TP-2015-0220- INP_REST_NetAPI_NotificationChannel_V1_0_ERP_for_Candidat e_re_approval |

Appendix B. Static Conformance Requirements (Normative)

The notation used in this appendix is specified in [SCRRULES].

B.1 SCR for REST.NC Server

| Item | Function | Reference | Requirement |
|-------------------------|--|------------|-------------|
| REST-NC-SUPPORT-S-001-M | Support for the RESTful Notification Channel API | 5, 6 | |
| REST-NC-SUPPORT-S-002-M | Support for the XML request & response format | 6 | |
| REST-NC-SUPPORT-S-003-M | Support for the JSON request & response format | 6 | |
| REST-NC-SUPPORT-S-004-O | Support for the application/x-www-form-urlencoded format | Appendix C | |

B.1.1 SCR for REST.NC.Channels Server

| Item | Function | Reference | Requirement |
|--------------------------|--|-----------|--|
| REST-NC-CHANNELS-S-001-M | Support for management of Notification Channels | 6.1 | REST-NC-CHANNELS-S-003-O OR REST-NC-CHANNELS-S-004-O OR REST-NC-CHANNELS-S-006-O |
| REST-NC-CHANNELS-S-002-O | Retrieving a list of Notification Channels - GET | 6.1.3 | |
| REST-NC-CHANNELS-S-003-O | Creating a Long Polling Notification Channel – POST (XML or JSON) | 6.1.5 | REST-NC-LONGPOLL-S-001-O |
| REST-NC-CHANNELS-S-004-O | Creating a OMA Push Notification Channel – POST (XML or JSON) | 6.1.5 | REST-NC-OMAPUSH-S-001-O AND REST-NC-REFRESH-S-001-O |
| REST-NC-CHANNELS-S-005-O | Creating a Notification Channel – POST (application/x-www-form-urlencoded) | C.1 | |
| REST-NC-CHANNELS-S-006-O | Creating a WebSockets Notification Channel – POST (XML or JSON) | 6.1.5 | REST-NC-REFRESH-S-001-O AND REST-NC-WEBSOCKET-S-001-O |

B.1.2 SCR for REST.NC.IndividualChannel Server

| Item | Function | Reference | Requirement |
|----------------------------|---|-----------|-------------|
| REST-NC-INDCHANNEL-S-001-M | Support for access to individual Notification Channel | 6.2 | |
| REST-NC-INDCHANNEL-S-002-M | Retrieving Notification Channel information - GET | 6.2.3 | |
| REST-NC-INDCHANNEL-S-003-M | Terminating Notification Channel – DELETE | 6.2.6 | |

B.1.3 SCR for REST.NC.LongPolling Server

| Item | Function | Reference | Requirement |
|--------------------------|---|-----------|--------------------------|
| REST-NC-LONGPOLL-S-001-O | Support for access to notifications via long polling | 6.3 | REST-NC-LONGPOLL-S-002-O |
| REST-NC-LONGPOLL-S-002-O | Retrieving notifications from the server using Long Polling – POST (XML or JSON) | 6.3.5 | |
| REST-NC-LONGPOLL-S-003-O | Retrieving notifications from the server using Long Polling – POST (application/x-www-form-urlencoded) | C.2 | |

B.1.4 SCR for REST.NC.OMAPush Server

| Item | Function | Reference | Requirement |
|-------------------------|---|-----------|-------------|
| REST-NC-OMAPUSH-S-001-O | Acting as a Push Initiator by pushing notifications to OMA Push Enabler | - | |

B.1.5 SCR for REST.NC.Refresh Server

| Item | Function | Reference | Requirement |
|-------------------------|---|-----------|-------------------------|
| REST-NC-REFRESH-S-001-O | Support for Refresh of Notification Channel | 6.4 | REST-NC-REFRESH-S-003-O |
| REST-NC-REFRESH-S-002-O | Retrieving ChannelLifetime – GET | 6.4.3 | |
| REST-NC-REFRESH-S-003-O | Updating ChannelLifetime – PUT | 6.4.4 | |

B.1.6 SCR for REST.NC.WebSockets Server

| Item | Function | Reference | Requirement |
|-------------------------|--|-----------|-------------------------|
| REST-NC-WEBSOCK-S-001-O | Allow opening a WebSockets connection, serve notifications through this connection | I.1 | REST-NC-WEBSOCK-S-002-O |
| REST-NC-WEBSOCK-S-002-O | Support the connCheck/connAck mechanism | I.3 | |

Appendix C. Application/x-www-form-urlencoded Request Format for POST Operations (Normative)

This section defines a format for the RESTful Notification Channel REST API requests where the body of the request is encoded using the application/x-www-form-urlencoded MIME type.

Note: only the request body is encoded as application/x-www-form-urlencoded, the response is still encoded as XML or JSON depending on the preference of the client and the capabilities of the server. Names and values MUST follow the application/x-www-form-urlencoded character escaping rules from [W3C_URLENC].

The encoding is defined below for the following Notification Channel REST operations which are based on POST requests:

- Create a Notification Channel
- Retrieve notifications from Notification Server

The application/ x-www-form-urlencoded request format is not supported for a WebSockets-based Notification Channel.

C.1 Creating a Notification Channel

This operation is used to create a Notification Channel, see section 6.1.5.

The request parameters are as follows:

| Name | Type/Values | Optional | Description |
|------------------|-------------|----------|---|
| clientCorrelator | xsd:string | Yes | <p>A correlator that the client can use to tag this particular resource representation during a request to create a resource on the server.</p> <p>This element SHOULD be present. Note: this allows the client to recover from communication failures during resource creation and therefore avoids duplicate channel creation in such situations.</p> <p>In case the field is present, the server SHALL not alter its value, and SHALL provide it as part of the representation of this resource. In case the field is not present, the server SHALL NOT generate it.</p> |
| applicationTag | xsd:string | Yes | <p>A tag that the client MAY use to tag this particular resource on the server. In case the field is present, the server SHALL not alter its value, and SHALL provide it as part of the representation of this resource. In case the field is not present, the server SHALL NOT generate it.</p> |
| channelType | xsd:string | No | <p>Specifies the type of Notification Channel to be used (method that will be used to receive new notifications on the channel). Allowed string values are defined in 5.2.3.1.</p> |
| maxNotifications | xsd:int | Yes | <p>Defines the maximum number of notifications that may be delivered in a notification list.</p> <p>If not specified, a default value specified by the server policy will apply, and the server SHOULD include that value in the response to the client.</p> |

| | | | |
|-----------------|---------|-----|---|
| channelLifetime | xsd:int | Yes | <p>Lifetime (duration) of Notification Channel in seconds.</p> <p>Client can specify desired lifetime of Notification Channel in POST request when creating Notification Channel, however the server in the response to the request may change the desired lifetime according to its server policy.</p> <p>If the element is not present in the POST request, a default channel lifetime specified by server policy will apply.</p> <p>The server SHALL always include the channel lifetime in the response either when it was modified compared to what the client requested, or a default channel lifetime is used.</p> |
|-----------------|---------|-----|---|

C.1.1 Example 1: Create Notification Channel (Long Polling method), using tel URI (Informative)

C.1.1.1 Request

```
POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: nnnn
Accept: application/xml

clientCorrelator=123&
applicationTag=myApp&
channelType=LongPolling&
maxNotifications=1&
channelLifetime=7200
```

C.1.1.2 Response

```
HTTP/1.1 201 Created
Location: http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123
Date: Thu, 04 Jun 2009 02:51:59 GMT
Content-Type: application/xml
Content-Length: nnnn

<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationChannel xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <clientCorrelator>123</clientCorrelator>
  <applicationTag>myApp</applicationTag>
  <channelType>LongPolling</channelType>
  <channelData xsi:type="nc:LongPollingData">
    <channelURL>http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123/notifications
    </channelURL>
    <maxNotifications>1</maxNotifications>
  </channelData>
  <channelLifetime>7200</channelLifetime>
  <callbackURL>http://example.com/callbackUrl/cbu111</callbackURL>
```

```
<resourceURL>http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123</resourceURL>
</nc.notificationChannel>
```

C.1.2 Example 2: Create Notification Channel (OMA Push method), using tel URI (Informative)

C.1.2.1 Request

```
POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: nnnn
Accept: application/xml

clientCorrelator=987&
applicationTag=myApp&
channelType=OMAPush&
appld=x-wap-application:wml.ua&
maxNotifications=1&
channelLifetime=7200
```

C.1.2.2 Response

```
HTTP/1.1 201 Created
Location: http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch987
Date: Thu, 04 Jun 2009 02:51:59 GMT
Content-Type: application/xml
Content-Length: nnnn

<?xml version="1.0" encoding="UTF-8"?>
<nc.notificationChannel xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <clientCorrelator>987</clientCorrelator>
  <applicationTag>myApp</applicationTag>
  <channelType>OMAPush</channelType>
  <channelData xsi:type="nc:OMAPushData">
    <appld>x-wap-application:wml.ua</appld>
    <maxNotifications>1</maxNotifications>
  </channelData>
  <channelLifetime>7200</channelLifetime>
  <callbackURL>http://example.com/callbackUrl/cbu222</callbackURL>
  <resourceURL>http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch987</resourceURL>
</nc.notificationChannel>
```

C.1.3 Example 3: Create Notification Channel, using ACR (Informative)

C.1.3.1 Request

```
POST /exampleAPI/notificationchannel/v1/acr%3Apseudonym123/channels HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: nnnn
```


Accept: application/xml

clientCorrelator=123&
 applicationTag=myApp&
 channelType=LongPolling&
 maxNotifications=1&
 channelLifetime=7200

C.1.3.2 Response

HTTP/1.1 201 Created

Location: http://example.com/exampleAPI/notificationchannel/v1/acr%3A%2B123/channels/ch123

Date: Thu, 04 Jun 2009 02:51:59 GMT

Content-Type: application/xml

Content-Length: nnnn

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationChannel xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <clientCorrelator>123</clientCorrelator>
  <applicationTag>myApp</applicationTag>
  <channelType>LongPolling</channelType>
  <channelData xsi:type="nc:LongPollingData">
    <channelURL>http://example.com/exampleAPI/notificationchannel/v1/acr%3A%2B123/channels/ch123/notifications
    </channelURL>
    <maxNotifications>1</maxNotifications>
  </channelData>
  <channelLifetime>7200</channelLifetime>
  <callbackURL>http://example.com/callbackUrl/cbu111</callbackURL>
  <resourceURL>http://example.com/exampleAPI/notificationchannel/v1/acr%3A%2B123/channels/ch123</resourceURL>
</nc:notificationChannel>
```

C.2 Retrieving notifications from the Notification Server

This operation is used to retrieve new notifications from the Notification Server if the Notification Channel involved is of Long Polling type, see section 6.3.5.

The request parameters are as follows:

| Name | Type/Values | Optional | Description |
|------------------------------|-------------|----------|--|
| longPollingRequestParameters | (empty) | No | Provides the body of the request, which is an empty string in this version of specification. |

C.2.1 Example 1: Single notification delivered in a NotificationList (Informative)

C.2.1.1 Request

POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123/notifications HTTP/1.1
 Host: example.com
 Content-Type: application/x-www-form-urlencoded
 Content-Length: nnnn
 Accept: application/xml

```
longPollingRequestParameters=
```

C.2.1.2 Response

```
HTTP/1.1 200 OK
Date: Thu, 04 Jun 2009 02:51:59 GMT
Content-Type: application/xml
Connection: close
Content-Length: nnnn

<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationList xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1">
  <pr:presenceNotification xmlns:pr="urn:oma:xml:rest:netapi:presence:1">
    <presentityUserId>tel:+19585550100</presentityUserId>
    <callbackData>1234</callbackData>
    <resourceStatus>Active</resourceStatus>
    <presence>
      <person>
        <mood>
          <moodValue>Happy</moodValue>
        </mood>
      </person>
    </presence>
    <link rel="PresenceSubscription"
      href="http://example.com/exampleAPI/v1/presence/tel%3A%2B19585550101/subscriptions/presenceSubscriptions/
tel%3A%2B19585550100/sub001"/>
  </pr:presenceNotification>
</nc:notificationList>
```

C.2.2 Example 2: Single notification delivered outside a NotificationList (Informative)

C.2.2.1 Request

```
POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123/notifications HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: nnnn
Accept: application/xml

longPollingRequestParameters=
```

C.2.2.2 Response

```
HTTP/1.1 200 OK
Date: Thu, 04 Jun 2009 02:51:59 GMT
Content-Type: application/xml
Connection: close
Content-Length: nnnn

<?xml version="1.0" encoding="UTF-8"?>
<pr:presenceNotification xmlns:pr="urn:oma:xml:rest:netapi:presence:1">
  <presentityUserId>tel:+19585550100</presentityUserId>
```

```
<callbackData>1234</callbackData>
<resourceStatus>Active</resourceStatus>
<presence>
  <person>
    <mood>
      <moodValue>Happy</moodValue>
    </mood>
  </person>
</presence>
<link rel="PresenceSubscription"
  href="http://example.com/exampleAPI/v1/presence/tel%3A%2B19585550101/subscriptions/presenceSubscriptions/
  tel%3A%2B19585550100/sub001"/>
</pr:presenceNotification>
```

Appendix D. JSON examples (Informative)

JSON (JavaScript Object Notation) is a lightweight, text-based, language-independent data interchange format. It provides a simple means to represent basic name-value pairs, arrays and objects. JSON is relatively trivial to parse and evaluate using standard JavaScript libraries, and hence is suited for REST invocations from browsers or other processors with JavaScript engines. Further information on JSON can be found at [RFC7159].

The following examples show the request and response for various operations using the JSON data format. The examples follow the XML to JSON serialization rules in [REST_NetAPI_Common]. A JSON response can be obtained by using the content type negotiation mechanism specified in [REST_NetAPI_Common].

For full details on the operations themselves please refer to the section number indicated.

D.1 Retrieve active Notification Channels (section 6.1.3.1)

Request:

```
GET /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels HTTP/1.1
Host: example.com
Accept: application/json
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Thu, 04 Jun 2009 02:51:59 GMT

{"notificationChannelList": {
  "notificationChannel": [
    {
      "applicationTag": "myApp",
      "callbackURL": "http://example.com/callbackUrl/cbu111",
      "channelData": {
        "channelURL": "http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123/notifications",
        "maxNotifications": "1",
      },
      "channelLifetime": "7200",
      "channelType": "LongPolling",
      "clientCorrelator": "123",
      "resourceURL": "http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123"
    },
    {
      "applicationTag": "someOtherApp",
      "callbackURL": "http://example.com/callbackUrl/cbu222",
      "channelData": {
        "appld": "x-wap-application:wml.ua",
        "maxNotifications": "5",
      },
      "channelLifetime": "3600",
      "channelType": "OMAPush",
      "clientCorrelator": "987",
      "resourceURL": "http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch987"
    }
  ]
}
```

```
  ],  
  "resourceURL": "http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels"  
}}
```

D.2 Create Notification Channel (Long Polling method), using tel URI (section 6.1.5.1)

Request:

```
POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels HTTP/1.1  
Host: example.com  
Content-Type: application/json  
Content-Length: nnnn  
Accept: application/json  
  
{  
  "notificationChannel": {  
    "applicationTag": "myApp",  
    "channelData": {  
      "maxNotifications": "1",  
    },  
    "channelLifetime": "7200",  
    "channelType": "LongPolling",  
    "clientCorrelator": "123"  
  }  
}
```

Response:

```
HTTP/1.1 201 Created  
Location: http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123  
Date: Thu, 04 Jun 2009 02:51:59 GMT  
Content-Type: application/json  
Content-Length: nnnn  
  
{  
  "notificationChannel": {  
    "applicationTag": "myApp",  
    "callbackURL": "http://example.com/callbackUrl/cbu111",  
    "channelData": {  
      "channelURL": "http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123/notifications",  
      "maxNotifications": "1",  
    },  
    "channelLifetime": "7200",  
    "channelType": "LongPolling",  
    "clientCorrelator": "123",  
    "resourceURL": "http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123"  
  }  
}
```

D.3 Create Notification Channel (OMA Push method), using tel URI (section 6.1.5.2)

Request:

```
POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels HTTP/1.1
Host: example.com
Content-Type: application/json
Content-Length: nnnn
Accept: application/json

{"notificationChannel": {
  "applicationTag": "myApp",
  "channelData": {
    "appld": "x-wap-application:wml.ua",
    "maxNotifications": "1",
  },
  "channelLifetime": "7200",
  "channelType": "OMAPush",
  "clientCorrelator": "987"
}}
```

Response:

```
HTTP/1.1 201 Created
Location: http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch987
Date: Thu, 04 Jun 2009 02:51:59 GMT
Content-Type: application/json
Content-Length: nnnn

{"notificationChannel": {
  "applicationTag": "myApp",
  "callbackURL": "http://example.com/callbackUrl/cbu222",
  "channelData": {
    "appld": "x-wap-application:wml.ua",
    "maxNotifications": "1",
  },
  "channelLifetime": "7200",
  "channelType": "OMAPush",
  "clientCorrelator": "987",
  "resourceURL": "http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch987"
}}
```

D.4 Create Notification Channel (OMA Push method with largeDataPolling), using tel URI (section 6.1.5.3)

Request:

```
POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Content-Length: nnnn
```

```
{
  "notificationChannel": {
    "applicationTag": "myApp",
    "channelData": {
      "appld": "x-wap-application:wml.ua",
      "largeDataPolling": {
        "maxPollingNotifications": "10",
        "pollingEnabled": "true"
      },
      "maxNotifications": "1",
      "type": "nc:OMAPushData"
    },
    "channelLifetime": "7200",
    "channelType": "OMAPush",
    "clientCorrelator": "987"
  }
}
```

Response:

```
HTTP/1.1 201 Created
Location: http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch987
Date: Thu, 04 Jun 2015 02:51:59 GMT
Content-Type: application/json
Content-Length: nnnn

{"notificationChannel": {
  "applicationTag": "myApp",
  "callbackURL": "http://example.com/callbackUrl/cbu222",
  "channelData": {
    "appld": "x-wap-application:wml.ua",
    "largeDataPolling": {
      "maxPollingNotifications": "10",
      "pollingEnabled": "true"
    },
    "maxNotifications": "1",
    "type": "nc:OMAPushData"
  },
  "channelLifetime": "7200",
  "channelType": "OMAPush",
  "clientCorrelator": "987",
  "resourceURL": "http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch987"
}
```

D.5 Create Notification Channel (OMA Push method with LargeDataPolling) not supported (section 6.1.5.4)

Request:

```
POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Content-Length: nnnn
```

```
{
  "notificationChannel": {
    "applicationTag": "myApp",
    "channelData": {
      "appld": "x-wap-application:wml.ua",
      "largeDataPolling": {
        "maxPollingNotifications": "10",
        "pollingEnabled": "true"
      },
      "maxNotifications": "1",
      "type": "nc:OMAPushData"
    },
    "channelLifetime": "7200",
    "channelType": "OMAPush",
    "clientCorrelator": "987"
  }
}
```

Response:

```
HTTP/1.1 403 Forbidden
Date: Thu, 04 Jun 2015 02:51:59 GMT
Content-Type: application/json
Content-Length: nnnn

{"requestError": {"policyException": {
  "messageId": "POL2006",
  "text": "Requested feature %1 is not available",
  "variables": "LargeDataPolling"
}}}
```

D.6 Create Notification Channel (Long Polling method), using ACR (section 6.1.5.5)

Request:

```
POST /exampleAPI/notificationchannel/v1/acr%3Apseudonym123/channels HTTP/1.1
Host: example.com:80
Content-Type: application/json
Content-Length: nnnn
Accept: application/json

{"notificationChannel": {
  "applicationTag": "myApp",
  "channelData": {
    "maxNotifications": "1",
  },
  "channelLifetime": "7200",
  "channelType": "LongPolling",
  "clientCorrelator": "123"
}}
```

Response:

```
HTTP/1.1 201 Created
Location: http://example.com/exampleAPI/notificationchannel/v1/acr%3Apseudonym123/channels/ch123
```


Date: Thu, 04 Jun 2009 02:51:59 GMT
Content-Type: application/json
Content-Length: nnnn

```
{"notificationChannel": {  
  "applicationTag": "myApp",  
  "callbackURL": "http://example.com/callbackUrl/cbu111",  
  "channelData": {  
    "channelURL": "http://example.com/exampleAPI/notificationchannel/v1/acr%3A%2B19585550100/channels/ch123/notifications ",  
    "maxNotifications": "1",  
  },  
  "channelLifetime": "7200",  
  "channelType": "LongPolling",  
  "clientCorrelator": "123",  
  "resourceURL": "http://example.com/exampleAPI/notificationchannel/v1/acr%3A%2B19585550100/channels/ch123"  
}}
```

D.7 Create Notification Channel (WebSockets method), using tel URI (section 6.1.5.6)

Request:

```
POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels HTTP/1.1  
Host: example.com  
Content-Type: application/json  
Content-Length: nnnn  
Accept: application/json
```

```
{"notificationChannel": {  
  "applicationTag": "myApp",  
  "channelData": {  
    "maxNotifications": "5",  
  },  
  "channelLifetime": "7200",  
  "channelType": "WebSockets",  
  "clientCorrelator": "987"  
}}
```

Response:

```
HTTP/1.1 201 Created  
Location: http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch378  
Date: Thu, 28 Jun 2013 02:51:59 GMT  
Content-Type: application/json  
Content-Length: nnnn
```

```
{"notificationChannel": {  
  "applicationTag": "myApp",  
  "callbackURL": "http://example.com/callbackUrl/cbu112",  
  "channelData": {  
    "channelURL": "ws://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch378/ws",  
    "maxNotifications": "5",  
  },  
  "channelLifetime": "3600",  
}}
```

```
"channelType": "WebSockets",
"clientCorrelator": "987",
"resourceURL": "http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch378"
}}
```

D.8 Example: Attempt to create Notification Channel of unsupported type (section 6.1.5.7)

Request:

```
POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Content-Length: nnnn

{"notificationChannel": {
  "applicationTag": "myApp",
  "channelData": {
    "maxNotifications": "1",
  },
  "channelLifetime": "7200",
  "channelType": "LongPolling",
  "clientCorrelator": "123"
}}
```

Response:

```
HTTP/1.1 400 Bad Request
Date: Thu, 28 Jun 2013 02:51:59 GMT
Content-Type: application/json
Content-Length: nnnn

{"requestError": {"policyException": {
  "messageId": "POL1023",
  "text": "Notification channel type %1 not supported. Supported types: %2.",
  "variables": [
    "LongPolling",
    "OMAPush, WebSockets"
  ]
}}}
}}
```

D.9 Retrieve individual Notification Channel (section 6.2.3.1)

Request:

```
GET /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch456 HTTP/1.1
Host: example.com
Accept: application/json
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Thu, 04 Jun 2009 02:51:59 GMT

{"notificationChannel": {
  "applicationTag": "someOtherApp",
  "callbackURL": "http://example.com/callbackUrl/cbu333",
  "channelData": {
    "channelURL": "http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch456/notifications",
    "maxNotifications": "5",
  },
  "channelLifetime": "7200",
  "channelType": "LongPolling",
  "clientCorrelator": "456",
  "resourceURL": "http://example.com/exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch456"
}}
```

D.10 Removing Notification Channel (section 6.2.6.1)

Request:

```
DELETE /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch456 HTTP/1.1
Host: example.com
Accept: application/json
```

Response:

```
HTTP/1.1 204 No Content
Date: Thu, 04 Jun 2009 02:51:59 GMT
```

D.11 Single notification delivered in a NotificationList (section 6.3.5.1)

Request:

```
POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123/notifications HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Content-Length: nnnn

{"longPollingRequestParameters": null}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Thu, 04 Jun 2009 02:51:59 GMT
```

```
{
  "notificationList": {
    "presenceNotification": {
      "callbackData": "1234",
      "link": {
        "href": "http://example.com/exampleAPI/v1/presence/tel%3A%2B19585550101/subscriptions/presenceSubscriptions/tel%3A%2B19585550100/sub001",
        "rel": "PresenceSubscription"
      },
      "presence": {
        "person": {
          "mood": {
            "moodValue": "Happy"
          }
        }
      },
      "presentityUserId": "tel:+19585550100",
      "resourceStatus": "Active"
    }
  }
}
```

D.12 Multiple notifications delivered (section 6.3.5.2)

Request:

```
POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123/notifications HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Content-Length: nnnn

{"longPollingRequestParameters": null}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Thu, 04 Jun 2009 02:51:59 GMT

{"notificationList": [
  {
    "inboundMessageNotification": {
      "inboundMessage": {
        "destinationAddress": "tel:+19585550100",
        "inboundMMSMessage": {
          "subject": "Who is RESTing on the beach?"
        },
        "link": {
          "href": "http://example.com/exampleAPI/v1/messaging/inbound/subscriptions/sub123",
          "rel": "Subscription"
        }
      },
      "messageId": "msg123",
      "resourceURL": "http://example.com/exampleAPI/v1/messaging/inbound/registrations/reg123/messages/msg123",
      "senderAddress": "tel:+19585550101"
    }
  },
  {
    "inboundMessageNotification": {
      "inboundMessage": {
        "destinationAddress": "tel:+19585550100",
        "inboundMMSMessage": {
          "subject": "Who is still RESTing on the beach?"
        },
        "link": {
          "href": "http://example.com/exampleAPI/v1/messaging/inbound/subscriptions/sub123",
          "rel": "Subscription"
        }
      },
      "messageId": "msg1234",
      "resourceURL": "http://example.com/exampleAPI/v1/messaging/inbound/registrations/reg123/messages/msg1234",
      "senderAddress": "tel:+19585550102"
    }
  }
],
  "presenceNotification": {
```

```
"callbackData": "1234",
"link": {
  "href": "http://example.com/exampleAPI/v1/presence/tel%3A%2B19585550101/subscriptions/presenceSubscriptions/
tel%3A%2B19585550100/sub001",
  "rel": "PresenceSubscription"
},
"presence": {"person": {"mood": {"moodValue": "Happy"}}},
"presentityUserId": "tel:+19585550100",
"resourceStatus": "Active"
}
}}
```

D.13 Server timeout (section 6.3.5.3)

Request:

```
POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123/notifications HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Content-Length: nnnn

{"longPollingRequestParameters": null}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Thu, 04 Jun 2009 02:51:59 GMT

{"notificationList": null}
```

D.14 Single notification delivered in a NotificationList (section 6.3.5.4)

Request:

```
POST /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch123/notifications HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Content-Length: nnnn

{"longPollingRequestParameters": null}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Thu, 04 Jun 2009 02:51:59 GMT
```

```
{
  "presenceNotification": {
    "callbackData": "1234",
    "link": {
      "href": "http://example.com/exampleAPI/v1/presence/tel%3A%2B19585550101/subscriptions/presenceSubscriptions/
tel%3A%2B19585550100/sub001",
      "rel": "PresenceSubscription"
    },
    "presence": { "person": { "mood": { "moodValue": "Happy" } } },
    "presentityUserId": "tel:+19585550100",
    "resourceStatus": "Active"
  }
}
```

D.15 Retrieve remaining Notification Channel lifetime (section 6.4.3.1)

Request:

```
GET /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch456/channelLifetime HTTP/1.1
Host: example.com
Accept: application/json
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Thu, 28 Jun 2013 02:51:59 GMT

{"notificationChannelLifetime": {"channelLifetime": "1724"}}
```

D.16 Update Notification Channel lifetime (section 6.4.4.1)

Request:

```
PUT /exampleAPI/notificationchannel/v1/tel%3A%2B19585550100/channels/ch456/channelLifetime HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Content-Length: nnnn

{"notificationChannelLifetime": {"channelLifetime": "7200"}}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Thu, 28 Jun 2013 02:51:59 GMT

{"notificationChannelLifetime": {"channelLifetime": "3600"}}
```

Appendix E. Operations mapping to a pre-existing baseline specification (Informative)

As this specification does not have a baseline specification, this appendix is empty

Appendix F. Light-weight resources (Informative)

As this version of the specification does not define any Light-weight Resources, this appendix is empty.

Appendix G. Authorization aspects (Normative)

This appendix specifies how to use the RESTful Notification Channel API in combination with some authorization frameworks.

G.1 Use with OMA Authorization Framework for Network APIs

The RESTful Notification Channel API MAY support the authorization framework defined in [Autho4API_10].

A RESTful Notification Channel API supporting [Autho4API_10]:

- SHALL conform to section D.1 of [REST_NetAPI_Common];
- SHALL conform to this section G.1.

G.1.1 Scope values

G.1.1.1 Definitions

In compliance with [Autho4API_10], an authorization server serving clients requests for getting authorized access to the resources exposed by the RESTful Notification Channel API:

- SHALL support the scope values defined in the table below;
- MAY support scope values not defined in this specification.

| Scope value | Description | For one-time access token |
|---|---|---------------------------|
| oma_rest_notificationchannel.all_{apiVersion} | Provide access to all defined operations on the resources in this version of the API. The {apiVersion} part of this identifier SHALL have the same value as the “apiVersion” URL variable which is defined in section 5.1. This scope value is the union of the other scope values listed in next rows of this table. | No |
| oma_rest_notificationchannel.longpoll | Provide access to all operations defined for using Long Polling on Notification Channel. | No |
| oma_rest_notificationchannel.omapush | Provide access to all operations defined for using OMA Push on Notification Channel. | No |
| oma_rest_notificationchannel.websockets | Provide access to all operations defined for using WebSockets on Notification Channel. | No |

Table 1: Scope values for RESTful Notification Channel API

G.1.1.2 Downscoping

In the case where the client requests authorization for “oma_rest_notificationchannel.all_{apiVersion}” scope, the authorization server and/or resource owner MAY restrict the granted scope to some of the following scope values:

- “oma_rest_notificationchannel.longpoll”

- “oma_rest_notificationchannel.omapush”
- “oma_rest_notificationchannel.websockets”

G.1.1.3 Mapping with resources and methods

Tables in this section specify how the scope values defined in section G.1.1.1 for the RESTful Notification Channel API map to the REST resources and methods of this API. In these tables, the root “oma_rest_notificationchannel.” of scope values is omitted for readability reasons.

| Resource | URL Base URL: http://{serverRoot}/notificationchannel/{apiVersion} | Section reference | HTTP verbs | | | |
|---------------------------------|--|-------------------|---|-----|---|---|
| | | | GET | PUT | POST | DELETE |
| Notification Channels | / {userId}/channels | 6.1 | all_{apiVersion} or longpoll or omapush or websockets | n/a | all_{apiVersion} or longpoll or omapush or websockets | n/a |
| Individual Notification Channel | / {userId}/channels/{channelId} | 6.2 | all_{apiVersion} or longpoll or omapush or websockets | n/a | n/a | all_{apiVersion} or longpoll or omapush or websockets |
| Notification Channel lifetime | / {userId}/channels/{channelId}/channelLifetime | 6.4 | all_{apiVersion} or omapush or websockets | n/a | n/a | all_{apiVersion} or omapush or websockets |

Table 2: Required scope values for: Management of Notification Channel

| Resource | URL <specified by the server> | Section reference | HTTP verbs | | | |
|-------------------|---|-------------------|------------|-----|------------------------------|--------|
| | | | GET | PUT | POST | DELETE |
| Notification list | <Resource URL is provided by the server when the Notification Channel is created> | 6.3 | n/a | n/a | all_{apiVersion} or longpoll | n/a |

Table 3: Required scope values for: Retrieval of notifications from Notification Server

G.1.2 Use of 'acr:auth'

This section specifies the use of 'acr:auth' in place of an end user identifier in a resource URL path.

An 'acr' URI of the form 'acr:auth', where 'auth' is a reserved keyword MAY be used to avoid exposing a real end user identifier in the resource URL path.

A client MAY use 'acr:auth' in a resource URL in place of the {userId} resource URL variable in the resource URL path, when the RESTful Notification Channel API is used in combination with [Autho4API_10].

In the case the RESTful Notification Channel API supports [Autho4API_10], the server:

- SHALL accept 'acr:auth' as a valid value for the resource URL variable {userId}.

SHALL conform to [REST_NetAPI_Common] section 5.8.1.1 regarding the processing of 'acr:auth'

Appendix H. Notification server - Push enabler interaction

(Informative)

This appendix provides further information on Notification Server Interaction with the Push Enabler for forwarding the event to the targeted device and application on the device.

In delivering the Push MESSAGE, the Notification Server has several implementation options:

- a) Delivery via a Push Proxy Gateway (PPG) as defined in [OMA_PUSH], using either the Push Access Protocol [PushPAP] or the PushREST API [PushREST]. Depending upon the size of the notification and the intended bearer(s), the Notification Server may deliver the notification content directly, or provide an indirect reference to the notification content which the application may retrieve upon receiving the Push message. How the Notification Server determines the supported notification content size is unspecified, but as a general rule any notification content of less than 512 compressed/binary bytes or less than 2K uncompressed bytes should be deliverable via any OMA Push-OTA bearer binding.

Push PAP Example: Delivering Indirect Reference to Notification Content Available from Enabler Server

```
POST /pap HTTP/1.1
Content-Length: 1041
Content-Type: multipart/related; boundary=PMasdfgkjhqwert; type="application/xml"
Host: ppg.example.com:9002
Connection: close

--PMasdfgkjhqwert
Content-Type: application/xml

<?xml version="1.0"?>
<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP 1.0//EN" "http://www.wapforum.org/DTD/pap_1.0.dtd">
<pap product-name="OMA-Notification-Server-1.0">
<push-message push-id="1079025501:mms_12.25.203.86_1223_1078969978_21:134:0:1"
source-reference="notserver.example.com">
  <address address-value="WAPPUSH="+14255551212/TYP=PLMN@example.com"/>
  <quality-of-service bearer="SMS" bearer-required="false" delivery-method="unconfirmed" network="GSM"
network-required="false"/>
</push-message>
</pap>
--PMasdfgkjhqwert

Content-Length: 373
Content-Type: text/vnd.wap.si
X-Wap-Application-Id: myapp.com/f7adaea2-2bfe-1869-8314-1cc82b1aa4b8

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE si PUBLIC "-//WAPFORUM//DTD SI 1.0//EN" "http://www.wapforum.org/DTD/SI.dtd">
<si>
  <indication href="http://mmsapi.example.com/notification/myapp.com/f7adaea2-2bfe-1869-8314-1cc82b1aa4b8"
si-id = "1079025501:mms_12.25.203.86_1223_1078969978_21:134:0:1"
  >Your message was delivered.</indication>
</si>

--PMasdfgkjhqwert--
```

Push PAP Example: Delivering a “largePollNotification”

```

POST /pap HTTP/1.1
Content-Length: 1041
Content-Type: multipart/related; boundary=PMasdfgkljkhqwert; type="application/xml"
Host: ppg.example.com:9002
Connection: close

--PMasdfgkljkhqwert
Content-Type: application/xml

<?xml version="1.0"?>
<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP 1.0//EN" "http://www.wapforum.org/DTD/pap_1.0.dtd">
<pap product-name="OMA-Notification-Server-1.0">
<push-message push-id="1079025501:mms_12.25.203.86_1223_1078969978_21:134:0:1"
source-reference="notserver.example.com">
  <address address-value="WAPPUSH="+14255551212/TYPE=PLMN@example.com"/>
  <quality-of-service bearer="SMS" bearer-required="false" delivery-method="unconfirmed" network="GSM"
network-required="false"/>
</push-message>
</pap>
--PMasdfgkljkhqwert

Content-Length: nnn
Content-Type: application/xml
X-Wap-Application-Id: myapp.com/f7adaea2-2bfe-1869-8314-1cc82b1aa4b8

<?xml version="1.0" encoding="UTF-8"?>
<nc:largePollingNotification xmlns:nc="urn:oma+xml:rest:netapi:notificationchannel:1">
  <channelURL>http://example.com/largePollingChannel/123</channelURL>
  <channelExpiry>2015-06-03T21:32:52Z</channelExpiry>
</nc:largePollingNotification>

--PMasdfgkljkhqwert—

```

PushREST Example: Directly Delivering Notification Content

```

PUT /ExampleAPI/push/v1/pi1.example.com/pushMessages/id123 HTTP/1.1
Host: ppg.example.com:9002
Content-Type: multipart/related; boundary=qwertyuioplkhgfsazxcvbnm; type="application/json"
Accept: application/json
Content-Length: 2794
Connection: close

--qwertyuioplkhgfsazxcvbnm
Content-Type: application/json

{"push-message": {
  "address": [
    {"address-value": "wappush="+14255551212/type=plmn@example.com " }
  ],
  "deliver-before-timestamp": "2010-11-08T18:13:51.0Z",
  "ppg-notify-requested-to": "http://notserver.example.com/Push/f7adaea2-2bfe-1869-8314-1cc82b1aa4b8",
  "progress-notes-requested": "true",
  "quality-of-service": {"priority": "medium", "bearer": "SMS" "bearer-required": "false" "delivery-method":

```

```

    "confirmed" "network": "GSM" "network-required": "false"},
    "source-reference": "notserver.example.com"
  }}

--qwertyuioplkjhgfdsazxcvbnm
Content-Type: application/xml
X-Wap-Application-Id: myapp.com/f7adaea2-2bfe-1869-8314-1cc82b1aa4b8

<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationList xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1">
  <pr:presenceNotification xmlns:pr="urn:oma:xml:rest:netapi:presence:1">
    <presentityUserId>tel:+19585550100</presentityUserId>
    <callbackData>1234</callbackData>
    <resourceStatus>Active</resourceStatus>
    <presence>
      <person>
        <mood>
          <moodValue>Happy</moodValue>
        </mood>
      </person>
    </presence>
    <link rel="PresenceSubscription"
href="http://example.com/exampleAPI/v1/presence/tel%3A%2B19585550101/subscriptions/presenceSubscriptions/
tel%3A%2B19585550100/sub001"/>
  </pr:presenceNotification>
  <mms:inboundMessageNotification xmlns:mms="urn:oma:xml:rest:netapi:messaging:1">
    <inboundMessage>
      <destinationAddress>tel:+19585550100</destinationAddress>
      <senderAddress>tel:+19585550101</senderAddress>
      <resourceURL>http://example.com/exampleAPI/v1/messaging/inbound/registrations/reg123/messages/msg123
</resourceURL>
      <link rel="Subscription" href="http://example.com/exampleAPI/v1/messaging/inbound/subscriptions/sub123"/>
      <messageId>msg123</messageId>
      <inboundMMSMessage>
        <subject>Who is RESTing on the beach?</subject>
      </inboundMMSMessage>
    </inboundMessage>
  </mms:inboundMessageNotification>
  <mms:inboundMessageNotification xmlns:mms="urn:oma:xml:rest:netapi:messaging:1">
    <inboundMessage>
      <destinationAddress>tel:+19585550100</destinationAddress>
      <senderAddress>tel:+19585550102</senderAddress>
      <resourceURL>http://example.com/exampleAPI/v1/messaging/inbound/registrations/reg123/messages/msg1234
</resourceURL>
      <link rel="Subscription" href="http://example.com/exampleAPI/v1/messaging/inbound/subscriptions/sub123"/>
      <messageId>msg1234</messageId>
      <inboundMMSMessage>
        <subject>Who is still RESTing on the beach?</subject>
      </inboundMMSMessage>
    </inboundMessage>
  </mms:inboundMessageNotification>
</nc:notificationList>
--qwertyuioplkjhgfdsazxcvbnm--

```

- b) Direct delivery of an OMA Push message using a Push-OTA (Over the Air) binding supported by the target device. How the Notification Server determines the supported Push-OTA bindings is unspecified. For details of Push-OTA bearer bindings, see [PushOTA].

Appendix I. Notification delivery using WebSockets (Normative)

I.1 Delivery mechanism

Subsequently to the creation of a NotificationChannel with channelType=WebSockets, the server responds with a NotificationChannel data structure that includes a “channelURL” element which provides a URI of scheme “ws:” or “wss”. The application creates a WebSockets connection on that URL returned, and listens on it for event notifications. As part of the WebSockets handshake, the application MUST use the subprotocol identifier defined in the next section.

When sending a single notification in the channel, the server MAY choose to encapsulate it in a NotificationList or send it direct as a root element. When sending multiple notifications in the channel at once, the server MUST encapsulate them in a NotificationList.

Occasionally, the server MAY choose to send an empty NotificationList through the connection. In contrast to the Long Polling delivery method, receiving an empty NotificationList SHALL NOT be interpreted by the client as an attempt of the server to close the connection.

I.2 Subprotocol registration

Implementations compliant with this specification MUST use “notificationchannel-netapi-rest.openmobilealliance.org” in the “Sec-WebSocket-Protocol” header [RFC6455].

The protocol identifier is registered with [IANA] with the following information:

Subprotocol Identifier: notificationchannel-netapi-rest.openmobilealliance.org

Subprotocol Common Name: OMA RESTful Network API for Notification Channel

Subprotocol Definition: OMA RESTful Network API for Notification Channel V 1.0, Open Mobile Alliance, OMA-TS-REST_NetAPI_NotificationChannel-V1_0, available from <http://www.openmobilealliance.org>

Reference: OMNA - Open Mobile Naming Authority <OMA-OMNA@mail.openmobilealliance.org>

I.3 Connection checking and keep-alive

The WebSockets protocol [RFC6455] defines a pair of messages called Ping and Pong which can be used to check whether a connection is still functioning, and to keep alive the connection. However, the WebSockets API [W3C_WebSock] does not expose these messages. Therefore, web applications would have to rely on the underlying infrastructure for connection checking and keep-alive purposes.

For web applications which want to deploy an application-layer mechanism for that, this specification defines the elements “ConnCheck” and “ConnAck” (see sections 5.2.2.11 and 5.2.2.12).

Servers that support WebSockets-based Notification channels MUST support receiving the connCheck element, and MUST return a connAck element as response. Additionally, the server MUST reset the channel lifetime upon successful delivery of the connAck element, and MUST return the new channel lifetime in the connAck element.

Client support for receiving the connCheck element is RECOMMENDED for clients that support WebSockets-based Notification channels. Clients that support the connCheck message MUST respond to a connCheck message with a connAck message without the channelLifetime element instantiated. On receipt of such a connAck message, the server MUST reset the channelLifetime to the value announced in the previous connCheck message.

If the server that has sent the connCheck message does not receive the connAck message before the next connCheck message is due to be sent, it MAY send additional connCheck messages and SHOULD consider the current Websockets connection faulty and close it if none of these messages is answered before an implementation-specific timeout occurs.

If the server has announced in a previous connCheck message that it intends a new connCheck message at a certain time and that message does not arrive within a sensible time interval, the client SHOULD consider the current Websockets connection faulty, close it and open another one.

A server that receives a request to open a Websockets connection from a client even though there exists a connection with that client SHOULD

- assume that the client has no intention of using the existing connection any longer
- refrain from sending any more notifications or connCheck messages over that connection
- use only the new connection for sending any messages, apart from outstanding connAck messages
- attempt to close the existing connection

Note: Clients and servers can also use the Ping-Pong mechanism that is defined by [RFC6455] to initiate connectivity checking and keep-alive. However, this is outside the scope of this specification as this mechanism is not exposed to the application layer.

I.4 Notification Payload Examples – XML format (Informative)

I.4.1 Example: Single notification delivered in a NotificationList

In this example a presence update is delivered to the application.

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationList xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1">
  <pr:presenceNotification xmlns:pr="urn:oma:xml:rest:netapi:presence:1">
    <presentityUserId>tel:+19585550100</presentityUserId>
    <callbackData>1234</callbackData>
    <resourceStatus>Active</resourceStatus>
    <presence>
      <person>
        <mood>
          <moodValue>Happy</moodValue>
        </mood>
      </person>
    </presence>
    <link rel="PresenceSubscription"
      href="http://example.com/exampleAPI/v1/presence/tel%3A%2B19585550101/subscriptions/presenceSubscriptions/
tel%3A%2B19585550100/sub001"/>
  </pr:presenceNotification>
</nc:notificationList>
```

I.4.2 Example: Multiple notifications delivered

In this example a presence update and message notification are delivered to the application.

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:notificationList xmlns:nc="urn:oma:xml:rest:netapi:notificationchannel:1">
  <pr:presenceNotification xmlns:pr="urn:oma:xml:rest:netapi:presence:1">
    <presentityUserId>tel:+19585550100</presentityUserId>
    <callbackData>1234</callbackData>
    <resourceStatus>Active</resourceStatus>
    <presence>
      <person>
        <mood>
```

```

    <moodValue>Happy</moodValue>
  </mood>
</person>
</presence>
<link rel="PresenceSubscription"
href="http://example.com/exampleAPI/v1/presence/tel%3A%2B19585550101/subscriptions/presenceSubscriptions/
tel%3A%2B19585550100/sub001"/>
</pr:presenceNotification>
<mms:inboundMessageNotification xmlns:mms="urn:oma:xml:rest:netapi:messaging:1">
  <inboundMessage>
    <destinationAddress>tel:+19585550100</destinationAddress>
    <senderAddress>tel:+19585550101</senderAddress>
    <resourceURL>http://example.com/exampleAPI/v1/messaging/inbound/registrations/reg123/messages/msg123
    </resourceURL>
    <link rel="Subscription" href="http://example.com/exampleAPI/v1/messaging/inbound/subscriptions/sub123"/>
    <messageId>msg123</messageId>
    <inboundMMSMessage>
      <subject>Who is RESTing on the beach?</subject>
    </inboundMMSMessage>
  </inboundMessage>
</mms:inboundMessageNotification>
<mms:inboundMessageNotification xmlns:mms="urn:oma:xml:rest:netapi:messaging:1">
  <inboundMessage>
    <destinationAddress>tel:+19585550100</destinationAddress>
    <senderAddress>tel:+19585550102</senderAddress>
    <resourceURL>http://example.com/exampleAPI/v1/messaging/inbound/registrations/reg123/messages/msg1234
    </resourceURL>
    <link rel="Subscription" href="http://example.com/exampleAPI/v1/messaging/inbound/subscriptions/sub123"/>
    <messageId>msg1234</messageId>
    <inboundMMSMessage>
      <subject>Who is still RESTing on the beach?</subject>
    </inboundMMSMessage>
  </inboundMessage>
</mms:inboundMessageNotification>
</nc.notificationList>

```

1.4.3 Example: Single notification delivered outside a NotificationList

In this example a presence update is delivered to the application.

```

<?xml version="1.0" encoding="UTF-8"?>
<pr:presenceNotification xmlns:pr="urn:oma:xml:rest:netapi:presence:1">
  <presentityUserId>tel:+19585550100</presentityUserId>
  <callbackData>1234</callbackData>
  <resourceStatus>Active</resourceStatus>
  <presence>
    <person>
      <mood>
        <moodValue>Happy</moodValue>
      </mood>
    </person>
  </presence>
  <link rel="PresenceSubscription"
href="http://example.com/exampleAPI/v1/presence/tel%3A%2B19585550101/subscriptions/presenceSubscriptions/
tel%3A%2B19585550100/sub001"/>

```

```
</pr:presenceNotification>
```

I.5 Notification Payload Examples – JSON (Informative)

I.5.1 Single notification delivered in a NotificationList

```
{
  "notificationList": {
    "presenceNotification": {
      "callbackData": "1234",
      "link": {
        "href": "http://example.com/exampleAPI/v1/presence/tel%3A%2B19585550101/subscriptions/presenceSubscriptions/tel%3A%2B19585550100/sub001",
        "rel": "PresenceSubscription"
      },
      "presence": {
        "person": {
          "mood": {
            "moodValue": "Happy"
          }
        }
      },
      "presentityUserId": "tel:+19585550100",
      "resourceStatus": "Active"
    }
  }
}
```

I.5.2 Multiple notifications delivered

```
{
  "notificationList": [
    {
      "inboundMessageNotification": {
        "inboundMessage": {
          "destinationAddress": "tel:+19585550100",
          "inboundMMSMessage": {
            "subject": "Who is RESTing on the beach?"
          },
          "link": {
            "href": "http://example.com/exampleAPI/v1/messaging/inbound/subscriptions/sub123",
            "rel": "Subscription"
          },
          "messageId": "msg123",
          "resourceURL": "http://example.com/exampleAPI/v1/messaging/inbound/registrations/reg123/messages/msg123",
          "senderAddress": "tel:+19585550101"
        }
      },
      "inboundMessageNotification": {
        "inboundMessage": {
          "destinationAddress": "tel:+19585550100",
          "inboundMMSMessage": {
            "subject": "Who is still RESTing on the beach?"
          },
          "link": {
            "href": "http://example.com/exampleAPI/v1/messaging/inbound/subscriptions/sub123",
            "rel": "Subscription"
          },
          "messageId": "msg1234",
          "resourceURL": "http://example.com/exampleAPI/v1/messaging/inbound/registrations/reg123/messages/msg1234",
          "senderAddress": "tel:+19585550102"
        }
      },
      "presenceNotification": {
        "callbackData": "1234",
        "link": {
          "href": "http://example.com/exampleAPI/v1/presence/tel%3A%2B19585550101/subscriptions/presenceSubscriptions/tel%3A%2B19585550100/sub001",
          "rel": "PresenceSubscription"
        },
        "presence": {
          "person": {
            "mood": {
              "moodValue": "Happy"
            }
          }
        },
        "presentityUserId": "tel:+19585550100",
        "resourceStatus": "Active"
      }
    }
  ]
}
```

```
}}
```

I.5.3 Single notification delivered in a NotificationList

```
{ "presenceNotification": {  
  "callbackData": "1234",  
  "link": {  
    "href": "http://example.com/exampleAPI/v1/presence/tel%3A%2B19585550101/subscriptions/presenceSubscriptions/  
tel%3A%2B19585550100/sub001",  
    "rel": "PresenceSubscription"  
  },  
  "presence": { "person": { "mood": { "moodValue": "Happy" } } },  
  "presentityUserId": "tel:+19585550100",  
  "resourceStatus": "Active"  
}}
```