SHARED MARKET PROTOCOL (SMP) TECHNICAL GUIDE

PROVIDES PARTICIPANTS WITH THE TECHNICAL SPECIFICATIONS FOR THE DELIVERY OF B2B TRANSACTIONS USING THE E-HUB

Version: 1.02

Published: 9 June 2017

AEMO
AUSTRALIAN ENERGY MARKET OPERATOR

IMPORTANT NOTICE

Purpose

The Australian Energy Market Operator (AEMO) has prepared this document to provide information about the Shared Market Protocol eHub. It provides participants with the technical specifications for the delivery of B2B Transactions using the e-Hub's APIs so they can develop their own systems.

Privacy and legal notices

The material in this publication may be used in accordance with the privacy and legal notices on AEMO's website.

Trademark Notices

Microsoft, Windows and SQL Server are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the US and other countries.

Distribution

Available to the public.

Document Identification

Application owner: Manager, Metering

Document owner: Manager, Market Systems

Prepared by: Power of Choice SMP team

Last update: Friday, 9 June 2017 7:36 PM

Documents made obsolete

The release of this document changes only the version of Shared Market Protocol (SMP) Technical Guide.

Further Information

For further information, please visit AEMO's website www.aemo.com.au or contact:

AEMO Information and Support Hub Phone: 1300 AEMO 00 (1300 236 600) and follow the prompts.
Email: supporthub@aemo.com.au

Feedback

Your feedback is important and helps us improve our services and products. To suggest improvements, please contact AEMO's Information and Support Hub.

Version Release History

Version	Date	Comments
1.00	24/04/2017	Initial version created to provide information regarding the e-Hub and enhancements/changes made as part of the National Electricity Amendment (Updating the Electricity B2B Framework) Rule 2016 No. 6.
1.01	19/05/2017	Updates: Feedback from SWG Meeting (01st May 2017) Additional information / updates

© AEMO 2017

Version	Date	Comments
1.02	09/06/2017	Updates: Additional information / updates

© AEMO 2017

CONTENTS

1 INTRODUCTION 2
1.1 Overview and purpose 2
1.2 How to use this guide 2
1.3 Related documents 2

2 E-HUB INFRASTRUCTURE 4
2.1 e-Hub services 5
2.2 Accessing the e-Hub services 6

3 SECURITY 8
3.1 Custom Ports 8
3.2 SSL 8
3.3 SSL Certificates 8

4 E-HUB APIS 9
4.1 RESTful Architecture 9
4.2 API Authentication & authorisation 10
4.3 API format 11

5 VALIDATION OVERVIEW 16

6 API SETTINGS 17
6.1 Push-Push pattern 17
6.2 Push-Pull pattern 17
6.3 Common header parameters 19
6.4 HubMessageManagement 20
6.5 B2BMessagingAsync 25
6.6 B2BMessagingSync 32
6.7 B2BMessagingPull 36
6.8 P2PMessagingSync 44
6.9 Connection & read timeout settings 48
6.10 Summary of API specifications 49

7 PEER-TO-PEER (P2P) MESSAGING PAYLOADS 51
7.1 Exchange free-form text 51
7.2 Exchange single attachment 52
7.3 Exchanging multiple attachments 53
7.4 aseXML PAYLOAD 54

© AEMO 2017

8 E-HUB CONFIGURATION 56
8.1 Participant-specific settings 56

9 SCENARIOS 57
9.1 B2B Asynchronous Message Exchange 57
9.2 Hub Inquiry 69
9.3 File naming (FTP to API) 71
9.4 Synchronous Messaging 74
9.5 Push-Pull messaging 82
9.6 Peer-to-Peer (P2P) messaging 90

10 VALIDATIONS 99

11 API SPECIFIC ASEXML PAYLOADS 100
11.1 HubQueueReport 100
11.2 HubFlowControlReport 101
11.3 HubFlowControlAlertNotification 102
11.4 PayloadExceptionAlert 103

APPENDIX A. ASYNC SORD EXAMPLE 105

APPENDIX B. E-HUB SYSTEMS 106

APPENDIX C. OBTAINING AN SSL CERTIFICATE 107

APPENDIX D. OBTAINING AN API KEY 110

TABLES

Table 1 – Related Documents 2
Table 2 – Illustration of how API Keys are provisioned 10
Table 3 – API Definition 11
Table 4 – API Methods 11
Table 5 – List of e-Hub APIs 17
Table 6 – e-Hub API Details - HubMessageManagment 20
Table 7 – e-Hub API Details – HubMessageManagment Resource & Methods 21
Table 8 – e-Hub HubMessageManagement Resource: alerts 21
Table 9 – e-Hub HubMessageManagement (/alerts): Query String & Request Header Parameters 21
Table 10 – e-Hub HubMessageManagement (/alerts): Response parameters 22
Table 11 – e-Hub HubMessageManagement Resource: ping 22
Table 12 – e-Hub HubMessageManagement (/ping): Request parameters 23
Table 13 – e-Hub HubMessageManagement (/ping): Response parameters 23
Table 14 – Participant API Details 24
Table 15 – Message Management - Participant API Details – Resources & Methods 24
Table 16 – Participant Resources: alerts 24
Table 17 – Participant HubMessageManagement: Request parameters 25
Table 18 – Participant HubMessageManagement: Response parameters 25
Table 19 – B2BMessagingAsync API - e-Hub API Details 26

© AEMO 2017

Table 20 – B2BMessagingAsync API - e-Hub API Details – Resources & Methods 26
Table 21 – e-Hub Resources: messages....26
Table 22 – e-Hub B2BMessagingAsync API (/messages): Request header parameters 27
Table 23 – e-Hub B2BMessagingAsync API (/messages): Response header parameters....27
Table 24 – e-Hub Resources: messageAcknowledgements 27
Table 25 – e-Hub B2BMessagingAsync API: Request parameters for /messageAcknowledgements 28
Table 26 – e-Hub B2BMessagingAsync API: Response parameters for /messageAcknowledgements....28
Table 27 – e-Hub Resources: queues 29
Table 28 – e-Hub B2BMessagingAsync API: Request parameters & query string parameters for /queues 29
Table 29 – e-Hub B2BMessagingAsync API: Response parameters for /queues 29
Table 30 – Participant API details....30
Table 31 – Participant Async API Details – Resources & Methods 30
Table 32 – Participant Resources: messages 30
Table 33 – Participant Async API: Request header parameters for /messages 31
Table 34 – Participant Async API: Response parameters for /messages....31
Table 35 – Participant Resources: messageAcknowledgements 31
Table 36 – Participant Async API: Request header parameters for /messageAcknowledgements 32
Table 37 – Participant Async API: Response parameters for /messageAcknowlegements....32
Table 38 – e-Hub API Details – B2BMessagingSync 33
Table 39 – e-Hub API Details – B2BMessagingSync – Resources & Methods 33
Table 40 – e-Hub Resources: /messages....33
Table 41 – e-Hub B2BMessagingSync API: Request header parameters for /messages 34
Table 42 – e-Hub B2BMessagingSync API: Response parameters for /messages....34
Table 43 – Participant API Details 35
Table 44 – Participant API Details – Sync API – Resources & Methods 35
Table 45 – Participant Resources: messages 35
Table 46 – Participant Sync API: Request header parameters for /messages 36
Table 47 – Participant Sync API: Response parameters for /messages....36
Table 48 – e-Hub API Details (Pull)....37
Table 49 – e-Hub API Details (Pull) – Resources & Methods....37
Table 50 – e-Hub Resources: messages (Pull) 37
Table 51 – e-Hub B2BMessagingPull API: Request header parameters for /messages 38
Table 52 – e-Hub B2BMessagingPull API: Response parameters for /messages....38
Table 53 – e-Hub Resources: messageAcknowledgements (Pull) - POST 39
Table 54 – e-Hub B2BMessagingPull API: Request parameters for /messageAcknowledgements 39
Table 55 – e-Hub B2BMessagingPull API: Response parameters for /messageAcknowledgements 40
Table 56 – e-Hub Resources: messageAcknowledgements (Pull) - DELETE....40
Table 57 – e-Hub B2BMessagingPull API: Request & query string parameters for /messageAcknowledgements (DEL)....40
Table 58 – e-Hub B2BMessagingPull API: Response parameters for /messageAcknowledgements (DEL) 41
Table 59 – e-Hub Resources: queues (Pull)....41
Table 60 – e-Hub B2BMessagingPull API: Request & query string parameters for /queues....42
Table 61 – e-Hub B2BMessagingPull API: Response parameters for /queues 43
Table 62 – P2P Offerings 44
Table 63 – e-Hub API Details (P2P) 44
Table 64 – e-Hub API Details (P2P) – Resources & Methods....45
Table 65 – e-Hub Resources: messages (P2P) 45
Table 66 – e-Hub P2PMessagingSync API: Request header parameters for /messages 45
Table 67 – e-Hub P2PMessagingSync API: Response parameters for /messages....46
Table 68 – Participant API Details (P2P) 46
Table 69 – Participant API Details (P2P) – Resources & Methods....46
Table 70 – Participant Resources: messages (P2P) 46

© AEMO 2017

Table 71 – Participant P2P API: Request header parameters for /messages 47
Table 72 – Participant P2P API: Response parameters for /messages....47
Table 73 – Summary of e-Hub & Participant API Implementation for each API Pattern 50
Table 74 – HTTP Request call Setting – Exchange Free-Form Text....51
Table 75 – HTTP Request call Setting – Exchange Single Attachment....52
Table 76 – HTTP Request call Setting – Exchange Multiple Attachments....53
Table 77 – HubQueueReport....100
Table 78 – HubQueueReport....101
Table 79 – HubQueueReport....102
Table 80 – HubQueueReport....103

FIGURES

Figure 1 – High-level e-Hub infrastructure 4
Figure 2 Context Diagram....5
Figure 3 – API Connection Diagram 7
Figure 4 – Push-Push message pattern 17
Figure 5 – Push-Pull message pattern....17
Figure 6 – B2BMessagingAsync API – Conceptual View 18
Figure 7 – B2BMessagingSync API – Conceptual View 19
Figure 8 – ASynchronous time out parameter setting....48
Figure 9 – Synchronous time out parameter setting 49
Figure 10 2BMessagingAsync – Normal-processing Scenario 57
Figure 11 – B2BMessagingAsync – Normal-processing Scenario (Queued)....59
Figure 12 – B2BMessagingAsync - Technical Validation Failure (e-Hub)....60
Figure 13 – B2BMessagingAsync - Payload Validation Failure 60
Figure 14 – Technical Validation Failure (Recipient) 62
Figure 15 – B2BMessagingAsync – Recipient Acknowledgement Payload Validation Failure (e-Hub) 63
Figure 16 – B2BMessagingAsync API - MACK Validation Failure (Initiator)....65
Figure 17 – B2BMessagingAsync API - Recipient Unavailable (Stop File Process)....66
Figure 18 – B2BMessagingAsync API - Request Hub Queue Details 68
Figure 19 – B2BMessagingAsync - Request Hub Queue Details – Technical Validation Failure 69
Figure 20 – Request Stop File Details 69
Figure 21 – Request Stop File Details – Technical Validation Failure 70
Figure 22 – File naming (FTP to API) 71
Figure 23 – File naming (API to FTP) 71
Figure 24 – Manage In-Flights FTP to API 72
Figure 25 – Manage In-Flights API to FTP 73
Figure 26 – Normal Processing (sync)....74
Figure 27 – Normal Processing – Notified Party (sync) 75
Figure 28 – Recipient TACK message (Sync) 76
Figure 29 – Technical Validation Failure – e-Hub (sync) 77
Figure 30 – Payload Validation Failure (sync) 77
Figure 31 – Recipient on FTP/not opted for sync 77
Figure 32 – Recipient Unavailable (sync) 78
Figure 33 – Recipient Connection Timeout (sync)....78
Figure 34 – Recipient HTTPS Rejection (sync) 79
Figure 35 – Payload Validation Failure (Recipient) (sync)....80
Figure 36 – Incorrect Message Payload – MACK status ='Accept' (sync)....81
Figure 37 – Normal Processing (Recipient opted) (pull) 82
Figure 38 – Normal Processing (Initiator opted) (pull) 84

© AEMO 2017

Figure 39 – Normal Processing (Empty queue) (pull) 86
Figure 40 – Normal Processing (maxResults not passed) (pull) 86
Figure 41 – Invalid Query String Parameters (Recipient opted) (pull) 87
Figure 42 – Incoming aseXML Schema Validation Failure (pull) 88
Figure 43 – Incorrect Payload (pull) 88
Figure 44 – Message Queue Exception (Recipient opted) (pull) 89
Figure 45 – Message Queue Exception (Initiator opted) (pull) 89
Figure 46 – Peer-to-Peer data exchange using FTP 90
Figure 47 – Peer-to-Peer data exchange using B2BMessagingAsync API 91
Figure 48 – Peer-to-Peer data exchange using P2PMessagingSync API 91
Figure 49 – Peer-to-Peer data exchange using P2PMessagingSync API – MACK accept & TACK accept 93
Figure 50 – Peer-to-Peer data exchange – TACK status of Reject 94
Figure 51 – Peer-to-Peer data exchange – Technical Validation Failure 94
Figure 52 – Peer-to-Peer data exchange – Payload Validation Failure 95
Figure 53 – Peer-to-Peer data exchange – Payload with only attachments 95
Figure 54 – Peer-to-Peer data exchange – Attachment list mismatch 95
Figure 55 – Peer-to-Peer data exchange – Recipient not opted for P2P Services 96
Figure 56 – Peer-to-Peer data exchange – Recipient Unavailable 96
Figure 57 – Peer-to-Peer data exchange – Recipient Connection Timeout 97
Figure 58 – Peer-to-Peer data exchange – Recipient HTTP Rejection 98
Figure 59 – Asynchronous message exchange – Service Order Process (example) 105

© AEMO 2017

1 INTRODUCTION

1.1 Overview and purpose

The Shared Market Protocol (SMP) is an initiative to provide a standard for communications to support smart meters. AEMO has extended AEMO's e-Hub functionality to support this initiative in line with the B2B Procedure: Technical Delivery Specifications. AEMO's e-Hub now includes an API gateway and API portal to support web service communication between participants.

AEMO has produced this SMP Technical Guide (guide) to provide participants with the technical specifications for the delivery of B2B Transactions using the e-Hub API's. This detail is to assist participants with developing their own systems to utilise these APIs.

1.2 How to use this guide

- The B2B Procedure Technical Delivery Specification (TDS) references APIs as 'webservices'. APIs referenced in this document and 'webservices' referenced in the TDS are synonymous.
- If there are any inconsistencies between this guide, the B2B Procedure: Technical Delivery Specification, any other B2B Procedure, or the National Electricity Rules (NER) the procedures and rules prevail.

1.3 Related documents

Table 1 – Related Documents

Title	Location
B2B e-Hub Accreditation Process	http://www.aemo.com.au/Electricity/National-Electricity-Market-NEM/Retail-and-metering/Business-to-business-procedures
B2B Mapping to aseXML	http://www.aemo.com.au/Electricity/National-Electricity-Market-NEM/Retail-and-metering/Business-to-business-procedures
B2B Procedure: Technical Delivery Specifications	http://www.aemo.com.au/Electricity/National-Electricity-Market-NEM/Retail-and-metering/Business-to-business-procedures
Connecting to AEMO's IT Systems	https://www.aemo.com.au/-/media/Files/Electricity/NEM/IT-Systems-and-Change/2016/Connecting-to-AEMOs-Electricity-IT-Systems.pdf
Guide to Information Systems v2.03	https://www.aemo.com.au/-/media/Files/IT_Changes/Guide-to-Information-Systems.pdf
Guide to MSATS B2B	http://www.aemo.com.au/media/Files/Other/electricityops/Guide_to_MSATS_B2B_v10.00.pdf

© AEMO 2017

Title	Location
Guidelines for Development of A Standard for Energy Transactions in XML (aseXML), also known as the 'aseXML Guidelines'.	http://www.aemo.com.au/Electricity/National-Electricity-Market-NEM/IT-systems-and-change/aseXML_standards/aseXML-Guidelines

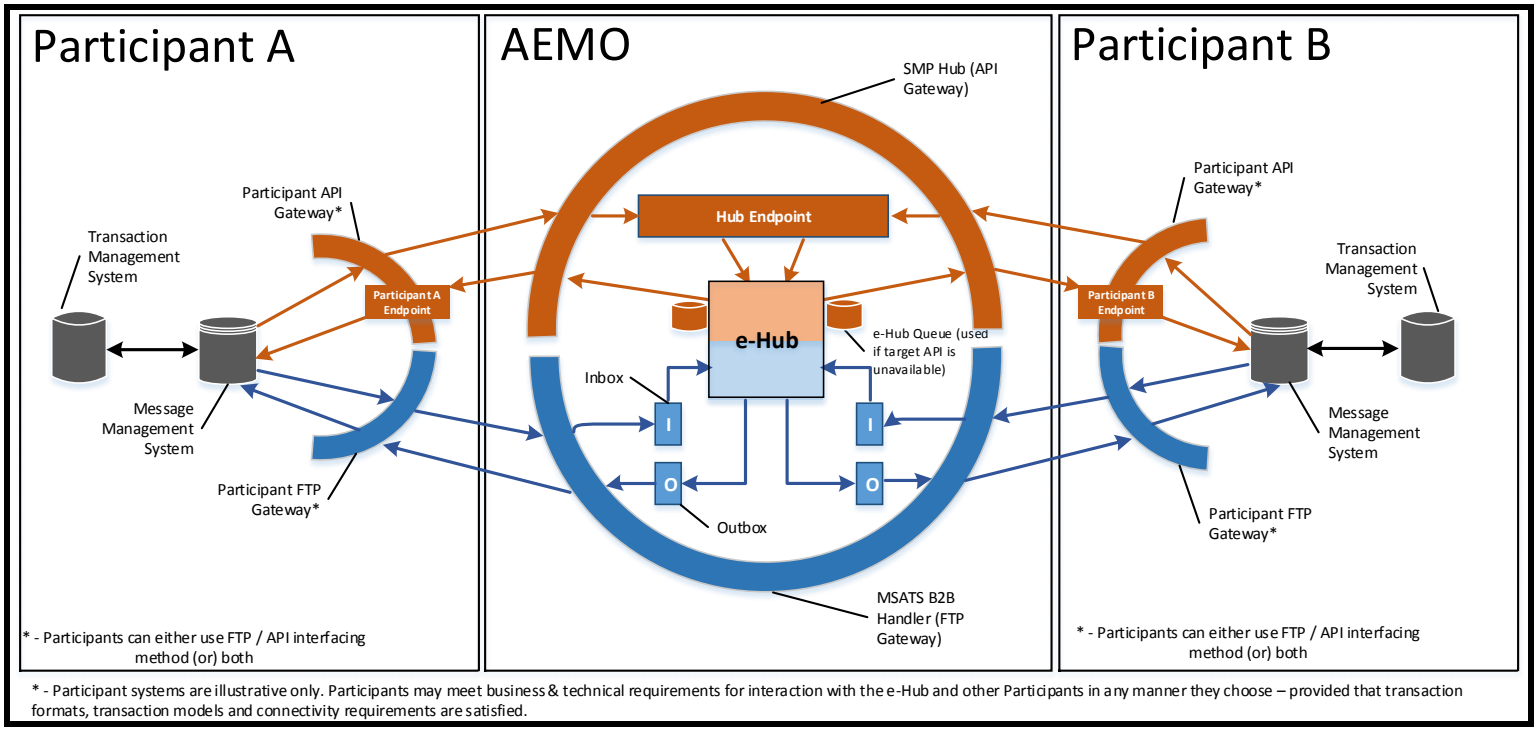
© AEMO 2017

2 E-HUB INFRASTRUCTURE

The e-Hub is AEMO's provided B2B communication platform supporting Electricity Retail transactions. It includes the SMP platform (providing an API gateway and portal to support B2B and value add web services) and the MSATS B2B Handlers (providing B2B FTP support).

This diagram illustrates the AEMO and participant components (hardware and software) of the National B2B Infrastructure, relating to the centralised e-Hub.

Figure 1 – High-level e-Hub infrastructure

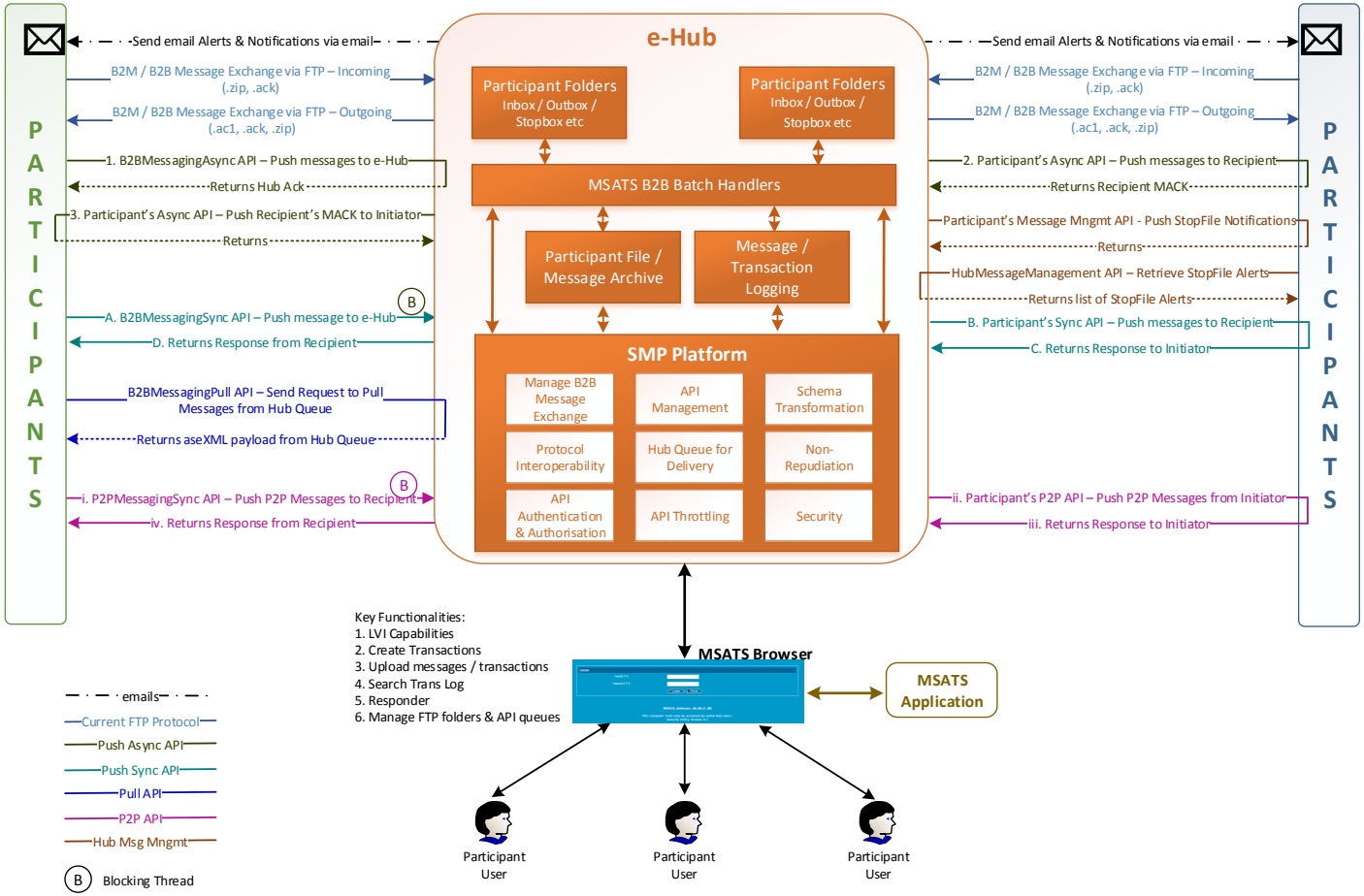


© AEMO 2017

2.1 e-Hub services

This diagram illustrates the summary of e-Hub service offerings.

Figure 2 Context Diagram



2.2 Accessing the e-Hub services

AEMO provisions access to participants or intending participants in accordance with the accreditation process outlined in the B2B e-Hub Accreditation Process.

AEMO provides the following pre-production and production environment e-Hub services:

1. The MSATS Browser functionality accessible over MarketNet.
2. The FTP Gateway accessible over MarketNet (to support B2B hokey-pokey protocol).
3. The e-Hub Portal functionality accessible over MarketNet or the internet.
4. The e-Hub API functionality accessible over internet or MarketNet. Either to push the messages or pull the message from the e-Hub queue using RESTful APIs.

The address details for the e-Hub systems are available in **Appendix B**.

MarketNet is AEMO's private data network connection. For details about the options available to connect to MarketNet, see Guide to Information Systems on AEMO's website.

All participants require MSATS web portal access, at a minimum for managing configuration related to the B2B services. For more detail, see e-Hub Configuration on page 56.

2.2.1 MSATS browser

For details about connecting to the MSATS Browser, see Connecting to AEMO's Electricity IT Systems on AEMO's website.

2.2.2 FTP Gateway

For details about connecting to the FTP Gateway, Connecting to AEMO's Electricity IT Systems on AEMO's website.

2.2.3 e-Hub Portal

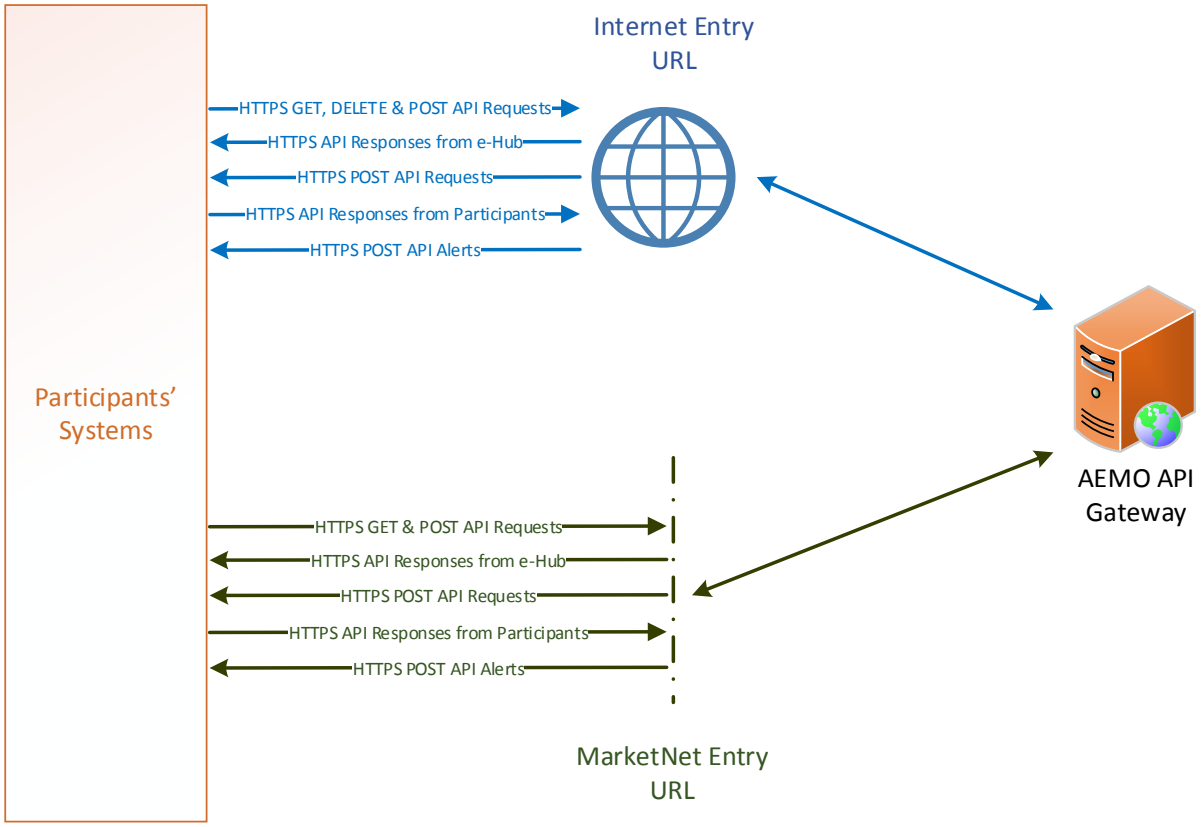
The e-Hub Portal is accessible over MarketNet or the internet. It provides details about AEMO's published APIs and managing API Keys. For details about managing API keys, see API Keys on page 10.

© AEMO 2017

2.2.4 e-Hub API Gateway

The e-Hub API Gateway is accessible over the internet via MarketNet as illustrated in the diagram below.

Figure 3 – API Connection Diagram



2.2.5 System requirements to connect to e-Hub API Gateway

To connect to the e-Hub API Gateway, a participant requires:

- Access either via MarketNet or the internet.
- SSL authentication using digital certificates.
- An API Key provided by AEMO.
- Their IP address range white listed by AEMO.

For the e-Hub API Gateway to connect to a participant's API, a participant is required to configure:

- SSL authentication using digital certificates.
- An API Key provided by the participant (optional).

AEMO provisions the SSL certificates and API Key as part of the accreditation process. For more detail, see SECURITY on page 8.

The SSL certificates and API Key to access pre-production is different from those used to access production.

3 SECURITY

3.1 Custom Ports

The e-Hub API Gateway uses custom ports as a defence against simple Denial of Service (DoS) attacks:

- For HTTPS the port is 9319 (Participant connecting to the e-Hub APIs).
- Participants may nominate any port between 9318 and 9330 in the URLs they supply to the SMP administration. These URLs apply to the participant gateways.

3.2 SSL

The e-HUB is configured for SSL connectivity in compliance with the TLS v1.1 and v1.2 protocols.

3.3 SSL Certificates

All communications between the e-HUB and a participant's gateway are carried out using HTTPS. HTTP is not supported on the e-HUB. TLS/SSL encryption is managed using public/private key pairs, with a different key pair required to connect to each environment (pre-production/production). Each participant must create / obtain a private key and a Certificate Signing Request (CSR).

A private key and CSR is usually created at the same time, making a key pair. A CSR is usually generated on the server where the certificate will be installed and contains information that will be included in the certificate such as the organisation name, common name (CN), locality and country. It also contains the public key that will be included in the certificate.

The process for obtaining certificates is provided in **Appendix C**.

© AEMO 2017

4 E-HUB APIS

AEMO's e-Hub APIs take advantage of existing messaging standards such as aseXML, maintaining maximum flexibility and consistency for participants. The market systems standard is the transfer of aseXML documents between participant gateways and the market systems.

The e-Hub APIs are designed to support the following messaging patterns:

1. **Push/Push** - A participants API gateway receives and sends using asynchronous and/or synchronous communication. For more detail, see Push-Push pattern on page 17.
2. **Push/Pull** - A participants develops a system using asynchronous communication with messages being queued in the e-Hub. For more detail, see Push-Pull pattern on page 17.

The e-Hub APIs allow:

1. Interoperability with participants using FTP (only for the asynchronous APIs).
2. Receiving and sending the B2B transactions between participants.
3. Receiving and sending Peer-2-Peer free form messages and/or attachments.
4. Receiving alerts and notifications from the e-Hub.

4.1 RESTful Architecture

AEMO chose RESTful (REST) for its web services architecture because of its lightweight nature and ability to transmit data directly over HTTPS. REST is an alternative to SOAP and WSDL. The REST architecture makes it possible to start small, developing what is required with available resources, and scaling up as the number of services increase.

The REST approach uses the features of HTTP to make requests and follows these design principles:

1. Services are provided using HTTPS protocol over MarketNet/Internet.
2. Directory structure-like URIs for example, https://<web service host>/<system>/<business_function>.
3. Transfer of variety of payloads such as XML (current solution) and JavaScript Object Notation (JSON) in future

Resources are addressed by mapping to a location within a hierarchy of URIs. For example, the root of the hierarchy might represent the web service/API application and provide a listing of the resources available. Drilling down one level then provides specific information about a particular resource, and further levels provide data from specific resource records.

AEMO's goal in implementing a RESTful web services approach is to achieve the following:

1. Performance: quality of responsiveness.
2. Scalability: many users can simultaneously use the systems.
3. Generality: solve a wide variety of problems.

© AEMO 2017

4. Simplicity: no complex interactions, easy to prove the system is doing as it is supposed to.
5. Modifiability: extensible in the face of new requirements and technologies.

4.2 API Authentication & authorisation

SSL certificates are used for securing the transport layer such as, encrypted communication and secure interactions between participant systems and AEMO's systems. SSL certificates are also used to authenticate the initiating participant ID i.e. the participant ID associated with the certificate will be validated against the participant ID initiating the API request.

4.2.1 API Keys

The initiating participant is required to obtain an API key. The API Key is used to authorise the participant (by participant ID) requesting the API services.

The process for obtaining an API Key is provided in **Appendix D**.

API Keys are supplied for each combination of API and participant ID. If an organisation owns multiple participant IDs, an API key needs to be sourced for each combination of API and participant ID as illustrated in the table below:

Table 2 – Illustration of how API Keys are provisioned

Organisation: Organisation 1 (e.g. Retailer)		
ParticipantID	**API Name**	**API Key**
ParticipantID 1	API 1	abcd1234
ParticipantID 2	API 1	1234abcd
ParticipantID 1	API 2	123456789
ParticipantID 2	API 2	abcdefghij

4.2.2 API Key Example

A participant has access to B2BMessagingAsync (Push-Push pattern) and has an API key linked to this API. If the participant attempts to invoke the B2BMessagingPull API service using the API key for B2BMessagingAsync the invocation is unauthorised and the e-Hub denies access to this API.

4.3 API format

API URLs are in the following format:

https://<web service host>/<business_function>/<APIversion>/<resource>?querystring parameters

For example:

https://apis.prod.aemo.com.au:9319/ws/B2BMessagingAsync/1.0/messages (Internet URL Entry)
https://apis.prod.marketnet.net.au:9319/ws/B2BMessagingAsync/1.0/messages (MarketNet URL Entry)

Table 3 – API Definition

Parameter	Description
<protocol>	HTTPS
<web service host>	Names the server hosting the service or an external proxy • Internet web service host: apis.prod.aemo.com.au:9319 • MarketNet web service host: apis.prod.marketnet.net.au:9319
<Business_function>	API Name - The AEMO system providing the services e.g. B2BMessagingAsync Allowed values of API Names • HubMessageManagement • B2BMessagingAsync • B2BMessagingPull • B2BMessagingSync • P2PMessagingSync
<Resource>	Entities of a Business Function e.g. /messages Allowed values are specific to each API. Refer section 6 for details.
?querystring parameters	Query string parameters for GET method Note: Query string parameters will be passed in the URL and is applicable only to the GET method.

4.3.1 Methods

Table 4 – API Methods

HTTPS Method	Operation
GET	Retrieve Data
POST	Post Data
DELETE	Delete Data

© AEMO 2017

4.3.2 HTTP request

Notes:

- URL query string parameters & HTTP header attributes are case sensitive.
- HTTP request using GET / DELETE method has:
    a. Query string parameters in the URL.
    b. HTTP request header parameters. API Key will be passed in the HTTP request header.
    c. 'Content-Type' should not be passed in the HTTP request when using GET / DELETE method.
- HTTP request using POST method will have:
    a. No query string parameters in the URL.
    b. HTTP request header parameters. API Key will be passed in the HTTP request header.
    c. Other HTTP request header parameters that are specific to the API.
    d. Payload (for example, aseXML payload).
    e. Optional attachments (multi-part) for P2P APIs.

The HTTP Header consists of:

1. x-eHub-APIKey - AEMO supplied API key.
2. Content-type: application/xml - Format of the attached request file (not applicable to GET and DELETE methods).
3. Accept: application/xml - Request the response as an XML file.
4. Content-Length: nnn - Length of the requested message.
5. Other header parameters that are very specific to the API, /resource & method (for help, see API SETTINGS on page 17).

© AEMO 2017

4.3.2.1 HTTP Request example 1

In this example, the bold content is the HTTP request header parameter specific to APIs.

```
POST /ws/B2BMessagingAsync/1.0/messages HTTP/1.1
Content-Type: application/xml
Accept: application/xml
Content-Length: nnn
x-eHub-APIKey: cfb529e0-b91f-11e6-bf5e-a7414e7415c0
messageContextID: sordm_retailer_abcd1234 (for example, required as a
header parameter for B2BMessagingAsync API)

<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<ase:aseXML xmlns:ase="urn:aseXML:r32"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:aseXML:r32
http://www.nemmco.com.au/aseXML/schemas/r32/aseXML_r32.xsd">
<Header>
<From description="XXXX">Initiator</From>
   <To description="YYYY">Recipient</To>
   <MessageID>ABC_792867346</MessageID>
   <MessageDate>2017-03-02T01:02:25.710+10:00</MessageDate>
   <TransactionGroup>SORD</TransactionGroup>
   <Priority>Medium</Priority>
   <Market>NEM</Market>
</Header>
<Transactions>
   <Transaction initiatingTransactionID="159984331740" transactionDate="2017-
03-02T01:02:25.000+10:00" transactionID="792883623">
   <ServiceOrderResponse responseType="Closure" version="r17">
            <ServiceOrder>
                     <NMI checksum="0">1111111111</NMI>
                     <ServiceOrderNumber>1657143384</ServiceOrderNumber>
            <ServiceProviderReference>792883624</ServiceProviderReference>
                     </ServiceOrder>
<NotificationData xsi:type="ase:ElectricityServiceOrderNotificationData">
                     <SpecialNotes>
                              <CommentLine>AMI RWD meter</CommentLine>
                     </SpecialNotes>
                     <ServiceOrderStatus>Completed</ServiceOrderStatus>
<ActualDateTime>2017-03-01T00:35:44.000+10:00</ActualDateTime>
                     <Product>
<Code>No Charge</Code>
                     </Product>
            </NotificationData>
   </ServiceOrderResponse>
   </Transaction>
</Transactions>
</ase:aseXML>
```

HTTP Request example 2

The bold content is the query string parameters.

```
GET /ws/B2BMessagingAsync/1.0/queues?initiatingParticipantID='NEMMCO'
```

4.3.3 HTTP response

The HTTP Response has:

1. HTTP response code & description
    a. A successful request to the web services server is indicated by the response code of 200 OK.
    b. Appropriate HTTP response codes for technical / payload validation failures.

       For help, see VALIDATIONS on page 99.
    c. Optional HTTP response header parameters (for example, messageContextID).
    d. Optional payload (for example, aseXML payload).

4.3.3.1 HTTP Response Example 1 (200 OK)

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: nnn
<?xml version="1.0" encoding="ISO-8859-1"?>
<ase:aseXML xmlns:ase="urn:aseXML:r32"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:aseXML:r32
http://www.nemmco.com.au/aseXML/schemas/r32/aseXML_r32.xsd">
<Header>
<From>NEMMCO</From>
   <To>INITIATOR</To>
   <MessageID>B2BM1485736773061000</MessageID>
   <MessageDate>2017-01-30T10:39:33.0+10:00</MessageDate>
   <TransactionGroup>SITE</TransactionGroup>
   <Priority>Medium</Priority>
   <Market>NEM</Market>
</Header>
<Acknowledgements>
<MessageAcknowledgement duplicate="No"
initiatingMessageID="B2BRT20170130102804" receiptDate="2017-01-
30T10:39:33.0+10:00" receiptID="B2BR1485736757430001" status="Reject">
   <Event class="Message" severity="Error">
            <Code description="New request with previously used
RetServiceOrder.">1914</Code>
   </Event>
   </MessageAcknowledgement>
</Acknowledgements>
</ase:aseXML>
```

4.3.3.2 HTTP Response Example 2 (HTTP Response Code 404)

```
HTTP/1.1 405 Method Not Allowed
Content-Length: nnn
Date: Mon, 01 May 2017 18:00:00 GMT
Connection: close
```

© AEMO 2017

5 VALIDATION OVERVIEW

The e-Hub validates the incoming API requests or API responses. The validations are categorised as:

1. Technical validations
2. Payload validations

Technical validations relate to:

1. Connectivity (for example, SSL authentication).
2. API Key (for example, missing or invalid API keys).
3. Throttling limits

Payload validations are related to:

1. Payload (for example, validation of the aseXML payload)
2. Query string parameters (for example, missing / invalid query string parameters in GET method)
3. HTTP request / response header parameters (for example, missing / invalid HTTP request / response header parameters)

The validation matrix documented in VALIDATIONS on page 99 provides a detailed view of technical and payload validations the e-Hub performs.

6 API SETTINGS

The e-Hub supports the following API message patterns.

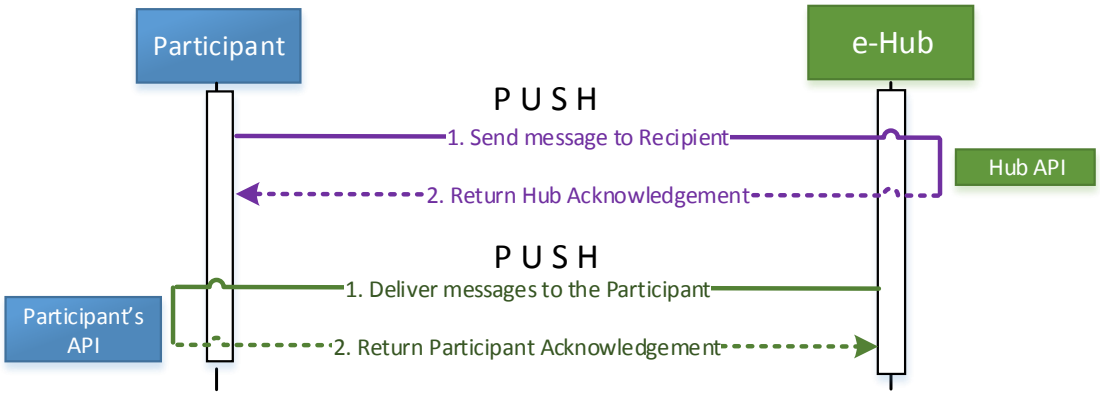
6.1 Push-Push pattern

The Push-Push pattern requires API implementation at:

1. The e-Hub (API Gateway and API Portal).
2. The Participant API Gateway.

This diagram below describes the Push-Push pattern.

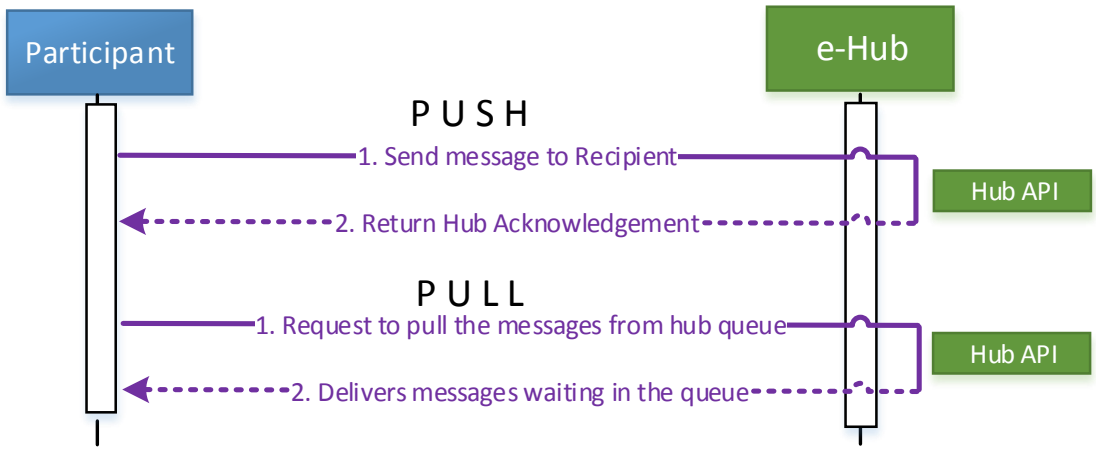
Figure 4 – Push-Push message pattern



6.2 Push-Pull pattern

The Push-Pull pattern requires API implementation only at the e-Hub. The diagram below describes the Push-Pull pattern.

Figure 5 – Push-Pull message pattern



The table below describes the APIs available for participants to interface with the e-Hub:

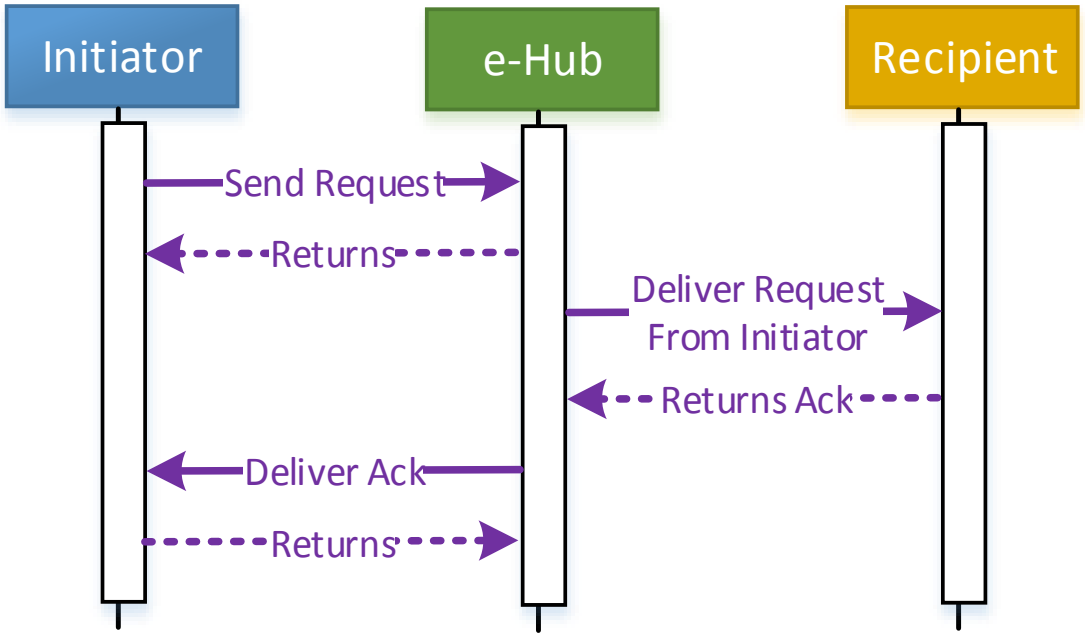
Table 5 – List of e-Hub APIs

API Name	Message Pattern
B2BMessagingAsync	Push-Push
B2BMessagingSync	Push-Push
B2BMessagingPull	Push-Pull
P2PMessagingSync	Push-Push
HubMessageManagement	Push from the e-Hub

6.2.1 B2BMessagingAsync

The Initiator 'pushes' their outgoing message to the e-Hub, and the e-Hub 'pushes' the message to the Recipient. The message delivery between the Initiator and the Recipient occurs in an asynchronous manner such as: initiator sends the message to the e-Hub and expects the outcome (such as MACKs, TACKs, responses) in a different API call.

Figure 6 – B2BMessagingAsync API – Conceptual View

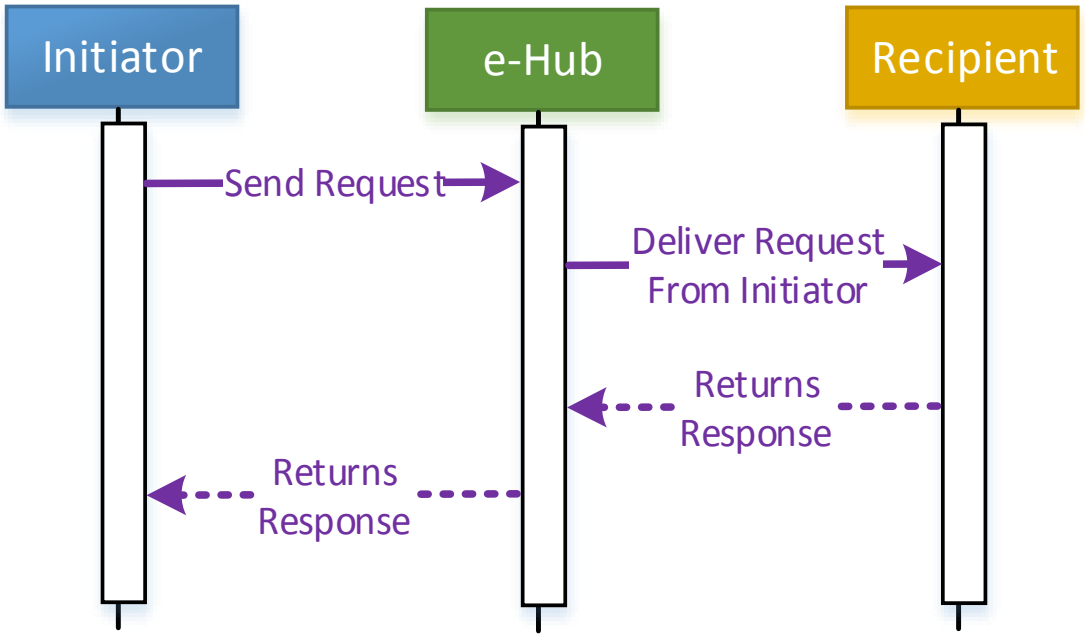


6.2.2 B2BMessagingSync

The participant 'pushes' their outgoing message to the e-Hub, and the e-Hub 'pushes' the message to the Recipient. The message delivery between the Initiator and the Recipient occurs in a synchronous manner or a blocking thread such as:

initiator sends the message to the e-Hub and expects the outcome (such as MACKs, TACKs, responses) in the same blocking thread.

Figure 7 – B2BMessagingSync API – Conceptual View



6.2.3 B2BMessagingPull

When this is used, the participant polls and 'pulls' messages from the e-Hub; electing this messaging pattern means that the participant is not required to implement APIs at their end.

6.2.4 P2PMessagingSync (Peer-to-Peer)

This API is used where agreed between participants to send transactions with attachments such as PDFs or images.

6.2.5 HubMessageManagement

This API is used where participants can retrieve the list of alerts such as B2B stop file alerts from the e-Hub.

Business transactions are sent as aseXML document carried as payloads inside the API's message and transmitted over HTTPS.

6.3 Common header parameters

6.3.1 messageContextID

The messageContextID is used in the following ways:

1. To provide a contextID for the message exchange. The participant/e-Hub uses the contextID of the original request when delivering its corresponding MACK(s).
2. If the Recipient is on FTP, the filename is set to <messageContextID>.zip.

3. The name of the archive files is set to <messageContextID>.zip.
4. To provide context to the failure messages to be returned where the incoming payload is unreadable such as, the payload is schema invalid.

The format is:

```
[TransactionGroup 0-9_a-z]{1,4} + [Priority h|m|l] + "_" +
[FromParticipantID]{1,10} + "_" + [0-9_a-z]{1,18}
```

For example:

```
sordl_retailer1_abcd1234.
```

messageContextID is case sensitive and required in lower case.

The e-Hub does not validate the uniqueness of messageContextID. Participants are required to ensure the messageContextID is unique when a new request is sent to the e-Hub.

For NotifiedParty transactions generated by the e-Hub the messageContextID is derived by the e-Hub, and <FromParticipantID> (in the aseXML file) is set to the ParticipantID of the Initiator (of the Service Order).

6.3.2 x-eHub-APIKey

When invoking e-Hub API resources, this parameter is the AEMO-supplied API Key. Note that an API Key is issued for each combination of API and ParticipantID.

API Keys are sourced from the AEMO API Portal. Initial API Keys are allocated as part of the registration and accreditation process.

6.4 HubMessageManagement

6.4.1 e-Hub implementation

This e-Hub interface is for use by *Market Participants* (participants). Participants utilise the HubMessageManagement API to retrieve the current list of stop files for all participants or for a specific participant (using alerts) resource.

The HubMessageManagement API also implements an additional resource ‘/ping’.

Participants can use this resource to:

1. Ensure technical requirements required to connect to e-Hub are validated (for example, appropriate network ports are open). Participants can utilise this function to ensure their systems can connect to e-Hub using their SSL certificates and API keys.
2. Determine if the e-Hub system is operational (ping-pong test).

Table 6 – e-Hub API Details – HubMessageManagment

Target Consumers	Market Participants
BASE URL – Internet Entry	https://apis.prod.aemo.com.au:9319/ws/
BASE URL – MarketNet Entry	https://apis.prod.marketnet.net.au:9319/ws/
API URL	https://apis.prod.aemo.com.au:9319/ws/HubMessageManagement/1.0/ https://apis.prod.marketnet.net.au:9319/ws/HubMessageManagement/1.0/

Example	GET https://apis.prod.aemo.com.au:9319/ws/HubMessageManagement/1.0/alerts?initiatingParticipantID='XYZ'
Accept	application/xml
Content-Length	nnn
API-Key	x-eHub-APIKey: <AEMO supplied>

The following resources are implemented by AEMO for the HubMessageManagement API.

Table 7 – e-Hub API Details – HubMessageManagment Resource & Methods

Resource	Supported Methods
alerts	GET
ping	GET

6.4.1.1 Alerts

Table 8 – e-Hub HubMessageManagement Resource: alerts

Resource	**alerts**
Method	**GET**
Description	Return Participants that have a B2B Stop file. Returns stop file details for all participants or the requested ParticipantID. Note: This API & resource will be enhanced in future to return the list of B2M stop files
Request	*Query String parameters* • initiating ParticipantID (mandatory) • queryParticipantID (optional) • alertType (optional) – default is 'B2BStopFile' if not supplied *HTTP Header Parameters* • x-eHub-APIKey
Response	• HTTP Response code & description *HTTP Header Parameters* • None *Payload* • aseXML Transaction payload – TransactionType: HubFlowControlReport. Refer to section 11 for details.

Query String & Request Header Parameters

Table 9 – e-Hub HubMessageManagement (/alerts): Query String & Request Header Parameters

Parameter	Values	Description	Optional/ Mandatory
initiatingParticipantID	Participant ID per MSATS	ParticipantID of the Participant invoking the API Note: If the API keys and certificates issued against the ParticipantID does not match with the	M

Parameter	Values	Description	Optional/ Mandatory
		ParticipantID passed in the query string, the system will send an exception	
alertType	B2BStopFile	If the value is set to 'B2BStopFile', details related to B2B stop files will be sent. In future this parameter will be extended to support the retrieval of B2M stop files. If this parameter is not passed; the e-Hub will default the alertType to 'B2BStopFile'.	O
queryParticipantID	ParticipantID per MSATS	If queryParticipantID is populated, e-Hub will retrieve the stop file for the queryParticipantID. The e-Hub will not validate if the queryParticipantID is valid. If an invalid queryParticipantID is sent, the e-Hub will send <ResultCount> = 0; meaning no stop file exists for the requested queryParticipantID. If queryParticipantID is not passed it will return the current list of stop files in the market.	O
x-eHub-APIKey	AEMO-supplied	AEMO-supplied API key.	M

HTTP Response

Table 10 – e-Hub HubMessageManagement (/alerts): Response parameters

Parameter	Technical Validations Pass?	Query String / Request Header Parameter Validations Pass?	Value	Examples/Remarks
HTTP Response Code	N	NA	Appropriate HTTP Response Code	e.g. 404 when Resource Name is invalid 405 when Method is invalid
	Y	Y	200 OK	aseXML Transaction payload TransactionType: HubFlowControlReport
	Y	N	500; description stating the exception	No Payload

6.4.1.2 Ping

Table 11 – e-Hub HubMessageManagement Resource: ping

Resource	ping
Method	GET
Description	Returns response code to state if the core e-Hub applications are available / operational. This API/resource will be used by the Participants to • Test if the Participant application / gateway is able to establish connectivity with the e-Hub and/or

	• Validate if the SSL configuration / certificates are valid and/or • Check if the core e-Hub applications are available
Request	*Query String parameters* • initiatingParticipantID *HTTP Header Parameters* • x-eHub-APIKey
Response	• HTTP Response code & description *HTTP Header Parameters* • None *Payload* • None

Query String & Request Header Parameters

Table 12 – e-Hub HubMessageManagement (/ping): Request parameters

Parameter	Values	Description	Optional/ Mandatory
initiatingParticipantID	Participant ID per MSATS	ParticipantID of the Participant invoking the API Note: If the API keys and certificates issued against the ParticipantID does not match with the ParticipantID passed in the query string, the system will send an exception	M
x-eHub-APIKey	AEMO-supplied	AEMO-supplied API key.	M

HTTP Response

Table 13 – e-Hub HubMessageManagement (/ping): Response parameters

Parameter	Scenario	HTTP Response Code & Description
HTTP Response Code	Connection Timeout	Standard connection time out exception
	Read Timeout (no response from e-Hub)	408; with appropriate description
	Any other technical issues related to connecting to the e-Hub	Appropriate HTTP or standard exception codes
	Technical validation failures (refer validation matrix)	Appropriate HTTP response code (refer validation matrix)
	Successful Ping i.e. Participant is able to connect to e-Hub, authentication & authorisation is valid and the e-Hub is operational	200 pong
	Core e-Hub application(s) is unavailable	500 ping failure

6.4.2 Participant implementation

Participants must implemented this interface.

Participants are required to define their URL and their API names. The e-Hub only registers the API name of the participant. By default, it uses the resources and methods mentioned in this section to push the messages to participants.

The e-Hub sends alerts to participants in the following scenarios:

1. A new B2B stop file is issued for any participant.
2. An existing B2B stop file is removed for any participant.
3. There are exceptions when processing the response payload for example, if a participant sends MACK payload as a response to the e-Hub initiated request and the MACK payload is schema invalid, the e-Hub utilises the '/alerts' resource implemented by the participant to notify the response payload validation failure.

Table 14 – Participant API Details

Target Consumers	AEMO e-Hub
API URL	As provided by Participants
Example	POST <Participant-provided URL>/alerts
Content Type	application/xml
Accept	application/xml
Content-Length	Nnn
API-Key (Optional)	Participant-provided name-value pair

Participants are required to implement the following resources and methods on their APIs.

Table 15 – Message Management - Participant API Details – Resources & Methods

Resource	Supported Methods
alerts	POST

6.4.2.1 Alerts

Table 16 – Participant Resources: alerts

Resource	**alerts**
Method	**POST**
Description	The e-Hub will send alerts to the Participants in the following scenarios • New B2B stop file is issued for any Participant (or) • An existing B2B stop file is removed from any Participant (or) • Notifying response payload validation failures
Request	*Query String parameters* • None *HTTP Header Parameters*

	• API Key if Participants require e-Hub to send their API Key – Optional *Payload* • aseXML schema, TransactionType: HubFlowControlAlertNotification or PayloadExceptionAlert. Refer to section 11 for details.
Response	• HTTP response code & description

Request Header Parameters

Table 17 – Participant HubMessageManagement: Request parameters

Parameter	Values	Description	Optional/ Mandatory
<API Key Name supplied by Participants>	As supplied by Participants	As supplied by Participants	O

Response sent by participants to the e-Hub

Table 18 – Participant HubMessageManagement: Response parameters

Parameter	Technical Validations Pass?	Payload Validations Pass?	Value	Examples/Remarks
HTTP Response Code	N	NA	Appropriate HTTP Response Code	e.g. 404 when Resource Name is invalid 405 when Method is invalid
	Y	Y	200 OK	No Payload.
	Y	N	500; description stating the exception	No Payload.

6.5 B2BMessagingAsync

B2B messages are exchanged between the participants in asynchronous fashion for example, separate API calls are made to transfer the message between the Initiator and Recipient.

The B2BMessagingAsync features include:

1. Participants call the B2BMessagingAsync API to push the messages to the Recipients
2. AEMO calls the participant's registered Async API to push the messages received from other Initiators
3. This pattern requires API implementation at the e-Hub and by participants:
   a. For faster delivery (speed) of messages.
   b. To suit exchange of high volumes of messages.

Timing requirements for the delivery of transactions and acknowledgements for the B2B Messaging Async services are captured in section 5.9, Table 10 (Timing Requirements (webservices only)) of the B2B Procedure Technical Delivery Specification.

6.5.1 e-Hub implementation

This interface is implemented by the e-Hub for use by *Market Participants.*

Table 19 – B2BMessagingAsync API - e-Hub API Details

Target Consumers	Market Participants
BASE URL – Internet Entry	https://apis.prod.aemo.com.au:9319/ws/
BASE URL – MarketNet Entry	https://apis.prod.marketnet.net.au:9319/ws/
API URL	https://apis.prod.aemo.com.au:9319/ws/B2BMessagingAsync/1.0/ https://apis.prod.marketnet.net.au:9319/ws/B2BMessagingAsync/1.0/
Example	POST https://apis.prod.aemo.com.au:9319/ws/B2BMessagingAsync/1.0/messages
Content Type	application/xml (not applicable to GET Method)
Accept	application/xml
Content-Length	nnn
API-Key	x-eHub-APIKey: <AEMO supplied>

The following resources are implemented by AEMO for the B2BMessagingAsync API.

Table 20 – B2BMessagingAsync API - e-Hub API Details – Resources & Methods

Resource	Supported Methods
messages	POST
messageAcknowledgements	POST
queues	GET

6.5.1.1 Messages

The e-Hub '/messages' resource is defined as follows:

Table 21 – e-Hub Resources: messages

Resource	**messages**
Method	**POST**
Description	• e-Hub receives a B2B aseXML message payload (i.e. payload contains transaction or transaction acknowledgement message) from the Initiator. Refer 9.1.1 for definitions of transaction / transaction acknowledgement message • e-Hub responds with a hub acknowledgement to the Initiator (Message Acknowledgement or event only payload) • e-Hub pushes the message to the intended Recipient
Request	*Query String parameters* • None *HTTP Header Parameters* • messageContextID – Mandatory • x-eHub-APIKey – Mandatory *Payload – Mandatory*

	• aseXML Transaction Message, or • aseXML Transaction Acknowledgement Message Note: Only one aseXML message payload is allowed in the HTTP request and multi-part messages are not permitted
Response	• HTTP response code and description *HTTP Header Parameters* • None *Payload – Mandatory (if technical validations pass)* • asexML Message Acknowledgment message (this could be an Event-Only MACK). Refer 9.1.1 for definitions of message acknowledgement message

e-Hub '/messages' resource Request Header Parameters are as follows:

Table 22 – e-Hub B2BMessagingAsync API (/messages): Request header parameters

Parameter	Values	Description	Optional/ Mandatory
messageContextID	Participant defined	Refer to section 6.1 for details.	M
x-eHub-APIKey	AEMO-supplied	AEMO-supplied API key.	M

e-Hub '/messages' resource Response Header Parameters are as follows:

Table 23 – e-Hub B2BMessagingAsync API (/messages): Response header parameters

Parameter	Technical Validations Pass?	Payload/ Recipient availability Validations Pass?	Value	Examples/Remarks
HTTP Response Code	N	NA	Appropriate HTTP Response Code	404 when Resource Name is invalid 405 when Method is invalid
	Y	Y	200 OK	aseXML hub Acknowledgement (status = 'Accept') will be sent
	Y	N	200 OK	aseXML hub Acknowledgement (NACK) will be sent. The <Event> element will state the exception details.

6.5.1.2 Message Acknowledgements

Table 24 – e-Hub Resources: messageAcknowledgements

Resource	**messageAcknowledgements**
Method	**POST**
Description	This resource will be used when Participants are required to send Message Acknowledgement in the HTTP request. messageContextID should be set to the messageContextID of the original message that is to be acknowledged.

	Note: If a Participant is required to send a transaction message or transaction acknowledgement message, '/messages' resource is to be used. If a Participant is required to send 'message acknowledgements' in the request payload, then '/messageAcknowledgements' resource is to be used.
Request	*Query String parameters* • None *HTTP Header Parameters* • messageContextID • x-eHub-APIKey *Payload* • aseXML Message Acknowledgement Message (includes event only MACKs) Note: Only one aseXML message payload is permitted in the HTTP request
Response	HTTP response code & description *HTTP Header Parameters* • None *Payload* • None

Request Header parameters

Table 25 – e-Hub B2BMessagingAsync API: Request parameters for /messageAcknowledgements

Parameter	Values	Description	Optional/ Mandatory
messageContextID	Participant defined	This should be set to the messageContextID of the original message that is to be acknowledged. Refer section 6.1.1 for details	M
x-eHub-APIKey	AEMO-supplied	AEMO-supplied API key	M

Response sent by the e-Hub to the requested participant

Table 26 – e-Hub B2BMessagingAsync API: Response parameters for /messageAcknowledgements

Parameter	Technical Validations Pass?	Payload/ Recipient availability Validations Pass?	Value	Examples/Remarks
HTTP Response Code	N	NA	Appropriate HTTP Response Code	404 when Resource Name is invalid 405 when Method is invalid
	Y	Y	200 OK	No Payload
	Y	N	500; description stating the exception	No Payload

6.5.1.3 Queues

Table 27 – e-Hub Resources: queues

Resource	queues
Method	**GET**
Description	Returns meta-data of messages in e-Hub queue for the participant initiating the API request.
Request	*Query String parameters* • initiatingParticipantID – Mandatory *HTTPS Header Parameters* • x-eHub-APIKey
Response	• HTTPS response code & description *HTTPS Header Parameters* • None *Payload (if technical validations pass)* • aseXML Transaction payload – TransactionType: HubQueueReport. Refer section 11 for details.

Query String / Request Header Parameters

Table 28 – e-Hub B2BMessagingAsync API: Request parameters & query string parameters for /queues

Parameter	Values	Description	Optional/ Mandatory
initiatingParticipantID	Participant ID per MSATS	ParticipantID of the Participant invoking the API	M
x-eHub-APIKey	AEMO-supplied	AEMO-supplied API key.	M

Response sent by the e-Hub to the requested participant

Table 29 – e-Hub B2BMessagingAsync API: Response parameters for /queues

Parameter	Technical Validations Pass?	Query String Parameter Validations Pass?	Value	Examples/Remarks
HTTP Response Code	N	NA	Appropriate HTTP Response Code	e.g. 404 when Resource Name is invalid 405 when Method is invalid
	Y	Y	200 OK	aseXML message listing the messages & its meta data in the e-Hub queue.
	Y	N	500; description stating the exception	No Payload.

6.5.2 Participant implementation

This interface must be implemented by participants.

Participants are required to define their URL and the API names. The e-Hub only registers the API name of the participant. By default, it uses the resources and methods illustrated in this section to push the messages to participants.

Participants are required to implement the following resources and methods on their APIs to accept the messages / message acknowledgements from the e-Hub.

Table 30 – Participant API details

Target Consumers	AEMO e-Hub
API URL	As provided by Participants
Example	POST <Participant-provided URL>/messages
Content Type	application/xml
Accept	application/xml
Content-Length	Nnn
API-Key (Optional)	Participant-provided name-value pair

The following resources are implemented by participants:

Table 31 – Participant Async API Details – Resources & Methods

Resource	Supported Methods
messages	POST
messageAcknowledgments	POST

6.5.2.1 Messages

Table 32 – Participant Resources: messages

Resource	**messages**
Method	**POST**
Description	• Receive B2B aseXML payload • Respond with Participant MACK to requesting participant
Request	*Query String parameters* • None *HTTP Header Parameters* • messageContextID – Mandatory • API Key if Participants require e-Hub to send their API Key – Optional *Payload – Mandatory* • aseXML Transaction Message, or • aseXML Transaction Acknowledgement Message Note: Only one aseXML message payload is permitted in the HTTP request
Response	• HTTP response code & description *HTTP Header Parameters*

	• None *Payload – Mandatory (if technical validations pass)* • asexML Message Acknowledgment message, or • Event-Only MACK

Request Header Parameters

Table 33 – Participant Async API: Request header parameters for /messages

Parameter	Values	Description	Optional/ Mandatory
messageContextID	Participant defined	The e-Hub will pass the Initiator's messageContextID. Refer to section 6.3.1 for details.	M
Participant Supplied API Key	Participant supplied	Provided by the e-Hub if Participant has supplied an API Key	O

Response sent by the participant to the e-Hub

Table 34 – Participant Async API: Response parameters for /messages

Parameter	Technical Validations Pass?	Payload Validations Pass?	Value	Examples/Remarks
HTTP Response Code	N	NA	Appropriate HTTP Response Code	404 when Resource Name is invalid 405 when Method is invalid
	Y	Y	200 OK	Participant is required to send the MACK payload (status = 'Accept')
	Y	N	200 OK	Participant is required to send the MACK payload (NACK)

6.5.2.2 Message Acknowledgements

Table 35 – Participant Resources: messageAcknowledgements

Resource	**messageAcknowledgements**
Method	**POST**
Description	This resource will be used when the e-Hub is required to send messageAcknowledgement (or event-only MACK) to a Participant.
Request	*Query String parameters* • None *HTTP Header Parameters* • messageContextID • API Key if Participants require e-Hub to send their API Key – Optional *Payload*

	• aseXML Message Acknowledgement message (including event-only MACKs) Note: Only one aseXML message payload is permitted in the HTTP request
Response	• HTTP response code & description *HTTP Header Parameters* • None *Payload* • None

Request Header Parameters

Table 36 – Participant Async API: Request header parameters for /messageAcknowledgements

Parameter	Values	Description	Optional/ Mandatory
messageContextID	Participant to derive	e-Hub will set this attribute to the messageContextID of the original message that is being acknowledged. Refer to section 6.3.1 for details.	M
Participant Supplied API Key	Participant supplied	Provided by the e-Hub if Participant requires an API Key.	O

Response sent by the participant

Table 37 – Participant Async API: Response parameters for /messageAcknowlegements

Parameter	Technical Validations Pass?	Payload/ Recipient availability Validations Pass?	Value	Examples/Remarks
HTTP Response Code	N	NA	Appropriate HTTP Response Code	404 when Resource Name is invalid 405 when Method is invalid
	Y	Y	200 OK	No Payload
	Y	N	500; description stating the exception	No Payload

6.6 B2BMessagingSync

Messages are delivered between participants in a blocking thread for example, the Initiator sends the request to the Recipient and the Recipient is expected to send the response (MACK Reject / TACK Reject / Response) in the same blocking thread.

The following rules apply:

1. The Initiator and the Recipient need to have bilateral agreement related to the use of B2BMessagingSync services and have this service enabled.
2. B2BMessagingSync API will not support queuing or interoperability.

© AEMO 2017

3. Refer section 6.7 for details related to 'Connection & Read Timeout Settings' for Sync services.

6.6.1 E-Hub implementation

This interface is implemented by the e-Hub for use by *Market Participants.*

Table 38 – e-Hub API Details – B2BMessagingSync

Target Consumers	Market Participants
BASE URL – Internet Entry	https://apis.prod.aemo.com.au:9319/ws/
BASE URL – MarketNet Entry	https://apis.prod.marketnet.net.au:9319/ws/
API URL	https://apis.prod.aemo.com.au:9319/ws/B2BMessagingSync/1.0/ https://apis.prod.marketnet.net.au:9319/ws/B2BMessagingSync/1.0/
Example	POST https://apis.prod.aemo.com.au:9319/ws/B2BMessagingSync/1.0/messages
Content Type	application/xml
Accept	application/xml
Content-Length	Nnn
API-Key	x-eHub-APIKey: <AEMO supplied>

The following resources are implemented by AEMO for the B2BMessagingSync API.

Table 39 – e-Hub API Details – B2BMessagingSync – Resources & Methods

Resource	Supported Methods
messages	POST

6.6.1.1 Messages

E-Hub messages resource is defined as follows:

Table 40 – e-Hub Resources: /messages

Resource	**Messages**
Method	**POST**

Description	• Receive B2B aseXML payload
	• If validations of the incoming request pass
	  ○ Block the Initiator's request thread
	  ○ Send the message to Recipient
	  ○ Send the Recipient's response to Initiator as response to the blocking thread
	• If validations of the incoming request fail
	  ○ Respond to the blocking thread with Hub acknowledgement stating the exception

Request	*Query String parameters* • None *HTTP Header Parameters*

	• messageContextID – Mandatory • x-eHub-APIKey – Mandatory *Payload – Mandatory* aseXML Transaction Message Note: Only one aseXML message payload is permitted in the HTTP request
Response	• HTTP Response Code *HTTP Header Parameters* • None *Payload (If there are no technical validation failures when the message is sent to the e-Hub)* • asexML Message Acknowledgment Message, or • Event-Only MACK, or • aseXML Transaction Acknowledgement Message, or • aseXML Transaction Message

Request Header Parameters

Table 41 – e-Hub B2BMessagingSync API: Request header parameters for /messages

Parameter	Values	Description	Optional/ Mandatory
messageContextID	Participant defined	Refer to section 6.3.1 for details.	M
x-eHub-APIKey	AEMO-supplied	AEMO-supplied API key.	M

HTTP Response

Table 42 – e-Hub B2BMessagingSync API: Response parameters for /messages

Parameter	Technical Validations Pass?	Payload/ Recipient availability Validations Pass?	Value	Examples/Remarks
HTTP Response Code	N	NA	Appropriate HTTP Response Code	404 when Resource Name is invalid 405 when Method is invalid
	Y	Y	200 OK	Recipient's aseXML MACK / TACK / response payload
	Y	N	200 OK	aseXML hub Acknowledgement (NACK). The <Event> element will state the details about the exception.

6.6.2 Participant implementation

This interface must be implemented by participants if opting-in for Synchronous services.

Participants are required to define their URL and the API names. The e-Hub only registers the API name of the participant. By default, it uses the resources and methods illustrated in this section to push the messages to participants. Participants are required to implement the following resources and methods on their APIs to accept the messages from the e-Hub or other participants.

Table 43 – Participant API Details

Target Consumers	AEMO e-Hub
API URL	As provided by Participants
Example	POST <Participant-provided URL>/messages
Content Type	application/xml
Accept	application/xml
Content-Length	Nnn
API-Key (Optional)	Participant-provided name-value pair

The following resources are implemented by participants:

Table 44 – Participant API Details – Sync API – Resources & Methods

Resource	Supported Methods
messages	POST

6.6.2.1 Messages

Table 45 – Participant Resources: messages

Resource	**messages**
Method	**POST**
Description	• Receive B2B aseXML payload • Respond with Participant MACK (status=Reject) when aseXML schema is invalid (or) TACK (status=Accept / Partial / Reject) (or) Response aseXML payload
Request	*Query String parameters* • None *HTTP Header Parameters* • messageContextID – Mandatory • API Key if Participants require e-Hub to send their API Key – Optional *Payload – Mandatory* • aseXML Transaction Message Note: Only one aseXML message payload is allowed in the HTTP request
Response	• HTTP response code & description *HTTP Header Parameters* • None *Payload (if there are no technical validation failures when the message is sent to the e-Hub or when the e-Hub is calling the Recipient's API)* • asexML Message Acknowledgment Message (status='Reject'), or

	• Event-Only MACK, or • aseXML Transaction Acknowledgement Message (status='Accept' / 'Partial' 'Reject') or • aseXML Transaction Message (response)

Request Header parameters (e-Hub sends to participants)

Table 46 – Participant Sync API: Request header parameters for /messages

Parameter	Values	Description	Optional/ Mandatory
messageContextID	Participant defined	The e-Hub will pass the messageContextID that the Initiator sent to the e-Hub. Refer to section 6.3.1 for details.	M
Participant Supplied API Key	Participant supplied	Provided by the e-Hub if Participant requires an API Key	O

Response sent by the participant to the e-Hub

Table 47 – Participant Sync API: Response parameters for /messages

Parameter	Technical Validations Pass?	Payload/ Recipient availability Validations Pass?	Value	Examples/Remarks
HTTP Response Code	N	NA	Appropriate HTTP Response Code	404 when Resource Name is invalid 405 when Method is invalid
	Y	Y	200 OK	aseXML MACK / TACK / Response payload
	Y	N	200 OK	Participant is required to send the MACK payload (NACK)

6.7 B2BMessagingPull

If messages are sent to a participant opted-in for the Pull messaging pattern, the messages will be queued in the e-Hub. The receiving participant can poll the e-Hub, 'pull' and process the messages that are queued.

If a participant opts-in for the Pull messaging pattern, the participant always uses the e-Hub APIs for both sending and receiving messages. The participants opting-in for the Pull API are not required to implement APIs at their end.

Participants opting-in for the Pull messaging pattern are responsible for implementing polling logic to poll their e-Hub queue for retrieving new messages (similar to batch programs used to poll their Outbox using FTP).

Participants can either opt-in for Push or Pull messaging pattern (mutually exclusive). This is a single setting applying across all transaction groups/types, for example: a participant cannot choose to use the Push messaging pattern for Service Orders, and Pull messaging pattern for Customer Details Notifications.

© AEMO 2017

Features of B2BMessagingPull:

1. This pattern does not require participants to implement APIs (for e-Hub to call).
2. Participants are required to implement the polling algorithm i.e. poll the e-Hub to retrieve (pull) the messages from the e-Hub at regular intervals; speed of message delivery to the Recipient is slower when compared to B2BMessagingAsync API.
3. Suits exchange of low volume of messages.

Timing requirements for the delivery of transactions and acknowledgements for the B2B Messaging Pull services are captured in the Section 5.9, Table 9 of the B2B Procedure Technical Delivery Specification. Timing requirements related to 'FTP to FTP' scenario applies to this API.

6.7.1 E-Hub implementation

This interface is implemented by the e-Hub for use by *Market Participants*.

Table 48 – e-Hub API Details (Pull)

Target Consumers	Market Participants
BASE URL – Internet Entry	https://apis.prod.aemo.com.au:9319/ws/
BASE URL – MarketNet Entry	https://apis.prod.marketnet.net.au:9319/ws/
API URL	https://apis.prod.aemo.com.au:9319/ws/B2BMessagingPull/1.0/ https://apis.prod.marketnet.net.au:9319/ws/B2BMessagingPull/1.0/
Example	POST https://apis.prod.aemo.com.au:9319/ws/B2BMessagingPull/1.0/messages
Content Type	application/xml (not applicable to GET / DELETE Method)
Accept	application/xml
Content-Length	nnn
API-Key	x-eHub-APIKey: <AEMO supplied>

The following resources are implemented by AEMO for the B2BMessagingPull API.

Table 49 – e-Hub API Details (Pull) – Resources & Methods

Resource	Supported Methods
messages	POST
messageAcknowledgments	POST
messageAcknowledgments	DELETE
queues	GET

6.7.1.1 Messages

The e-Hub messages resource is defined as follows:

Table 50 – e-Hub Resources: messages (Pull)

Resource	**messages**
Method	**POST**

Description	• e-Hub receives B2B aseXML message payload (i.e. payload has transaction or transaction acknowledgement message) from the Initiator • e-Hub responds with a Hub acknowledgement to the Initiator (Message Acknowledgement or event only payload) • e-Hub delivers the message to the Recipient
Request	*Query String parameters* • None *HTTP Header Parameters* • messageContextID – Mandatory • x-eHub-APIKey – Mandatory *Payload – Mandatory* • aseXML Transaction Message, or • aseXML Transaction Acknowledgement Message Note: Only one aseXML message payload is permitted in the HTTP request
Response	• HTTP response code & description *HTTP Header Parameters* • None *Payload (If there are no technical validation failures when the message is sent to the e-Hub) (refer to validation matrix)* • asexML Message Acknowledgment message, or • Event-Only MACK.

Request header parameters

Table 51 – e-Hub B2BMessagingPull API: Request header parameters for /messages

Parameter	Values	Description	Optional/ Mandatory
messageContextID	Participant defined	Refer to section 6.1.1	M
x-eHub-APIKey	AEMO-supplied	AEMO-supplied API key.	M

Response header parameters

Table 52 – e-Hub B2BMessagingPull API: Response parameters for /messages

Parameter	Technical Validations Pass?	Payload Validations Pass?	Value	Examples/Remarks
HTTP Response Code	N	NA	Appropriate HTTP Response Code	e.g. 404 when Resource Name is invalid 405 when Method is invalid
	Y	Y	200 OK	aseXML hub acknowledgement (status = ‘Accept’)

Parameter	Technical Validations Pass?	Payload Validations Pass?	Value	Examples/Remarks
	Y	N	200 OK	aseXML hub Acknowledgement (NACK). The <Event> element will state the details of the exception.

6.7.1.2 Message acknowledgements

6.7.1.2.1 POST

Table 53 – e-Hub Resources: messageAcknowledgements (Pull) – POST

Resource	**messageAcknowledgements**
Method	**POST**
Description	This resource will be used when Participants are required to send messageAcknowledgement in the HTTP request. messageContextID should be set to the messageContextID of the original message that is to be acknowledged. Note: If a Participant is required to send a transaction message or transaction acknowledgement message, '/messages' resource is to be used. If a Participant is required to send 'message acknowledgements' in the request payload, then 'messageAcknowledgements' resource is to be used.
Request	*Query String parameters* • None *HTTP Header Parameters* • messageContextID • x-eHub-APIKey *Payload* • aseXML Message Acknowledgement message (including event-only MACK) Note: Only one aseXML message payload is allowed in the HTTP request
Response	• HTTP response code & description *HTTP Header Parameters* • None *Payload* • None

Request Header Parameters

Table 54 – e-Hub B2BMessagingPull API: Request parameters for /messageAcknowledgements

Parameter	Values	Description	Optional/ Mandatory
messageContextID	Participant defined	This should be set to the messageContextID of the original message that is being acknowledged. Refer to section 6.3.1 for details.	M
x-eHub-APIKey	AEMO-supplied	AEMO-supplied API key.	M

Response sent by the e-Hub to the requested participant

Table 55 – e-Hub B2BMessagingPull API: Response parameters for /messageAcknowledgements

Parameter	Technical Validations Pass?	Payload Validations Pass?	Value	Examples/Remarks
HTTP Response Code	N	NA	Appropriate HTTP Response Code	e.g. 404 when Resource Name is invalid 405 when Method is invalid
	Y	Y	200 OK	No Payload
	Y	N	500; description stating the exception	No Payload

6.7.1.2.2 DELETE

Table 56 – e-Hub Resources: messageAcknowledgements (Pull) – DELETE

Resource	**messageAcknowledgements**
Method	**DELETE**
Description	Participants opting-in for Pull API are required to notify the e-Hub that the message acknowledgement was successfully received and processed. This notification will delete the message acknowledgement message in the e-Hub queue.
Request	*Query String parameters* • initiatingParticipantID - Mandatory • messageContextID - Mandatory *HTTP Header Parameters* • x-eHub-APIKey *Payload* • None
Response	• HTTP response code & description *HTTP Header Parameters* • None *Payload* • None

Query string / request header parameters

Table 57 – e-Hub B2BMessagingPull API: Request & query string parameters for /messageAcknowledgements (DEL)

Parameter	Values	Description	Optional/ Mandatory
initiatingParticipantID	Participant ID per MSATS	ParticipantID of the Participant invoking the API	M

Parameter	Values	Description	Optional/ Mandatory
messageContext ID (query string parameter)	Participant to populate	This should be set to the messageContextID of the message acknowledgement (MACK) that is being requested to be deleted from the e-Hub queue. Refer to section 6.3.1 for details.	M
x-eHub-APIKey	AEMO-supplied	AEMO-supplied API key.	M

Response sent by the e-Hub to the requested participant

Table 58 – e-Hub B2BMessagingPull API: Response parameters for /messageAcknowledgements (DEL)

Parameter	Technical Validations Pass?	Payload Validations Pass?	Value	Examples/Remarks
HTTP Response Code	N	NA	Appropriate HTTP Response Code	e.g. 404 when Resource Name is invalid 405 when Method is invalid
	Y	Y	200 OK	No payload
	Y	N	500; description stating the exception	No payload

6.7.1.3 Queue

Table 59 – e-Hub Resources: queues (Pull)

Resource	**queues**
Method	**GET**
Description	Returns the meta-data of messages in the e-Hub queue (or) the aseXML messages queued awaiting delivery to the Recipient (initiator of this API).
Request	*Query String parameters* • initiatingParticipantID – Mandatory • messageContextID – Optional • transactionGroup – Optional • priority – Optional • maxResults – Optional *HTTP Header Parameters* • x-eHub-APIKey
Response	**If maxResults is populated in the Request:** • HTTP response code & description *HTTP Header Parameters* • messageContextID of the message being delivered *Payload (if technical validations pass)*

	• aseXML payload received from the Initiator **If maxResults is not populated in the Request:** • HTTP response code & description *HTTP Header Parameters* • None *Payload (if technical validations pass)* • aseXML Transaction payload (meta-data only), TransactionType: HubQueueReport. Refer to section 11 for details.

Query string / request header parameters

Table 60 – e-Hub B2BMessagingPull API: Request & query string parameters for /queues

Parameter	Values	Description	Optional/ Mandatory
initiatingParticipantID	Participant ID per MSATS	ParticipantID of the Participant invoking the API	M
maxResults	1	The '/queues' resource could be used in two ways • To identify the meta data of messages (list of messages) in the e-Hub queue (or) • To pull the messages from e-Hub queue If maxResults query string parameter is not passed, then the e-Hub will provide a list of messages in the e-Hub queue for the initiating Participant. If maxResults is populated; the e-Hub will return messages in the e-Hub queue, providing the oldest message in the e-Hub queue (FIFO) if other parameters are not passed. The message that was pulled from the e-Hub queue will be deleted only when the Participant has successfully responded with a corresponding MACK. Note: If maxResults > 1; the e-Hub will default it to 1; i.e. the e-Hub will always retrieve only one message from the Hub queue. maxResults > 1 will be utilised in the future where Participants can specify the number of aseXML messages to pull from the e-Hub queue.	O
messageContextID	Participants to populate	If maxResults and messageContextID are passed, the API will attempt to retrieve (pull) the message matching the messageContextID. If only messageContextID is passed, the API will retrieve the meta data of the message(s) matching the messageContextID. Note: If an invalid messageContextID is passed, the e-Hub will send HTTP response code of 404 stating that the requested resource does not exist.	O
transactionGroup	Participants to populate	Valid values are: MTRD, MRSR, SORD, CUST, SITE, OWNP, OWNX, NPNX, PTPE If maxResults & transactionGroup are populated API will pull the message(s) matching the	O

Parameter	Values	Description	Optional/ Mandatory
		transactionGroup; providing the oldest message in the e-Hub queue (FIFO) If only transactionGroup is passed (no maxResults) API will retrieve the meta data of the message(s) matching the transactionGroup Note: If an invalid transactionGroup is passed, the e-Hub will send HTTP response code of 500 stating that transaction group is invalid	
Priority	Participants to populate	Valid values are: Low, Medium or High If maxResults & priority are populated API will pull the message(s) matching the priority; providing the oldest message in the e-Hub queue (FIFO) If only priority is passed (no maxResults) API will retrieve the meta data of message(s) matching the priority Note: If an invalid priority is passed, the e-Hub will send HTTP response code of 500 stating that priority is invalid	O
x-eHub-APIKey	AEMO-supplied	AEMO-supplied API key.	M

Response sent by the e-Hub to the requested participant

Table 61 – e-Hub B2BMessagingPull API: Response parameters for /queues

Parameter	Technical Validations Pass?	Query String Parameter Validations Pass?	Value	Examples/Remarks
HTTP Response Code	N	NA	Appropriate HTTP Response Code	e.g. 404 when Resource Name is invalid 405 when Method is invalid
	Y	Y	200 OK	If maxResults is passed, aseXML payload messageContextID of the payload If maxResults is not passed, aseXML payload containing the list of messages and its metadata in e-Hub queue.
	Y	N	500; description stating the exception	No Payload.

6.8 P2PMessagingSync

Participants can exchange the following Peer-to-Peer information via the e-Hub:

- Free-form information
- Documents (also called Attachments)

Participants can use the following capabilities to exchange free-form information and/or attachments:

Table 62 – P2P Offerings

Data Exchange	FTP	B2BMessaging Async	B2BMessaging Sync	B2BMessagingPull	P2PMessaging Sync
Free Form Text	✓	✓	✓	✓	✓
Free Form Text + Attachments	X	X	X	X	✓

Rules:

- P2PMessagingSync API does not support queuing or interoperability.
- The Initiator and the Recipient need to have bilateral agreement related to the use of P2PMessagingSync services and have this service enabled.
- Free form messaging using FTP, B2BMessagingAsync, B2BMessagingSync and B2BMessagingPull follow the same message patterns as described for other transaction groups.
- P2PMessagingSync API will support exchange of the following attachment types (configurable): pdf, csv, jpeg, gif, zip & txt. In future, this API supports additional attachment types.

6.8.1 E-Hub implementation

This interface is implemented by the e-Hub for use by *Market Participants*.

Table 63 – e-Hub API Details (P2P)

Target Consumers	Market Participants
BASE URL – Internet Entry	https://apis.prod.aemo.com.au:9319/ws/
BASE URL – MarketNet Entry	https://apis.prod.marketnet.net.au:9319/ws/
API URL	https://apis.prod.aemo.com.au:9319/ws/P2PMessagingSync/1.0/ https://apis.prod.marketnet.net.au:9319/ws/P2PMessagingSync/1.0/
Example	POST https://apis.prod.aemo.com.au:9319/ws/P2PMessagingSync/1.0/messages
Content Type	Refer to section 7 for details.
Accept	Refer to section 7 for details.
Content-Length	Refer to section 7 for details.
API-Key	x-eHub-APIKey: <AEMO supplied>

The following resources are implemented by AEMO for the P2PMessagingSync API.

Table 64 – e-Hub API Details (P2P) – Resources & Methods

Resource	Supported Methods
messages	POST

6.8.1.1 Messages

The e-Hub messages resource is defined as follows.

Table 65 – e-Hub Resources: messages (P2P)

Resource	**messages**
Method	**POST**
Description	Accepts and delivers one of the following in a synchronous or blocking thread: • Free-form text • Free-form text + documents / attachments
Request	*Query String parameters* • None *HTTP Header Parameters* • messageContextID – Mandatory • x-eHub-APIKey – Mandatory *Payload – Mandatory* • aseXML Transaction Message, or • aseXML Transaction Acknowledgement Message
Response	• HTTP response code & description *HTTP Header Parameters* • None *Payload – Mandatory (if technical validations pass)* • asexML Message Acknowledgment Message, or • Event-Only MACK, or • aseXML Transaction Acknowledgement Message

Request Header Parameters

Table 66 – e-Hub P2PMessagingSync API: Request header parameters for /messages

Parameter	Values	Description	Optional/ Mandatory
messageContextID	Participant defined	Refer to section 6.3.1 for details.	M
x-eHub-APIKey	AEMO-supplied	AEMO-supplied API key.	M

Response Header Parameters

Table 67 – e-Hub P2PMessagingSync API: Response parameters for /messages

Parameter	Technical Validations Pass?	Payload/ Recipient availability Validations Pass?	Value	Examples/Remarks
HTTP Response Code	N	NA	Appropriate HTTP Response Code	e.g. 404 when Resource Name is invalid 405 when Method is invalid
	Y	Y	200 OK	Recipient's aseXML MACK/TACK.
	Y	N	200 OK	aseXML hub Acknowledgement (NACK). The <Event> element will state the details of the exception.

6.8.2 Participant implementation

This interface must be implemented by participants if opting-in for P2P services.

Participants are required to define their URL and the API names. The e-Hub only registers the API name of the participant. By default, it uses the resources and methods mentioned in this section to push the messages to the participants. Participants are required to implement the following resources and methods on their APIs to accept the messages from the e-Hub or other participants.

Table 68 – Participant API Details (P2P)

Target Consumers	AEMO e-Hub
API URL	As provided by Participants
Example	POST <Participant-provided URL>/messages
Content Type	Refer to section 7 for details.
Accept	Refer to section 7 for details.
Content-Length	Refer to section 7 for details.
API-Key (Optional)	Participant-provided name-value pair

The following resources are implemented by participants.

Table 69 – Participant API Details (P2P) – Resources & Methods

Resource	Supported Methods
messages	POST

6.8.2.1 Messages

Table 70 – Participant Resources: messages (P2P)

Resource	**messages**
Method	**POST**
Description	Accepts one of the following in a synchronous / blocking thread: • Free-form text

	• Free-form text + documents / attachments
Request	*Query String parameters* • None *HTTP Header Parameters* • messageContextID – Mandatory • API Key if Participants require e-Hub to send their API Key – Optional *Payload – Mandatory* • aseXML Transaction Message, or • aseXML Transaction Acknowledgement Message
Response	• HTTP response code *HTTP Header Parameters* • None *Payload – Mandatory (if technical validations pass)* • asexML Message Acknowledgment Message, or • Event-Only MACK, or • aseXML Transaction Acknowledgement Message

Request header parameters

Table 71 – Participant P2P API: Request header parameters for /messages

Parameter	Values	Description	Optional/ Mandatory
messageContext ID	Participant defined	Refer to section 6.3.1 for details.	M
Participant Supplied API Key	Participant supplied	Provided by the e-Hub if Participant requires an API Key.	O

Response header parameters

Table 72 – Participant P2P API: Response parameters for /messages

Parameter	Technical Validations Pass?	Payload Validations Pass?	Value	Examples/Remarks
HTTP Response Code	N	NA	Appropriate HTTP Response Code	e.g. 404 when Resource Name is invalid 405 when Method is invalid
	Y	Y	200 OK	Participant is required to send the MACK payload or TACK payload.
	Y	N	200 OK	Participant is required to send the MACK payload (NACK).

6.9 Connection & read timeout settings

The connection & read timeout settings documented in this section are the e-Hub settings.

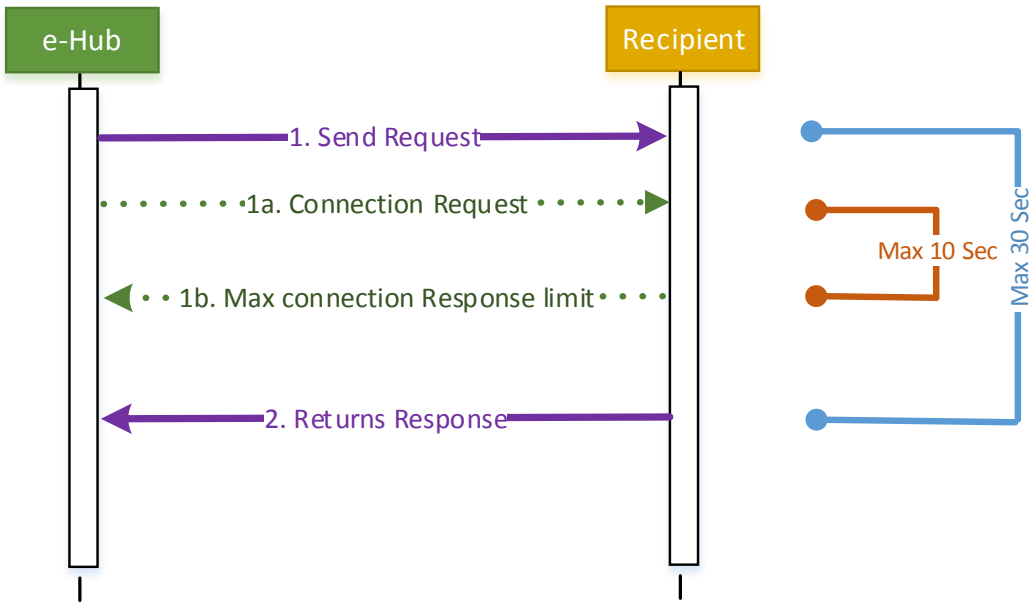
Definition:

- Connection timeout: e-Hub is unable to connect to the Recipient's endpoint
- Read timeout: the e-Hub is connected to Recipient's endpoint but the Recipient does not respond within the configured time

Participants are recommended to use the following settings when calling AEMO's e-Hub APIs:

1. The timeout values documented in this section are indicative; the values will be fine-tuned after the completion of performance testing.
2. Timeout parameters for Asynchronous services (i.e. HubMessageManagement API and B2BMessagingAsync API).
    a. The following diagram illustrates the timeout settings that e-Hub will implement when calling the participant's API in Asynchronous mode

Figure 8 – ASynchronous time out parameter setting



    b. If the e-Hub is unable to connect to the participant's end point within 10 seconds (configurable), the request will be timed out. The message will be queued in e-Hub queue and delivered using the scheduled delivery process. Refer section 9.1.2 for details.
    c. If the e-Hub is connected to the participant's end-point, the response is expected within (30 – (time taken to establish the connection)) seconds. If the response is not received within the expected timeframes, the HTTP request will be timed out (HTTP response code of 408). The message will be stored in the e-Hub queue for scheduled delivery (Refer section 9.1.2 for details).
    d. Worked examples
    e. If the connection is established after 5 seconds, the response is expected in 25 seconds

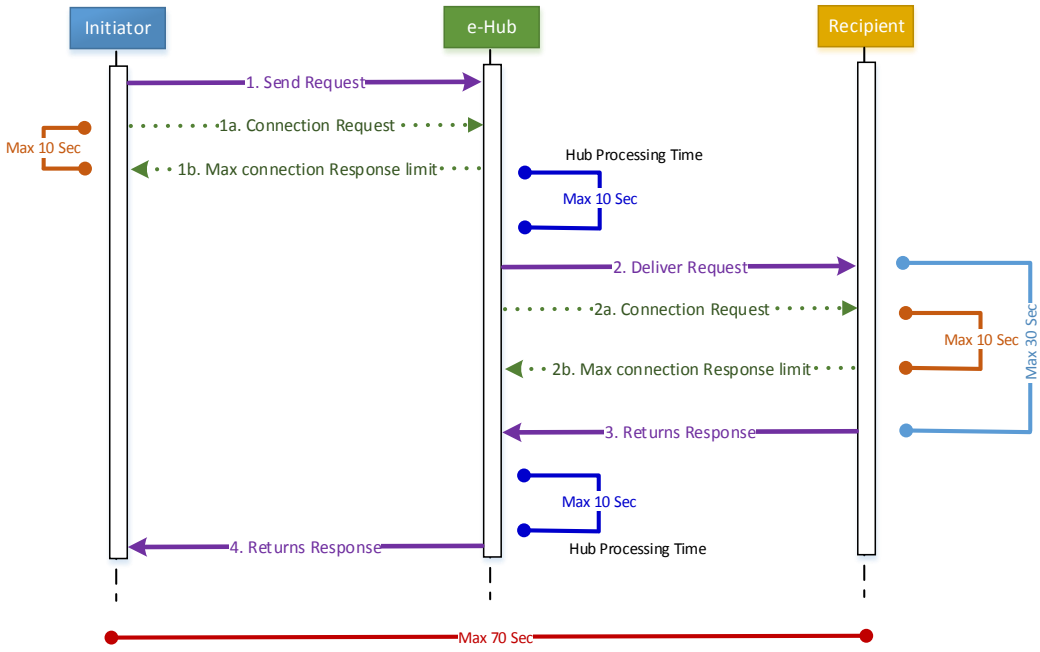
f. If the connection is established in a second, the HTTP response is expected in 29 seconds

g. If the participants encounter connection / read time outs when connecting to the e-Hub, participants are required to resubmit the message to the e-Hub

3. Timeout parameters for Synchronous services (i.e. P2PMessagingSync API and B2BMessagingSync API).

   a. The following diagram illustrates the timeout settings that e-Hub will implement when calling the participant's API in Synchronous mode. The participants are also recommended to use the following setting when implementing their APIs.

Figure 9 – Synchronous time out parameter setting



Refer section 9.4.7 – The scenario where the e-Hub HTTP request to the Recipient times out.

4. The above diagrams state that participants (Recipient) have 1-30 seconds to send the HTTP response. This is a worst-case scenario. Participants are requested to process and respond to the message within 10 seconds (after the connection is established) of receiving the message for Async scenario and 20 seconds of receiving the request for Sync scenario. The e-Hub has monitoring tools to internally report the participant's average response time.

5. Participants are recommended to set the timeout parameter (when calling the e-Hub's APIs) to 30 seconds (worst-case scenario) for Async services (including 10 seconds for connection timeout). The e-Hub attempts to process and respond to the incoming Async messages within 5 seconds of receiving the message.

6.10 Summary of API specifications

Participants can choose the protocol (API or FTP) at the transaction group level (via MSATS the browser screen).

If participants opt-in for APIs for all/few of the Transaction Groups, they are required to register for one of the following services:

1. Push-Push APIs (Asynchronous and Synchronous APIs).
2. Push-Pull APIs.
3. Leverage MSATS browser screens to manage the message exchange (using 'B2B Hub Queue' & 'Search Trans Log' screens).

If participants opt-in to use APIs for SORD & MTRD, they cannot use Push-Push APIs for SORD and Push-Pull APIs for MTRD. Participants must use one API pattern for managing messages related to Transaction Groups where protocol is set to 'API'.

The following table provides the summary of APIs that the e-Hub implement (participants are required to source API keys for these APIs using e-Hub's API Portal) and participants have to implement for each of the API patterns.

Table 73 – Summary of e-Hub & Participant API Implementation for each API Pattern

Participants opting in for Push-Push Pattern		Participants opting in for Push-Pull Pattern	
e-Hub API Keys to be sourced for the following APIs	**Implement the following APIs**	**e-Hub API Keys to be sourced for the following APIs**	**Implement the following APIs**
HubMessageManagement API – Mandatory	Hub Message Management API – Mandatory	HubMessageManagement API - Mandatory	None
B2BMessagingAsync API – Mandatory	B2B Messaging Async API – Mandatory	B2BMessagingPull API – Mandatory	
B2BMessagingSync API – Optional	B2B Messaging Sync API – Optional		
P2PMessagingSync API - Optional	P2P Messaging Sync API - Optional		

7 PEER-TO-PEER (P2P) MESSAGING PAYLOADS

The following rules apply:

- Free form information exchange is implemented via aseXML i.e. transaction type <PTPDataExchange> & element <FreeFormData> will support free form text exchange. Element <FreeFormData> is defined as “xsd:anyType”. Participants have to agree bilaterally on the format of the data sent via <FreeFormData> element.
- Transaction Group ‘PTPE’ will be used to exchange free form information and/or attachments between the participants
- Documents or attachments are exchanged using the HTTP protocol capabilities. HTTP supports multi-part/form data that will be leveraged to exchange documents / attachments. The list of attachments sent via HTTP request must be listed in the aseXML document i.e. transaction type <PTPDataExchange> & element <AttachmentList>. The e-Hub validates if the attachment names (list) in the HTTP request matches to the list of attachments documented in the aseXML payload. The e-Hub will deliver the message only when the attachment list in the aseXML and HTTP request matches.
- The e-Hub transforms aseXML messages if the Recipient has opted in for transformation (between current & supersede version). The data in the attachments is not altered / transformed by the e-Hub.
- P2PMessagingSync API supports exchange of the following attachment types (configurable): pdf, csv, jpeg, gif, zip & txt. In future, this API will support additional attachment types.

7.1 Exchange free-form text

The following settings exist:

Table 74 – HTTP Request call Setting – Exchange Free-Form Text

Example	POST https://apis.prod.aemo.com.au:9319/ws/P2PMessagingSync/1.0/messages
Content-Type	application/xml
Accept	application/xml
Content-Length	nnn
API-Key	x-eHub-APIKey: <AEMO supplied>

Example:

```
POST /ws/P2PMessagingSync/1.0/messages HTTP/1.1
Content-Type: application/xml
accept: application/xml
Content-Length: nnn
x-eHub-APIKey: cfb529e0-b91f-11e6-bf5e-a7414e7415c0
messageContextID: ptpem_retailer_abcd1234
<<<<<<<<<<<<<<<<<<aseXML Payload>>>>>>>>>>>>>>>>>>>>>
```

7.2 Exchange single attachment

The following settings exist:

Table 75 – HTTP Request call Setting – Exchange Single Attachment

Example	POST https://apis.prod.aemo.com.au:9319/ws/P2PMessagingSync/1.0/messages
Content-Type	multipart/form-data; boundary=P2PPayload
Accept	application/xml
Content-Length	Nnn
API-Key	x-eHub-APIKey: <AEMO supplied>

7.2.1 Settings for the aseXML Payload

Content-Type	application/xml

7.2.2 Settings for the attachment

Content-Disposition	form-Data; name=”files”
Filename	Name of the file e.g. Filename=’sord123.txt’
Content-Type	Content Type e.g. text/plain

© AEMO 2017

Example:

```
POST /ws/P2PMessaging/1.0/messages HTTP/1.1
Content-Type: multipart/form-data; boundary=P2PPayload
Accept: application/xml
Content-Length: nnn
x-eHub-APIKey: cfb529e0-b91f-11e6-bf5e-a7414e7415c0
messageContextID: ptpem_retailer_abcd1234

--P2PPayload
Content-Type: application/xml
<<<<<<<<<<<<<<<<<<aseXML Payload>>>>>>>>>>>>>>>>>>>>>

--P2PPayload
Content-Disposition: form-Data; name="files"; Filename='sord123.txt'
Content-Type: text/plain
<<<<<<<<<<<<<<<<<<Content of the
file>>>>>>>>>>>>>>>>>>>>>
--P2PPayload--
```

7.3 Exchanging multiple attachments

The following settings exist:

Table 76 – HTTP Request call Setting – Exchange Multiple Attachments

Example	POST https://apis.prod.aemo.com.au:9319/ws/P2PMessagingSync/1.0/messages
Content-Type	multipart/form-data; boundary=P2PPayload
Accept	application/xml
Content-Length	nnn
API-Key	x-eHub-APIKey: <AEMO supplied>

7.3.1 Settings for the aseXML Payload

Content-Type	application/xml

7.3.2 Settings for defining the boundary for multiple attachments

Content-Disposition	Form-Data; name="files"
Content-Type	multipart/form-data; boundary=P2PAttach

7.3.3 Settings for the attachment

Content-Disposition	form-Data; name="files"
Filename	Name of the file e.g. Filename='sord123.txt'
Content-Type	Content Type e.g. text/plain

Example:

```
POST /ws/P2PMessagingSync/1.0/messages HTTP/1.1
Content-Type: multipart/form-data; boundary=P2PPayload
Accept: application/xml
Content-Length: nnn
x-eHub-APIKey: cfb529e0-b91f-11e6-bf5e-a7414e7415c0
messageContextID: ptpem_retailer_abcd1234 (specific to
B2BMessaging API)
--P2PPayload
Content-Type: application/xml
<<<<<<<<<<<<<<<<<<aseXML Payload>>>>>>>>>>>>>>>>>>>>>

--P2PPayload
Content-Disposition: form-Data; name="files"
Content-Type: multipart/form-data; boundary=P2PAttach
--P2PAttach
Content-Disposition: file; Filename='sord123.txt'
Content-Type: text/plain
<<<<<<<<<<<<<<<<<<Content of the TXT
file>>>>>>>>>>>>>>>>>>>>>

--P2PAttach
Content-Disposition: file; Filename='sord123.gif'
Content-Type: image/gif
Content-Transfer-Encoding: binary
<<<<<<<<<<<<<<<<<<Content of the GIF
file>>>>>>>>>>>>>>>>>>>>>
--P2PAttach--
--P2PPayload—
```

Participants can follow their own naming standards when defining the boundary names. Participants are not required to use the boundary names as mentioned in this section (for example, 'P2PAttach' or 'P2PPayload').

7.4 aseXML PAYLOAD

A New Transaction Group PTPE – Peer-to-Peer-Exchange is introduced to support exchange of free-form data and/or attachments between the Participants.

7.4.1 PTPEDataExchange

Participants can exchange free-form data and/or attachments using the transaction type <PTPEDataExchange>. Element <FreeFromData> must be used to exchange free-form data. Exchange of attachments is supported only via P2PMessagingSync API; the list of attachments in the HTTP request must be listed in the <AttachmentList> element (aseXML). The e-Hub will validate to ensure that the list of attachments in the HTTP request matches to the list documented in <AttachmentList> element.

Table 77 – PTPEDataExchange

Field	Format	Use	Definition
FreeFormData	ANY	O	Participants must mutually agree on the format of the data. The e-Hub will not validate the content of this field. E.g. this field may carry CSV or JSON or XML data (etc.) as mutually agreed between the Participants. Since the field will be defined as 'xsd:anyType', the field 'FreeFormData' can also take attributes that are mutually agreed between the Participants as shown below <PTPDataExchange version="r36"> <FreeFormData format="unknown" contextID="test">1234567890</FreeFormData> </PTPDataExchange>
Type(Attachment type)	Varchar(30)	O	Type of the attachment. Must be one of the allowed attachment types. e.g. CSV, TXT
Name (Attachment name)	Varchar(260)	O	Name of the attachment. The name of the attachment must also include the file extension. e.g. test.csv

8 E-HUB CONFIGURATION

8.1 Participant-specific settings

8.1.1 Configuration via MSATS B2B browser

Refer to Guide to MSATS B2B for step-by-step instructions and on-boarding document for further details.

The following configurations related to B2B services are available in the MSATS B2B interface.

- Changing the protocol (FTP/API) setting by the Transaction Group

  Note: To change from FTP to API a participant must have been accredited for the relevant protocol, see B2B e-Hub Accreditation Process for more information.

  a. Each participant selects the protocol using the B2B browser. The protocol has to be selected for each B2B Transaction Group as shown in the table below.

Participant 1		
Transaction Group	**FTP**	**API**
SORD		✓
CUST	✓	
MTRD	✓	

Participant 2		
Transaction Group	**FTP**	**API**
SORD	✓	
CUST	✓	
MTRD	✓	

  b. Participants send and receive messages / files using the opted (selected) protocol for example, if the participant sends SORD messages using APIs, the participant receives SORD messages also via APIs.

  Note: Participants cannot receive SORDs via API & send SORDs via FTP and vice-versa.

  c. The e-Hub will validate each incoming message / file to determine if the participant is sending the message / file using the opted protocol. The e-Hub will trigger an exception if the participant sends a message / file using the protocol that is not opted (refer validation matrix for details)

Changing supported B2B schema versions by Transaction Group

Configuring the B2B/B2M alerts and email address for these alerts

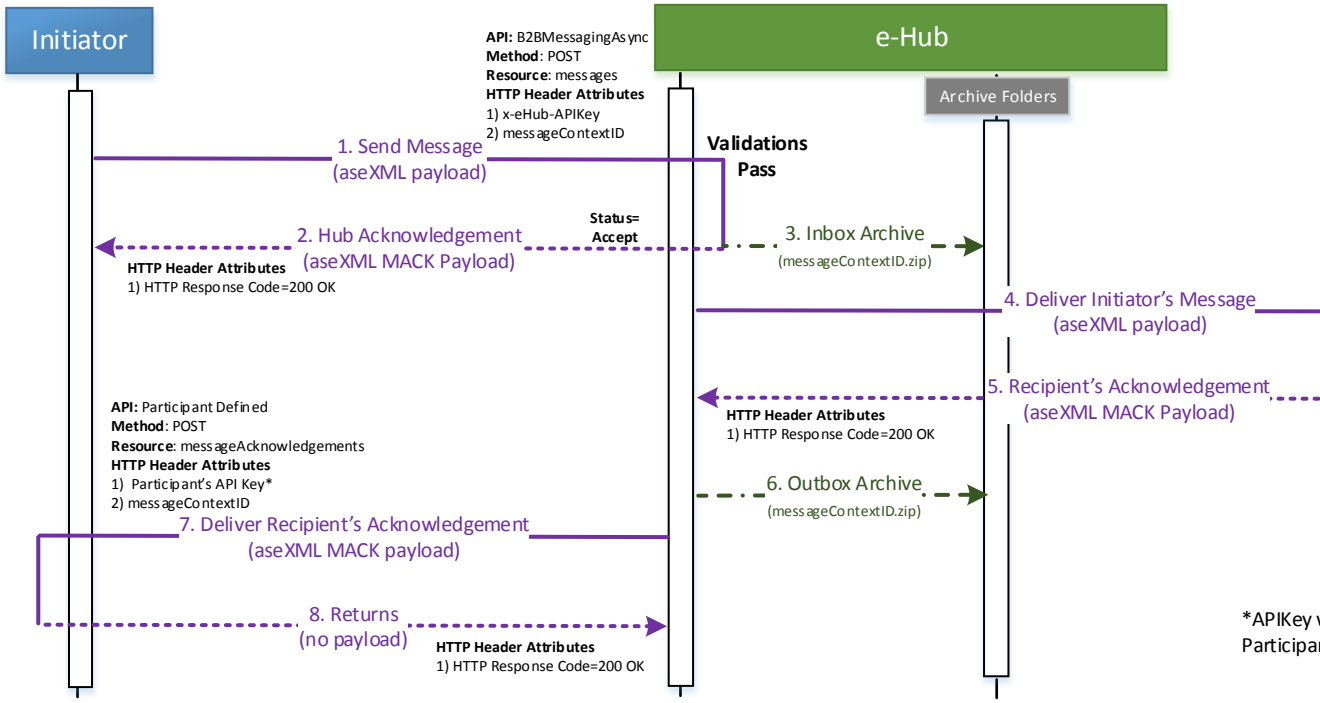
© AEMO 2017

9 SCENARIOS

9.1 B2B Asynchronous Message Exchange

9.1.1 Normal processing

Figure 10 2BMessagingAsync – Normal-processing Scenario



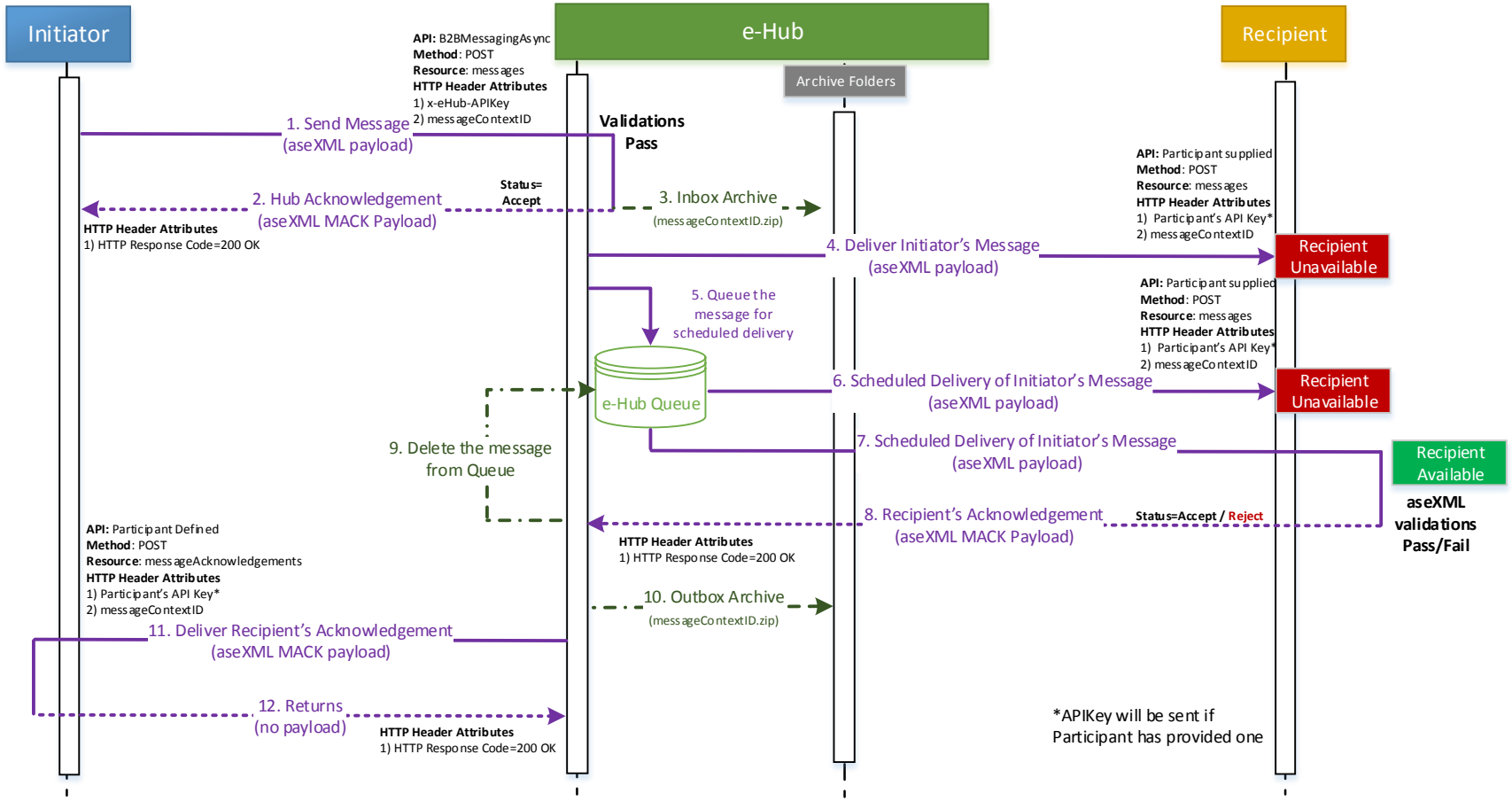
9.1.1.1 Summary

1. In the normal processing scenario (async) Figure 10 on page 57, the Recipient is available and has not opted for controlled throttling by the e-Hub. The message is delivered to the Recipient immediately.
2. The Initiator sends the following messages to the e-Hub using '/messages' resource:
    a. Transaction Messages – aseXML message containing <Transactions> element
    b. Transaction Acknowledgement Message – aseXML message containing <TransactionAcknowlegement> element but no <MessageAcknowledgement> element
3. The e-Hub validates the incoming message and sends the hub acknowledgement (aseXML MACK payload) as the HTTP response.
4. The incoming message is archived in the Initiator's inbox archive directory. Existing archival rules (including directory structure) are applicable. The messageContextID will be used as the archive file name i.e. messageContextID.zip.
5. The e-Hub delivers the message (message is transformed if required) to the Recipient. The e-Hub calls the Recipient's registered Async API and uses '/messages' resource. When the e-Hub calls a participant's API, an API Key will be passed if the participant has provided one to AEMO otherwise SSL authentication will provide security for the outbound call.
6. The Recipient validates the incoming message and issues the Recipient's acknowledgement as the response to the e-Hub API call.
7. The message that was sent to the Recipient is archived in the Recipient's outbox archive directory. Existing archival rules (including directory structure) are applicable. The messageContextID will be used as the archive file name i.e. messageContextID.zip. The message is archived only when the Recipient acknowledges the message.
8. The e-Hub sends the Recipient's acknowledgement (message is transformed if required) to the Initiator by calling the Initiator's registered Async API and uses the '/messageAcknowledgements' resource. The e-Hub will use 'messageAcknowledgements' resource if the payload is 'Message Acknowledgment Message':
    a. Message Acknowledgement Message – aseXML message containing at least one <MessageAcknowledgement> element (or) is an event-only MACK
9. The Initiator validates the Recipient's acknowledgement and accepts the payload.

© AEMO 2017

9.1.2 Normal processing (Queued)

Figure 11 – B2BMessagingAsync – Normal-processing Scenario (Queued)



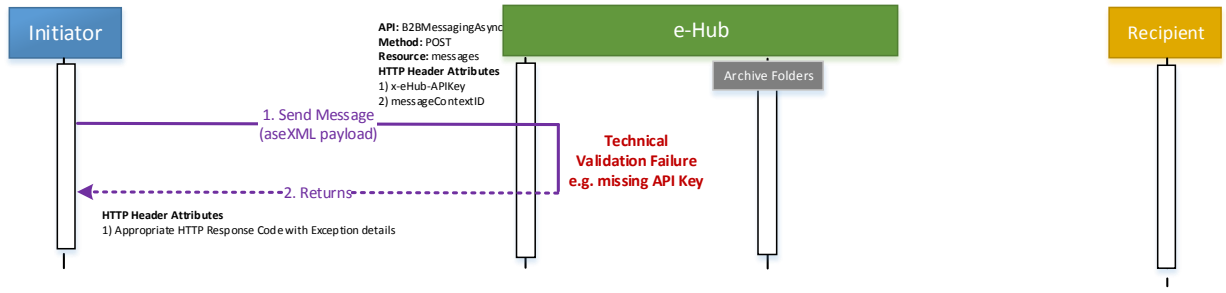
© AEMO 2017

9.1.2.1 Summary

1. When a Recipient is unavailable, the e-Hub stores the message in the e-Hub Queue and attempts to redeliver it periodically. The message is delivered when the Recipient's system becomes available.
2. The messages are also stored in e-Hub queue and delivered periodically if the Recipient (participant) has requested for controlled throttling.
3. The message is deleted from the e-Hub Queue after receiving the Recipient's acknowledgement.
4. The e-Hub will cease to deliver the messages to the Recipient after 10 days. The messages are available in the e-Hub queue (after 10 days). Participants can process such messages using MSATS B2B browser.
5. Messages in the e-Hub queue older than 15 days are purged. When the messages are purged from the e-Hub queue, the messages (does not apply to message acknowledgement messages) are not archived.

9.1.3 Technical validation failure (e-Hub)

Figure 12 – B2BMessagingAsync - Technical Validation Failure (e-Hub)

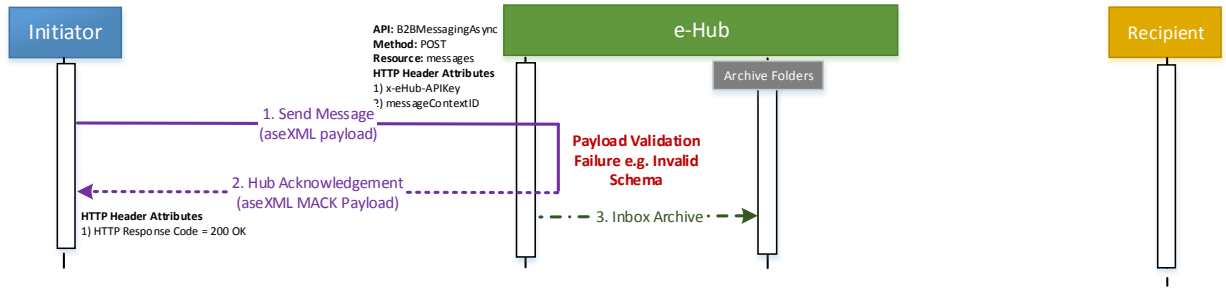


9.1.3.1 Summary

1. Refer section 10 for the list of technical validation failures
2. When a technical validation failure occurs, the incoming message is not archived and the e-Hub returns appropriate HTTP Response Code and description (with details of the exception).

9.1.4 Payload validation failure

Figure 13 – B2BMessagingAsync - Payload Validation Failure



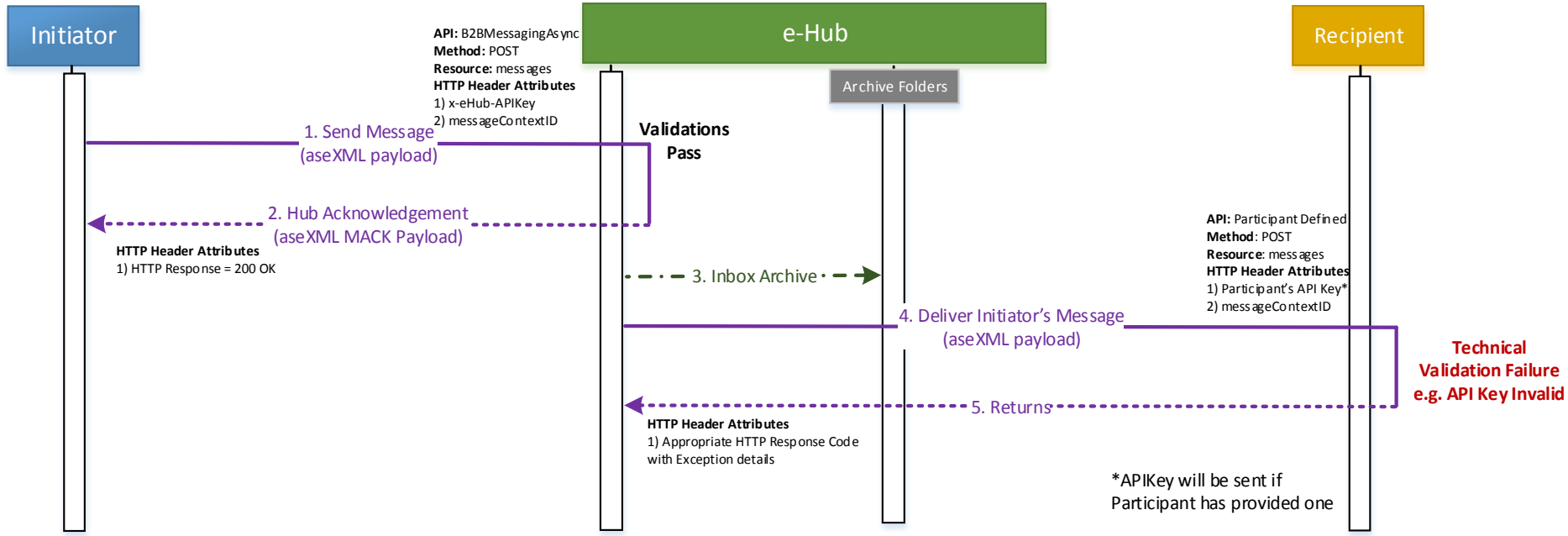
9.1.4.1 Summary

1. Refer section 10 for the list of payload validation failures.
2. When a payload validation failure occurs, the message is archived by the e-Hub. The e-Hub provides hub acknowledgement with HTTP Response Code of '200 OK'. The details of the payload validation failure is provided in the aseXML MACK payload. <From> ParticipantID in the hub acknowledgement is set to 'NEMMCO'.
3. The message is not delivered to the Recipient

© AEMO 2017

9.1.5 Technical validation failure (Recipient)

Figure 14 – Technical Validation Failure (Recipient)



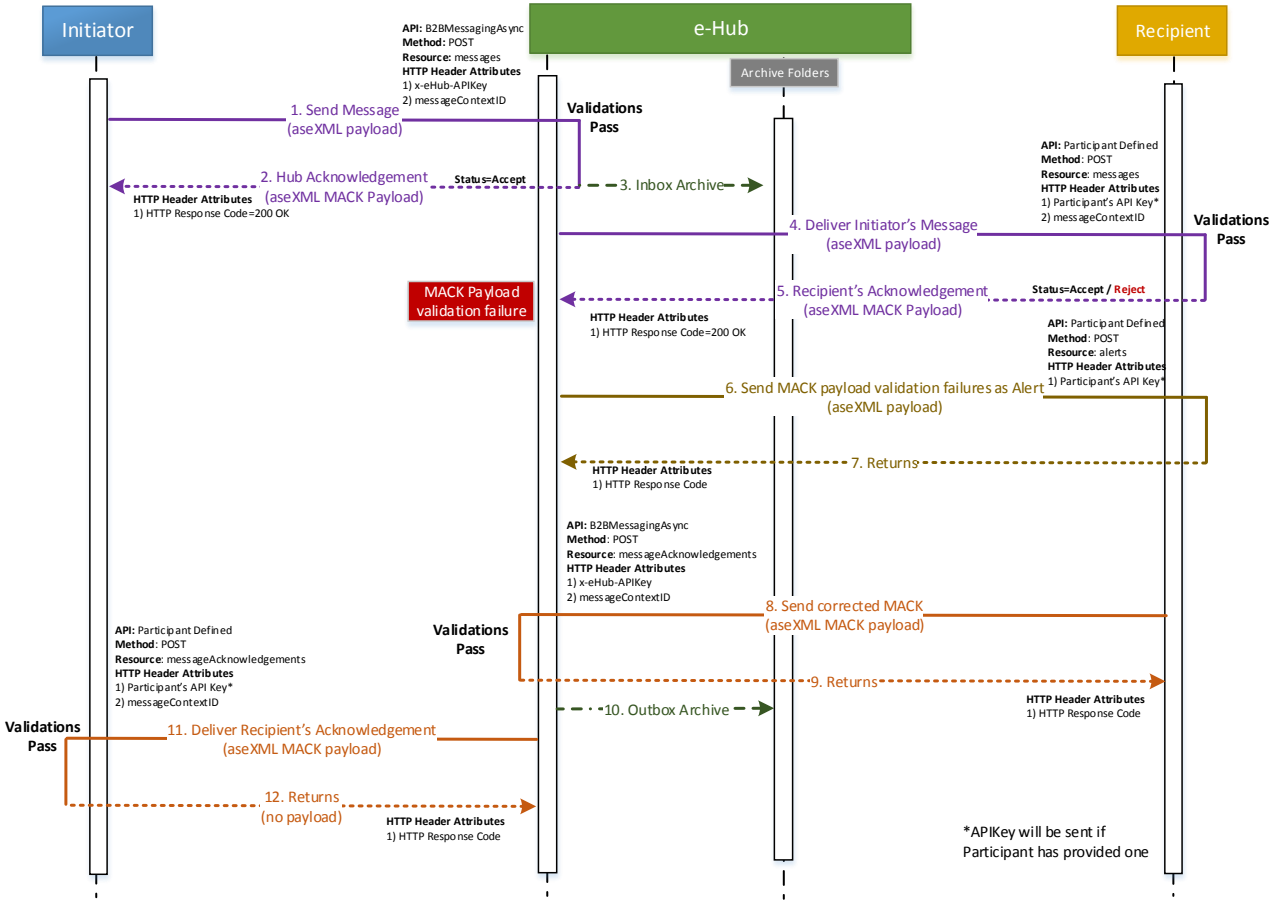
9.1.5.1 Summary

- When a technical validation failure (for example, invalid API Key) occurs when calling the Recipient's endpoint, it is expected that the Recipient returns an appropriate HTTP response code and description (set to the exception details).

© AEMO 2017

9.1.6 Validation failure of Recipient's Acknowledgement Payload (e-Hub)

Figure 15 – B2BMessagingAsync – Recipient Acknowledgement Payload Validation Failure (e-Hub)



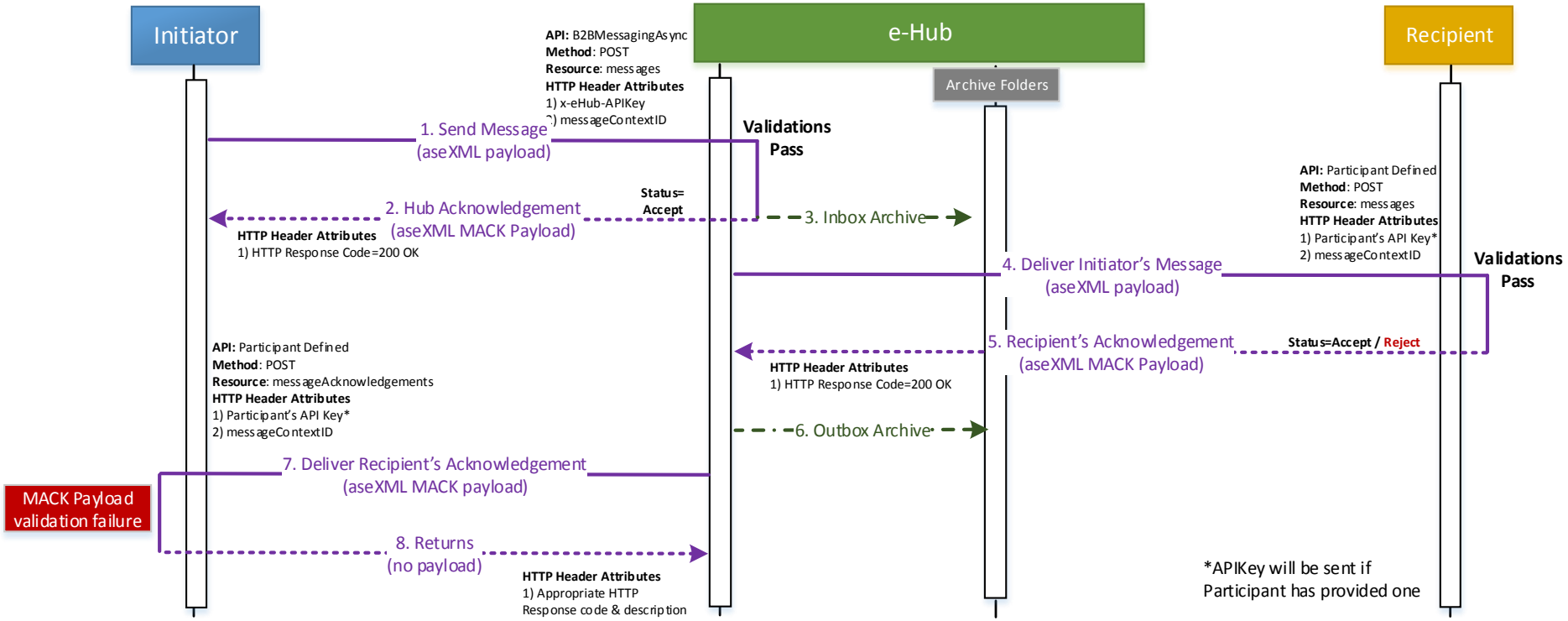
9.1.6.1 Summary

1. If the Recipient message acknowledgement received by the e-Hub is invalid, the e-Hub notifies the Recipient of the error as an alert.
2. The e-Hub notifies the exception details to the Recipient by calling the Recipient's registered Message Management API and using the '/alerts' resource.
3. The exception details are sent in an aseXML document using the new transaction type: PayloadExceptionAlert (refer section 11 for details).
4. The Recipient is expected to correct the issue in the payload and resubmit the message acknowledgement to the e-Hub. The Recipient calls the B2BMessagingAsync API of the e-Hub and uses the resource '/messageAcknowledgements' (as the payload that is being submitted is a Message Acknowledgement message). The messageContextID (when calling the '/messageAcknowledgements' resource) should be set to the original messageContextID of the message to which the MACK is being issued.
    a. The incoming Recipient Acknowledgement payload is validated. If the validations pass, e-Hub sends HTTP response code of 200 OK.
    b. If the payload validations fail, e-Hub sends HTTP response code of 500 and the exception details in the description.
5. The incoming message to the Recipient is archived (to Recipient's outbox archive directory) only when a valid Recipient acknowledgement is received.
6. The Recipient acknowledgement (corrected) is sent to the Initiator.

© AEMO 2017

9.1.7 MACK validation failure (Initiator)

Figure 16 – B2BMessagingAsync API - MACK Validation Failure (Initiator)



9.1.7.1 Summary

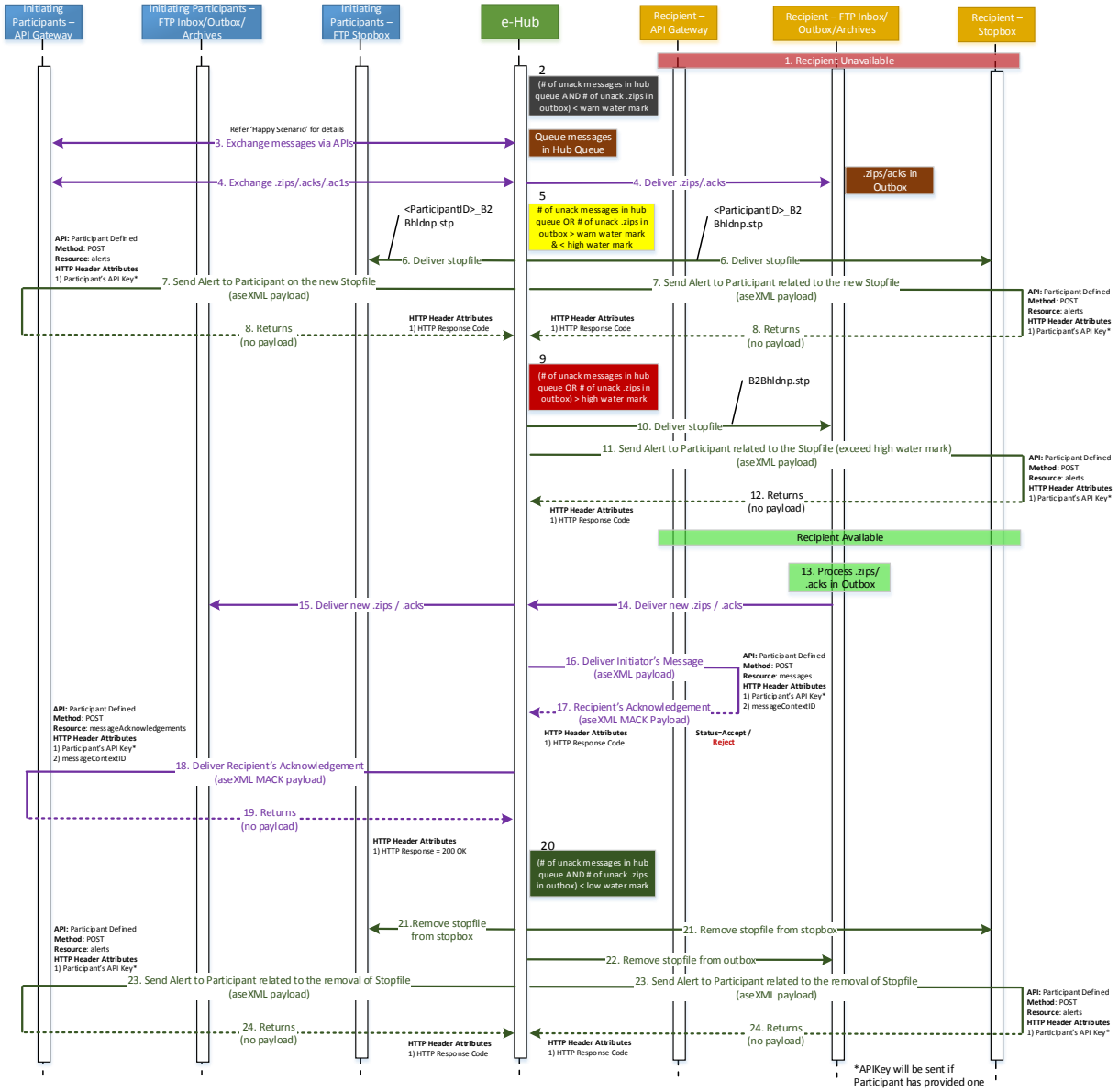
Where Recipient acknowledgement payload fails validation when received by the Initiator, the e-Hub expects a response with the appropriate HTTP response code.

7. If after further investigation, the Initiator determines that the issue is with the e-Hub, the Initiator should contact the AEMO Support Hub.

© AEMO 2017

9.1.8 Recipient unavailable & stop file process walkthrough

Figure 17 – B2BMessagingAsync API - Recipient Unavailable (Stop File Process)



9.1.8.1 Walkthrough

1. The Recipient is unavailable.
2. The incoming messages for the Recipient are stored in the e-Hub queue and incoming files are stored in the Recipient's outbox. The number of unacknowledged messages in e-Hub queue (and) the number of unacknowledged files in outbox < warn watermark.
3. New messages (if any) are stored in the e-Hub queue.
4. New files (if any) are stored in Recipient's outbox.
5. The number of unacknowledged messages in e-Hub queue (or) the number of unacknowledged files in outbox exceeds warn watermark.

6. The e-Hub places stop file in the stop box of all the participants.
7. The e-Hub sends the API stop file alert notification to the participants who have registered their Message Management APIs in the e-Hub. The e-Hub uses the '/alerts' resource.
8. The stop file alert (related to the new stop file) is sent in an aseXML document using the new transaction type: HubFlowControlAlertNotification (refer section 11 for details).
9. The number of unacknowledged messages in e-Hub queue (or) the number of unacknowledged files in outbox > high water mark.
10. The e-Hub places stop file in the outbox of the Recipient (the unavailable participant).
11. The e-Hub sends the API stop file alert notification (stating that the Recipient has exceeded the high water mark) if the Recipient has registered its Message Management APIs in the e-Hub.
12. The e-Hub uses the '/alerts' resource. The stop file alert is sent in an aseXML document using the new transaction type: HubFlowControlAlertNotification (refer section 11 for details).
13. Recipient becomes operational.
14. Recipient processes the files from the Outbox
15. Deliver new zips and ACKs.
16. Step 16, 17, 18 & 19: Deliver Initiators message: The e-Hub delivers the messages in the e-Hub queue to the Recipient via the scheduled delivery process.
17. Recipient's acknowledgement.
18. The Recipient's message acknowledgements are delivered back to the Initiator.
19. Returns no Payload.
20. The number of unacknowledged messages in the e-Hub queue and the number of unacknowledged files in outbox < low water mark.
21. The e-Hub removes the stop file from the stop box of all the participants.
22. The e-Hub also removes the stop file from the Recipient's outbox.
23. The e-Hub sends an API stop file alert notification to all the participants who have registered their Message Management APIs in the e-Hub. The e-Hub will use the '/alerts' resource. The stop file alert (removing an existing stop file) is sent in an aseXML document using the new transaction type: HubFlowControlAlertNotification (refer section 11 for details).

9.1.8.2 Summary

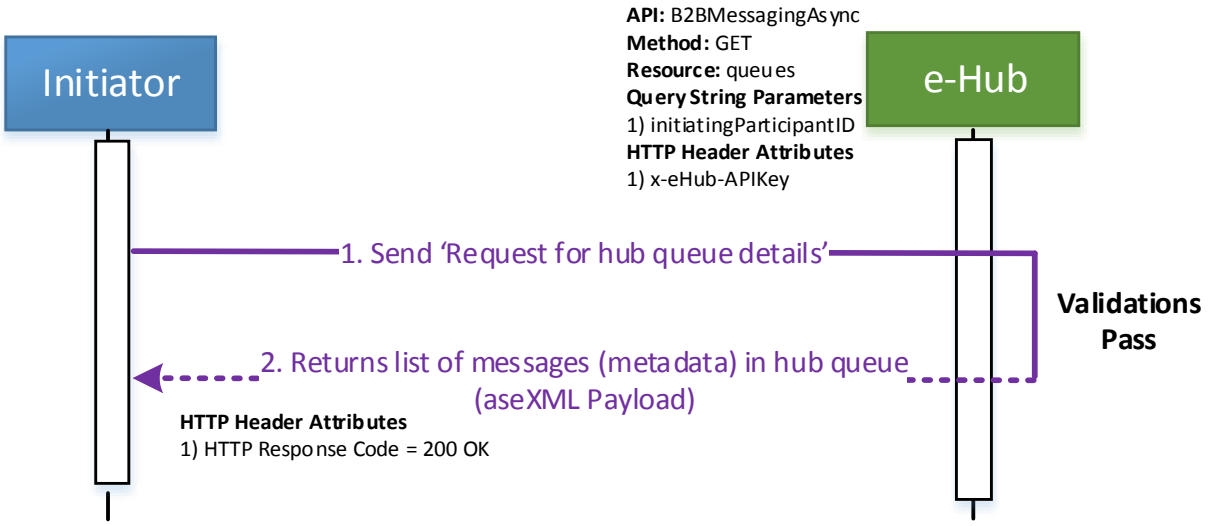
1. When a Recipient is unavailable and the total number of messages in its e-Hub queue /outbox exceed the warn water mark, a stop file will be delivered to all participants' stop boxes, and for those using APIs an alert will be issued.
2. If the same Recipient is still unavailable and the total number of messages in its e-Hub queue/outbox exceed the high water mark, a stop file will be placed in the Recipient's outbox and the e-Hub will attempt to send an alert indicating that the high water mark has been exceeded.

© AEMO 2017

3. When the Recipient becomes available, the removal of the stop file from the outbox of the Recipient and the stop files in all participants' stop boxes will only occur once the number of messages in the Recipient's e-Hub queue and outbox has fallen below the low water mark. Participants using APIs will also be issued with an alert to inform them that a stop file has been removed.

9.1.9 Request Hub Queue details

Figure 18 – B2BMessagingAsync API - Request Hub Queue Details



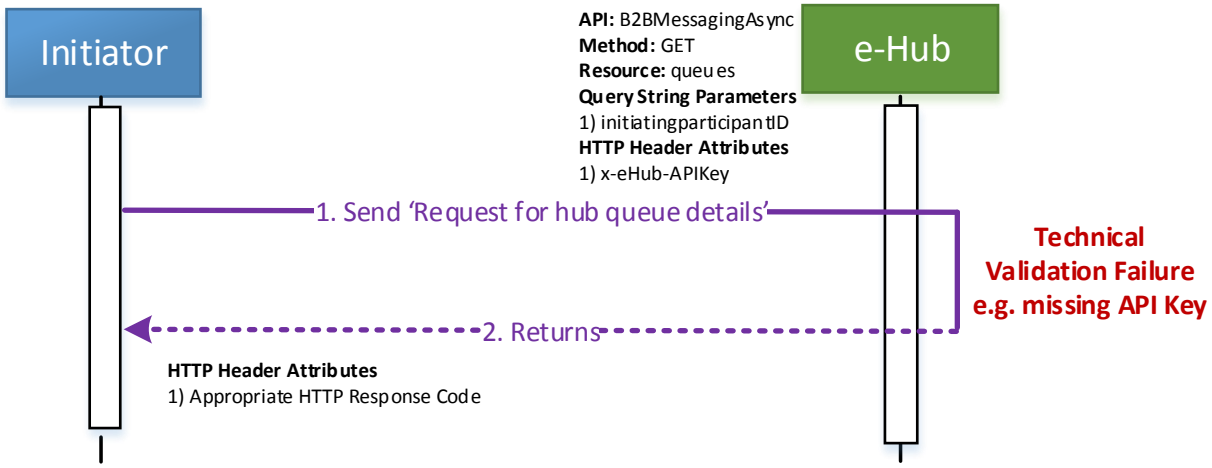
9.1.9.1 Summary

1. Participants can query the e-Hub to retrieve the list of messages queued in the e-Hub queue.
2. Participants call the B2BMessagingAsync API and use the resource '/queues'.

   The metadata of the messages in the e-Hub queue are sent in the aseXML payload (HTTP response payload), new transaction type: HubQueueReport (refer section 11 for details).

9.1.10 Request Hub Queue details – technical validation failure

Figure 19 – B2BMessagingAsync - Request Hub Queue Details – Technical Validation Failure



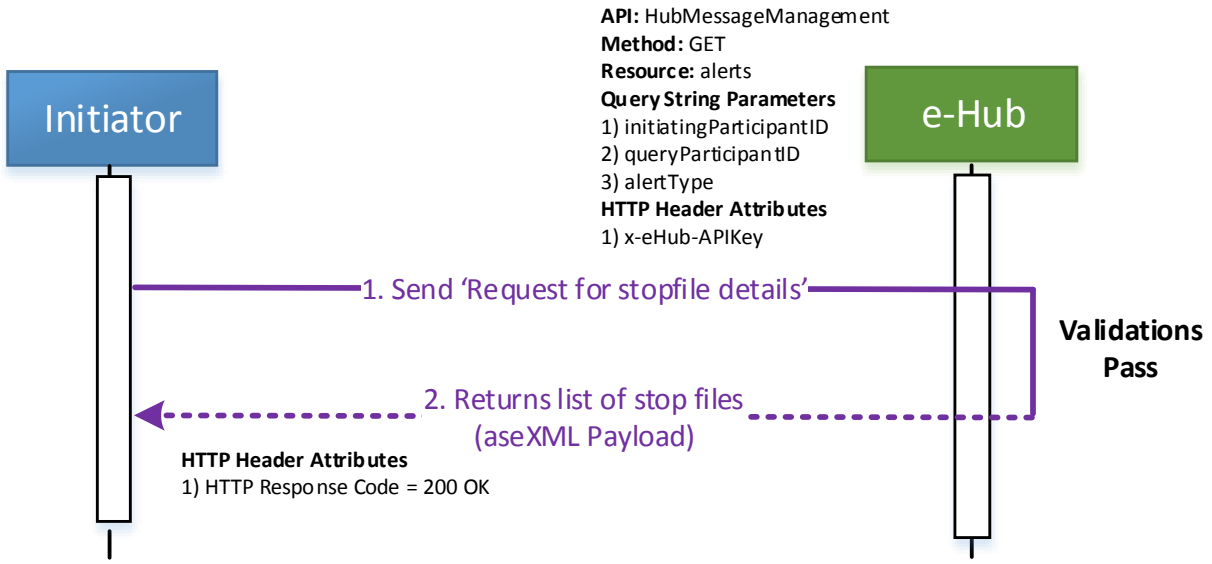
9.1.10.1 Summary

The e-Hub sends appropriate HTTP response code for any technical validation failures or HTTP response code of 500 when validation of the query string parameters fail (description set to the details of the exception).

9.2 Hub Inquiry

9.2.1 Request stop file details

Figure 20 – Request Stop File Details

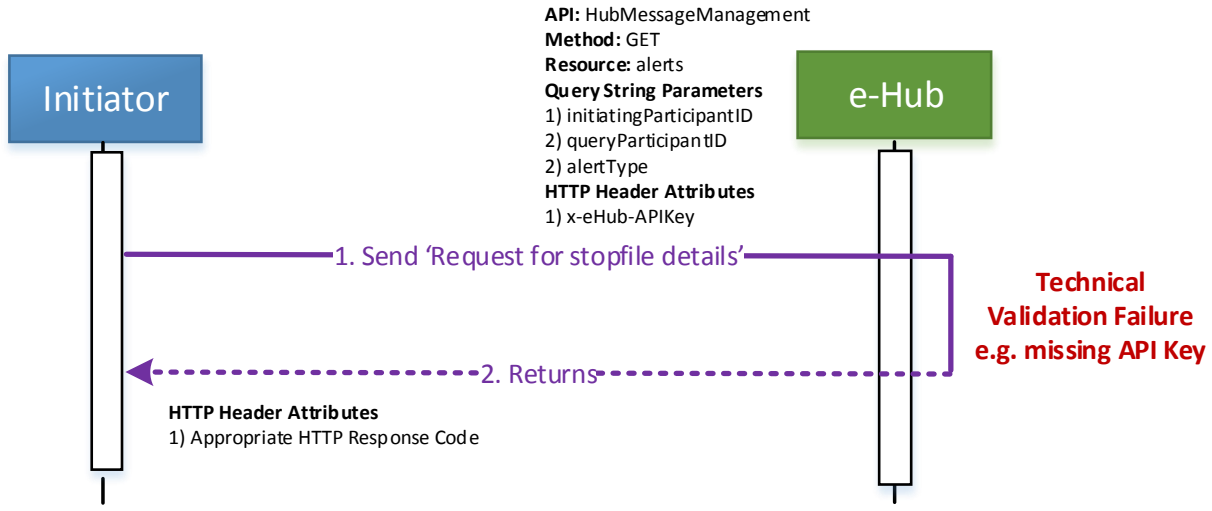


1. Participants can query the e-Hub to retrieve the list of B2B stop files that are currently active in the market.
2. Participant calls the HubMessageManagement API and uses the resource '/alerts' of e-Hub.

3. The stop file report is sent in the aseXML payload (HTTP response payload), new transaction type: HubFlowControlReport (refer section 11 for details).
4. This API is designed as a centralised API/resource providing various alert information.

9.2.2 Request stop file details – technical validation failure

Figure 21 – Request Stop File Details – Technical Validation Failure



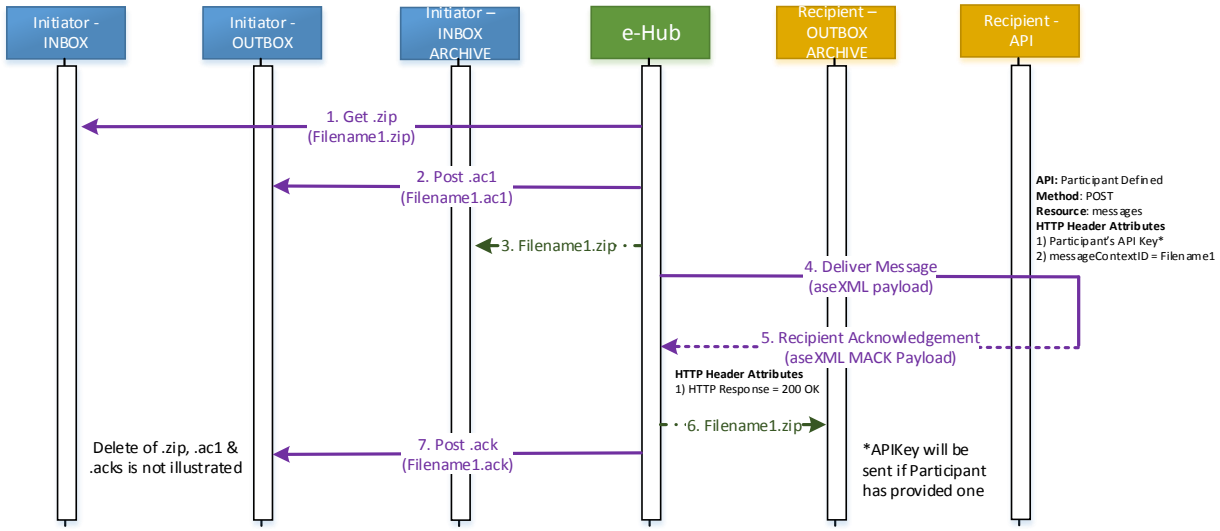
- The e-Hub sends the appropriate HTTP response code for any technical validation failures or HTTP response code of 500 when validation of the query string parameters fail (description set to the details of the exception).

9.2.3 Interoperability

The e-Hub interoperates between FTP and API interfacing methods. Interoperability is not applicable to Sync services and Hub Message Management services.

9.3 File naming (FTP to API)

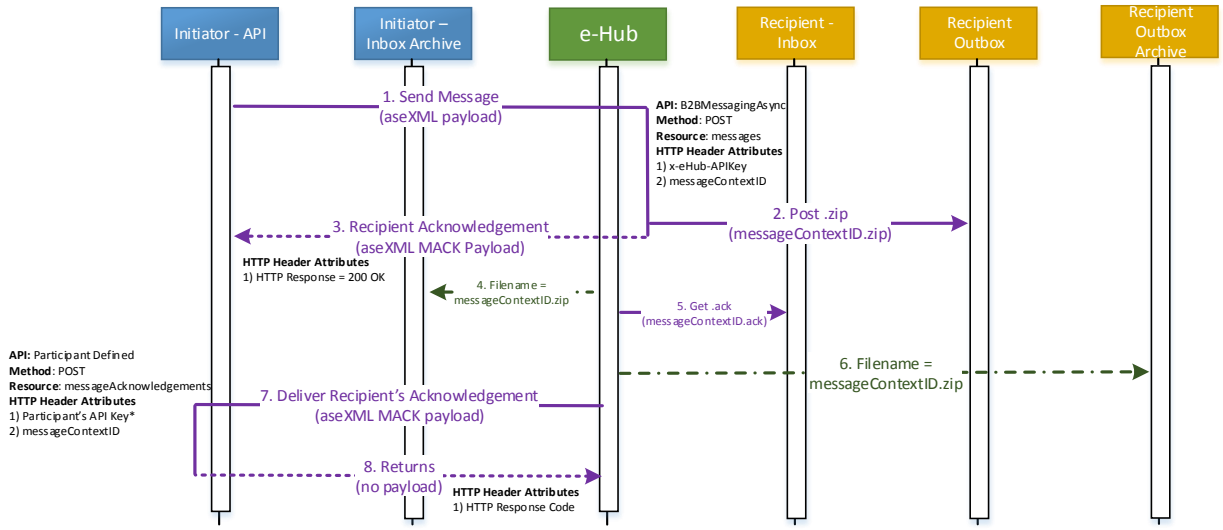
Figure 22 – File naming (FTP to API)



1. Where the Initiator uses FTP, messageContextID of the API call will be set to the file name sent by the Initiator
2. The message delivered to the Recipient will be archived in Recipient's outbox directory (when the Recipient's acknowledgement is received) and the archive file name will be set to <messageContextID>.zip i.e. filename of the .zip sent by the Initiator.

9.3.1 File naming (API to FTP)

Figure 23 – File naming (API to FTP)



1. Where the Initiator is using APIs, the file name delivered to the Recipient will be derived from the messageContextID of the original message i.e. filename sent to the Recipient will be set to messageContextID.zip

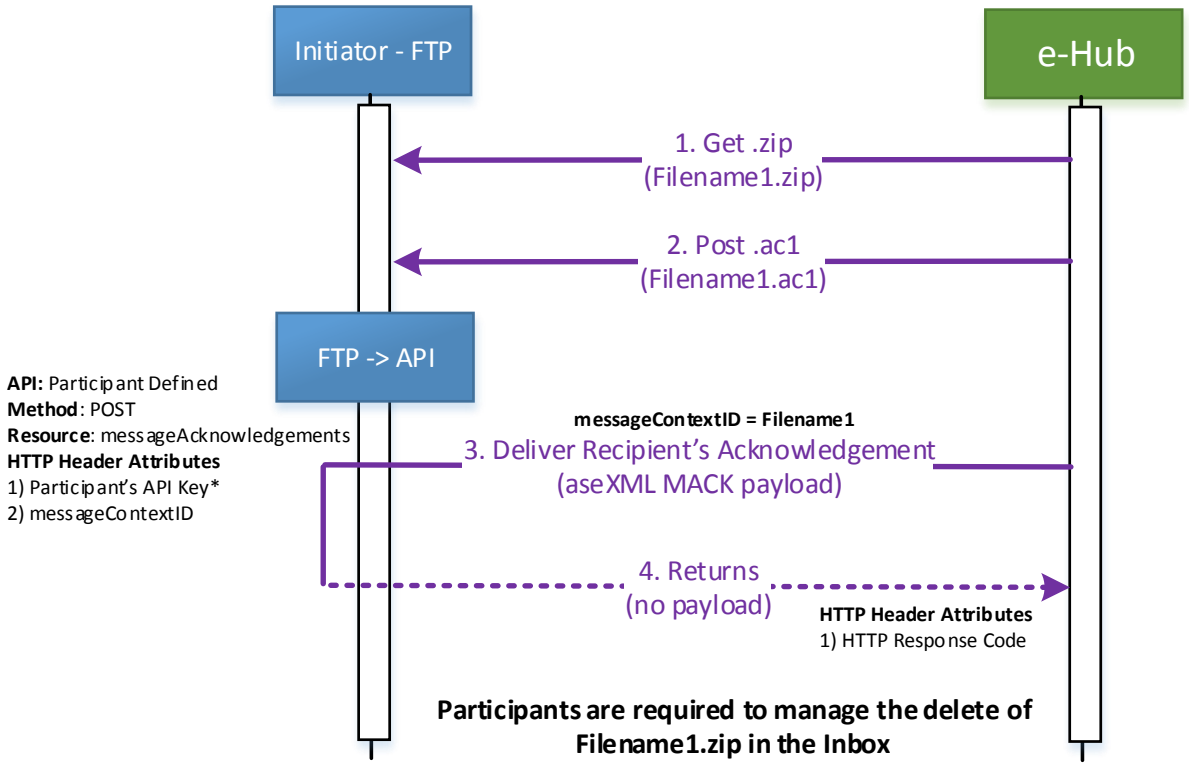
2. Similarly when the Recipient's acknowledgement is sent to the Initiator, messageContextID will be set to the filename of .ack file delivered by the Recipient

9.3.2 Manage in-flight messages during protocol change

Participants can choose the protocol (API or FTP) at the transaction group level using the 'Protocol and Transform' screen (MSATS browser). Participants can complete the protocol switch only when there are no unacknowledged files in the outbox and no unacknowledged messages in the e-Hub queue. This section illustrates how participants need to manage the messages waiting for Recipient's message acknowledgements during a protocol change.

9.3.2.1 FTP to API

Figure 24 – Manage In-Flights FTP to API



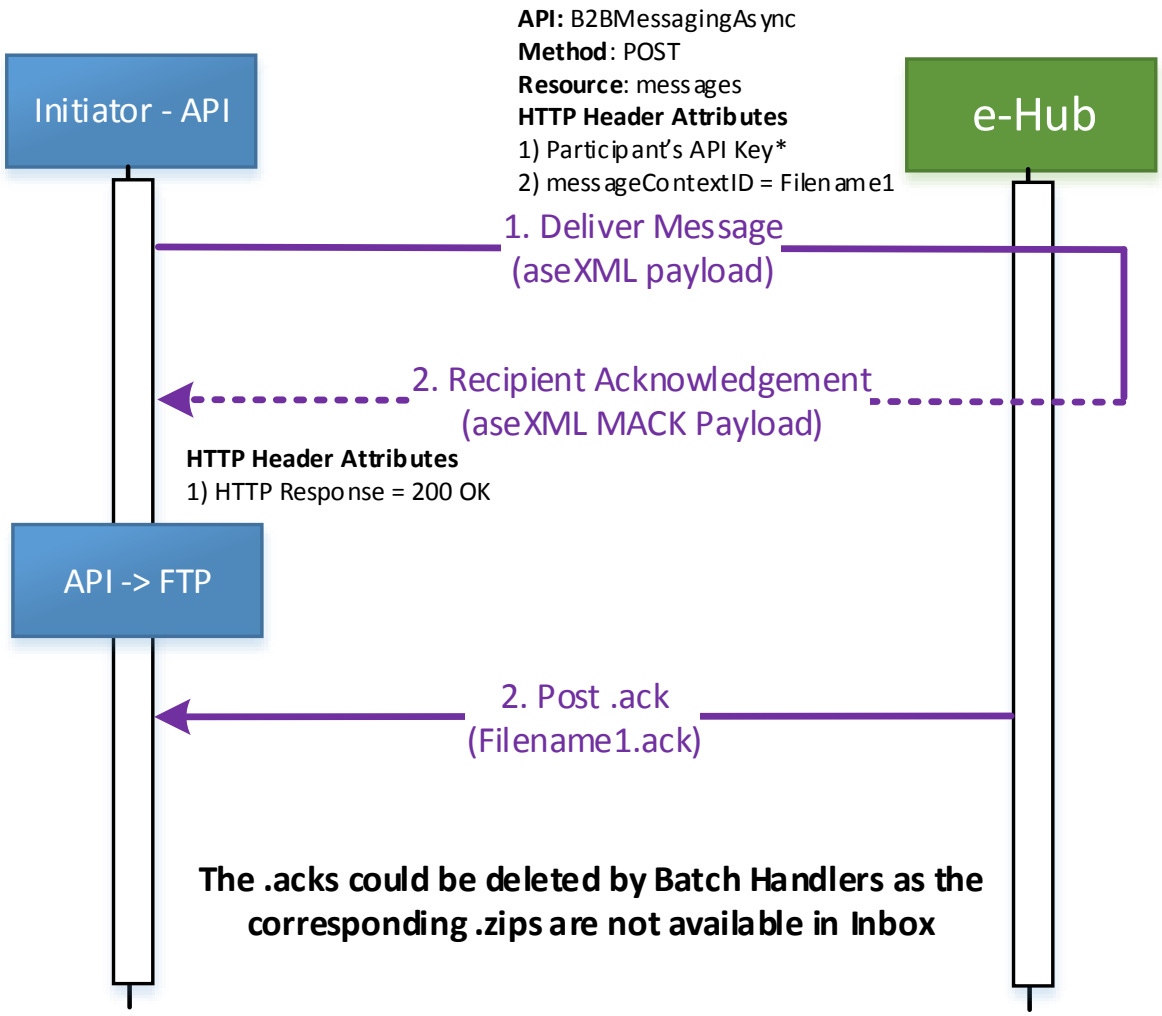
1. The Initiator sends the file to the Recipient and receives the hub acknowledgement i.e. .ac1. The Initiator's outbox does not contain any unacknowledged files.
2. The Recipient has not sent the message acknowledgement.
3. The Initiator changes the protocol from FTP to API (for all the transaction groups).
4. The Recipient sends the message acknowledgement.
5. As the Initiator has changed the protocol, the e-Hub delivers Recipient's message acknowledgement to the Initiator via an API call.
6. Participants (Initiators in the above example) are required to:
    a. Process the message acknowledgement via the new opted protocol.

© AEMO 2017

b. Clean the .zips in their inboxes manually (using MSATS browser or participant's FTP client) because the corresponding .acks are not delivered to the outbox.

9.3.2.2 API to FTP

Figure 25 – Manage In-Flights API to FTP



1. The Initiator sends the message to the Recipient via API and receives the hub acknowledgement.
2. The Recipient has not sent the acknowledgement.
3. The Initiator changes the protocol from API to FTP (for all the transaction groups)
4. The Recipient sends the message acknowledgement.
5. As the Initiator has changed the protocol, the e-Hub delivers the Recipient's message acknowledgement to the Initiator via FTP i.e. create '.ack' in the outbox of the Initiator.
6. In this scenario, an .ack exists without the corresponding .zip in the Inbox. The e-Hub could delete the .ack (as there is no corresponding .zip) before the Initiator picks the .ack file.
7. Participants (Initiators in the above example) are required to:

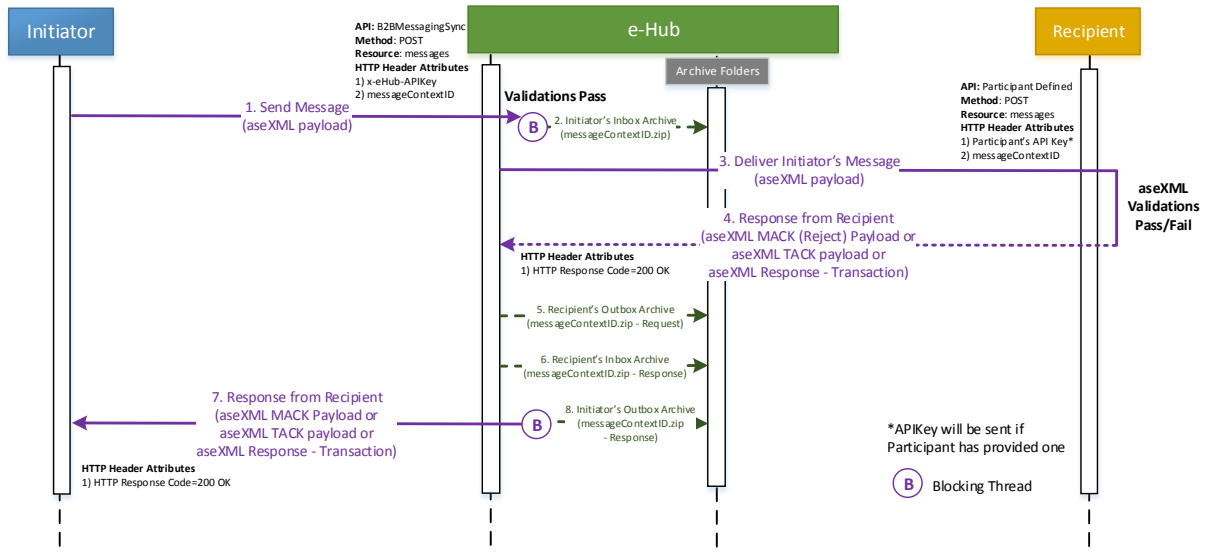
© AEMO 2017

c. Use MSATS browser (Search Trans Log screen) to track the status of the inflight messages (se Messaging

9.4 Synchronous Messaging

9.4.1 Normal processing (Sync)

Figure 26 – Normal Processing (sync)



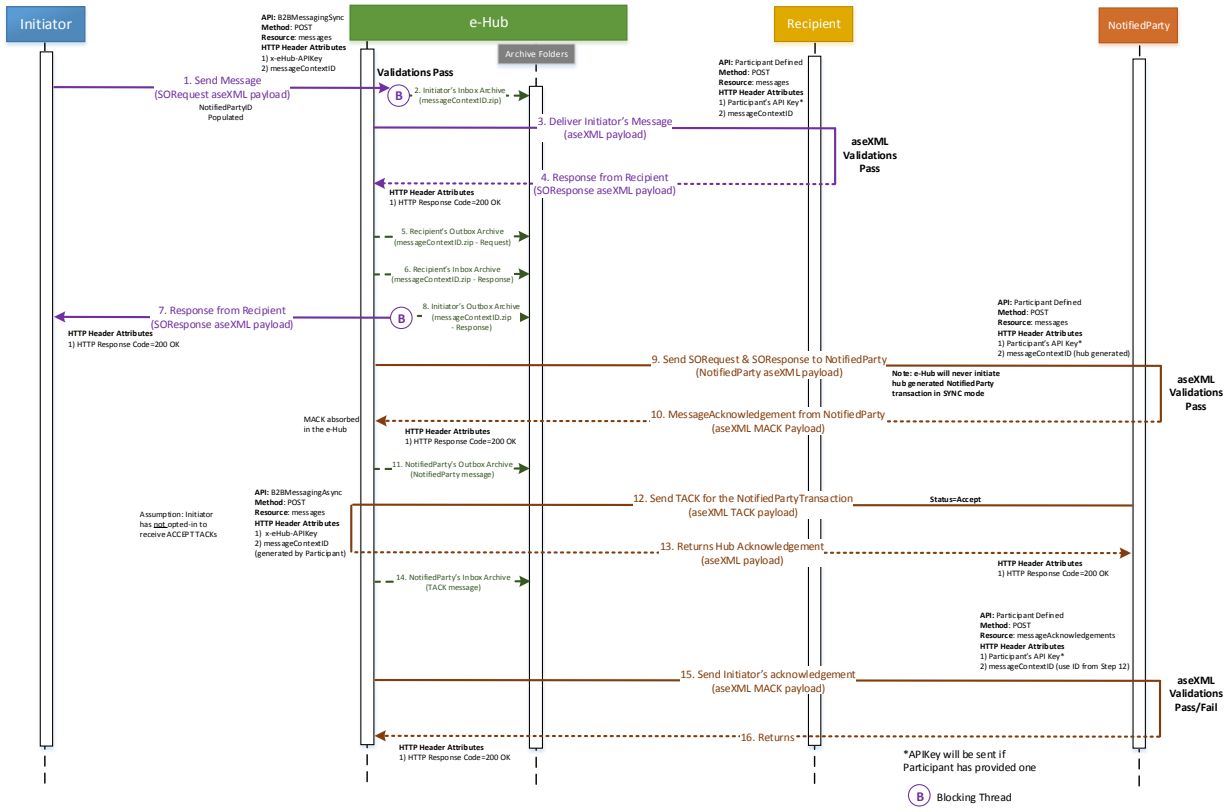
1. Where both Initiator and Recipient have agreed to synchronous messaging, a blocking thread is created and only closed when a response is provided by the Recipient (which may be a MACK payload (status = 'Reject'), TACK payload (status = 'Reject' or 'Accept' or 'Partial') or response payload).
2. The Initiator calls B2BMessagingSync API and uses resource '/messages' to initiate the synchronous request.
3. The incoming message must contain only one transaction, bundling of transactions is not permitted.
4. The e-Hub validates the incoming request and holds the incoming request. The incoming message is archived in Initiator's inbox archive directory (filename: messageContextID.zip).
5. The message (transformed if required) is sent to the Recipient by calling the participant's registered synchronous API and using '/messages' resource.
6. The Recipient validates the incoming request and sends one of the following responses:
    a. Message Acknowledgement (status = 'Reject') if the validation of incoming messages fails.
    b. Transaction Acknowledgement (status = 'Reject') if the business validation of incoming transaction fails.
    c. Transaction Acknowledgement (status = 'Accept' or 'Partial') if the business response (for example, service order response) will not be delivered within the e-Hub defined timeframes / the request does not require any business response (for example, CustomerDetailsNotification).

d. Valid business response i.e. a Transaction Message.

7. The incoming message to the Recipient is archived in Recipient's outbox archive directory (filename = messageContextID.zip) and the response message from the Recipient is archived in Recipient's inbox archive directory (filename = messageContextID.zip).
8. The blocking thread to the Initiator is responded with the response message (message is transformed if required) received from the Recipient.
9. The response message is archived in the Initiator's outbox archive directory (filename = messageContextID.zip).

9.4.2 Normal processing – Notified Party (Sync)

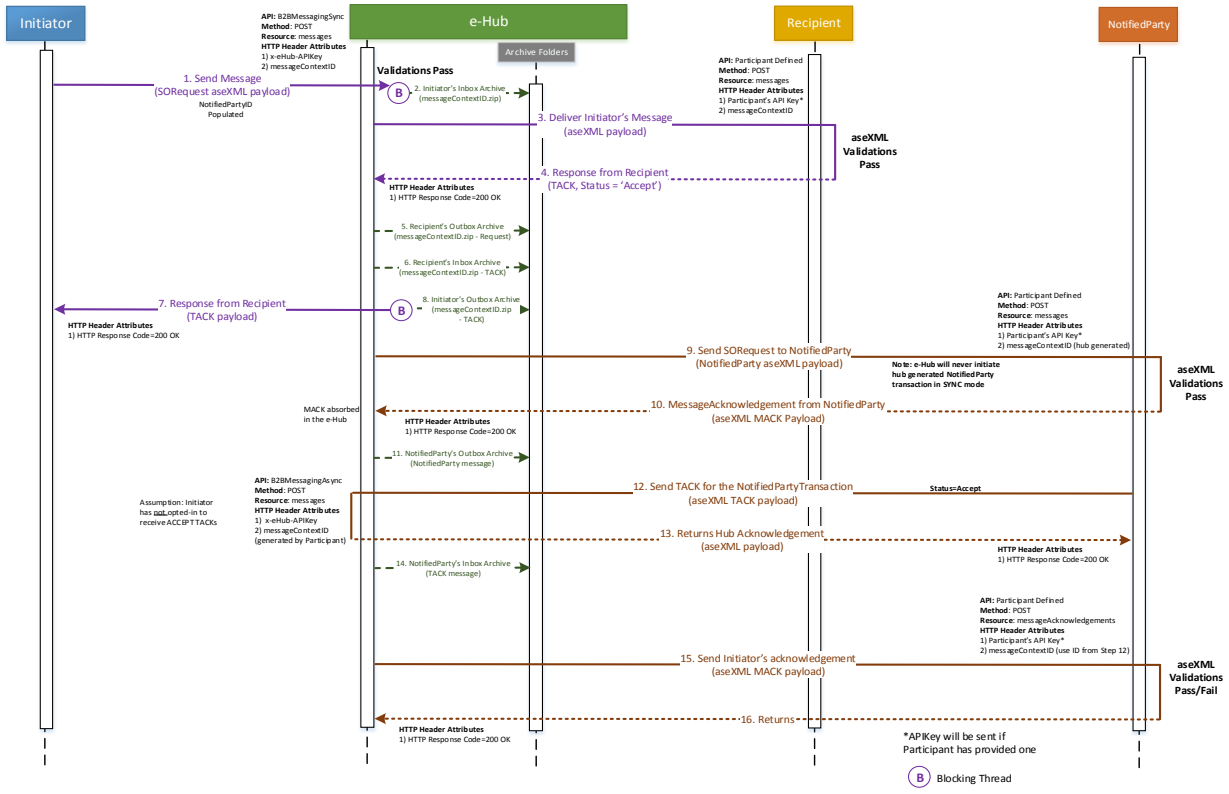
Figure 27 – Normal Processing – Notified Party (sync)



1. In the Service Order scenario above with Notified Parties where Initiator and Recipient use synchronous services, the e-Hub always initiates / manages the corresponding NotifiedParty messages using the asynchronous pattern (if NotifiedParty opts-in for APIs).
2. The NotifiedParty transactions related to Service Order Request and Service Order Response will be bundled in one message.

9.4.3 Normal processing – Recipient TACK status = 'Accept' (Sync)

Figure 28 – Recipient TACK message (Sync)

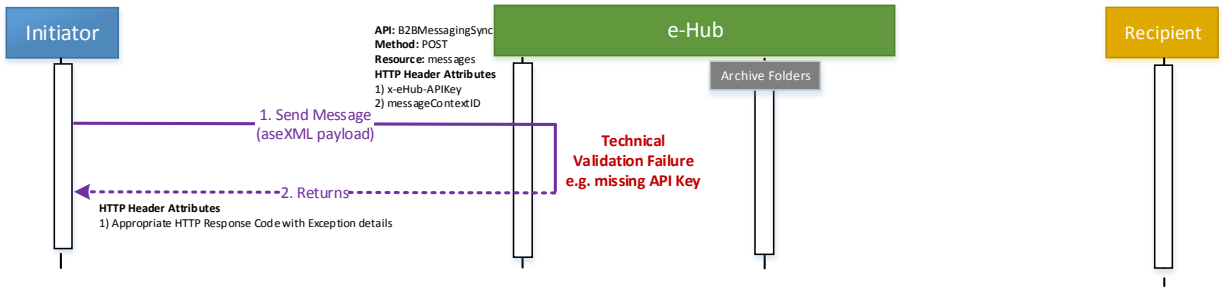


3. The Recipient sends TACK message (status of 'Accept') as the response to the synchronous call. The TACK message will be sent back to the Initiator. The Initiator and the Recipient will mutually agree on the mechanism (using sync or async process or other alternate means) to process the corresponding responses if any (for example, Service Order Response).
4. If the Initiator sends Service Order Request with NotifiedPartyID list populated and if the Recipient sends:
   a. MACK, status = 'Reject' – No messages will be triggered by the e-Hub to the NotifiedParty(ies)
   b. TACK status = 'Accept' – Details related to Service Order Request will be sent to NotifiedParty(ies) using NotifiedParty transaction
   c. TACK status = 'Reject' - No messages will be triggered by the e-Hub to the NotifiedParty(ies)
   d. Response – Details related to Service Order Request & Service Order Response (two different NotifiedParty transactions bundled in one message) will be sent to the Notifiedparty(ies)

© AEMO 2017

9.4.4 Technical validation failure (sync)

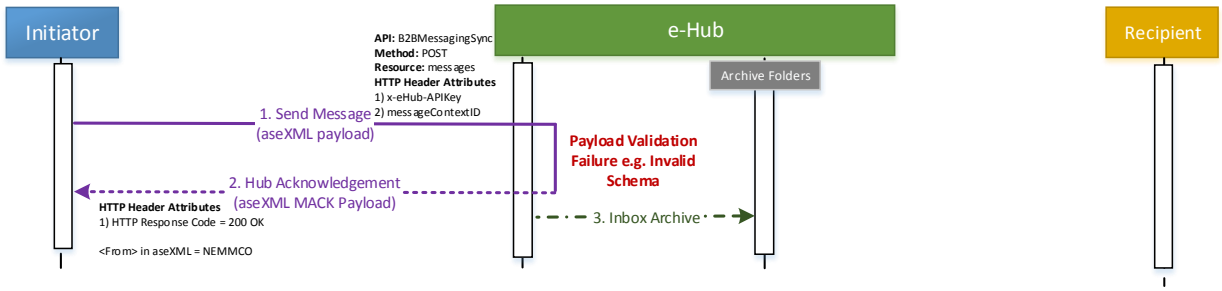
Figure 29 – Technical Validation Failure – e-Hub (sync)



- In the event of a technical validation failure, the e-Hub will return a response with the appropriate HTTP response code and exception details in the response code description. The incoming message is **not** archived by the e-Hub.

9.4.5 Payload validation failure (sync)

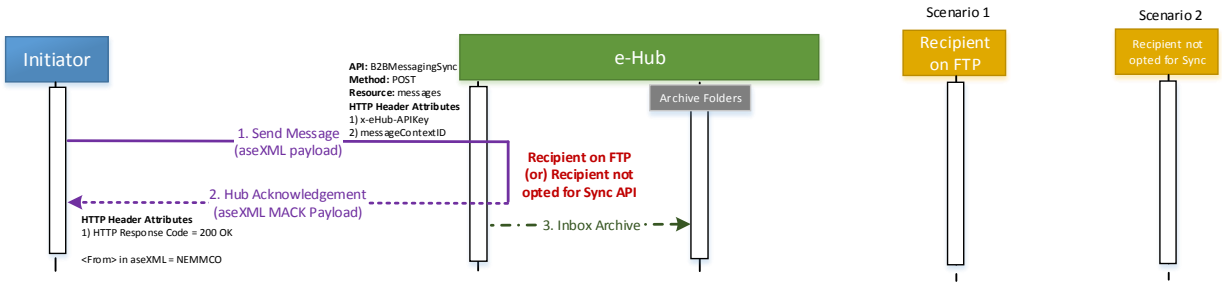
Figure 30 – Payload Validation Failure (sync)



- In the event of a payload validation failure, the e-Hub will archive the message and provide a hub acknowledgement (Response Code '200 OK'), and details of the payload validation failure is provided in the aseXML payload. The 'From' element in the aseXML payload is set to 'NEMMCO'.

9.4.6 Recipient on FTP not opted for sync services

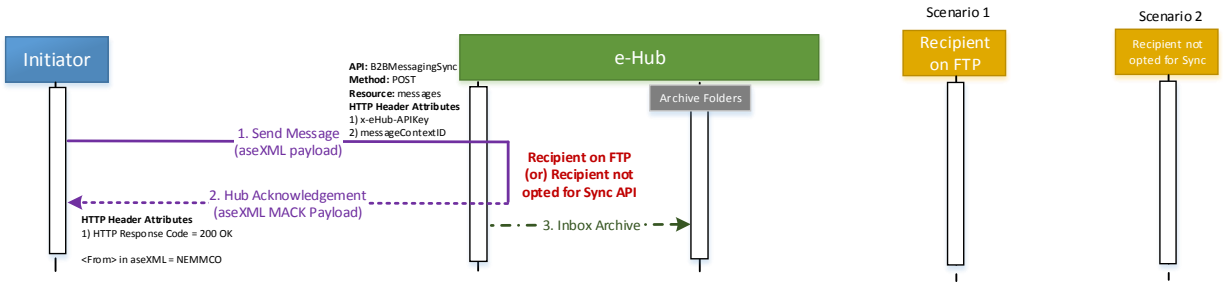
Figure 31 – Recipient on FTP/not opted for sync



1. Queuing of messages and protocol interoperability is not applicable to synchronous services
2. The e-Hub will reject the incoming synchronous messaging requests if the Recipient is not registered for synchronous services.

9.4.7 Recipient unavailable (Sync)

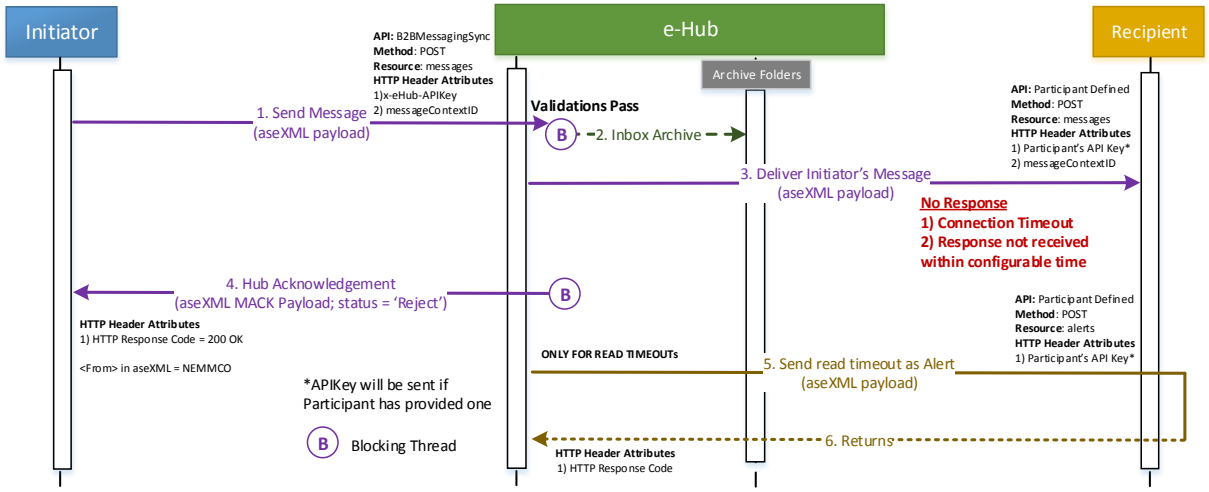
Figure 32 – Recipient Unavailable (sync)



1. Definition of parkbox process: The participant is in the process of updating their incoming aseXML version (or) the opted protocol (FTP or API) using the B2B Transform Screen.
2. In the event the Recipient is unavailable the message is rejected as e-Hub queuing is not applicable for synchronous messaging.
3. The Initiator and Recipient must resolve any issues and/or establish workarounds until the Recipient's system becomes available.

9.4.8 Recipient connection timeout (sync)

Figure 33 – Recipient Connection Timeout (sync)



Definition:

1. Connection timeout: e-Hub is unable to connect to the Recipient's endpoint
2. Read timeout: the e-Hub is connected to Recipient's endpoint but the Recipient does not respond within the configured time

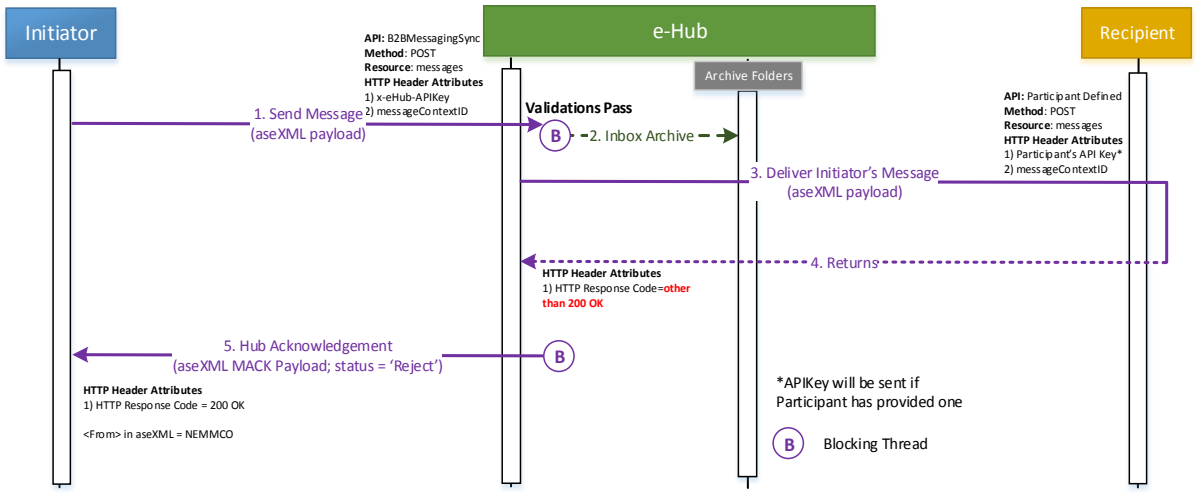
Rules:

1. In the event of a connection timeout or read timeout scenario, the e-Hub responds to the blocking thread (to the Initiator) with the appropriate exception details (i.e. appropriate event code) in aseXML message

2. In the event of read timeout, the e-Hub sends an alert to the Recipient stating that the connection has been terminated. The e-Hub will send this alert by calling the participant's registered hub message management API and using the '/alerts' resource. The alert will be sent in the aseXML message; transaction type: PayloadExceptionAlert
3. The Initiator and Recipient must resolve issues related to read timeouts i.e. determine the next steps and mechanism to re-deliver the response message.

9.4.9 Recipient HTTPS rejection (sync)

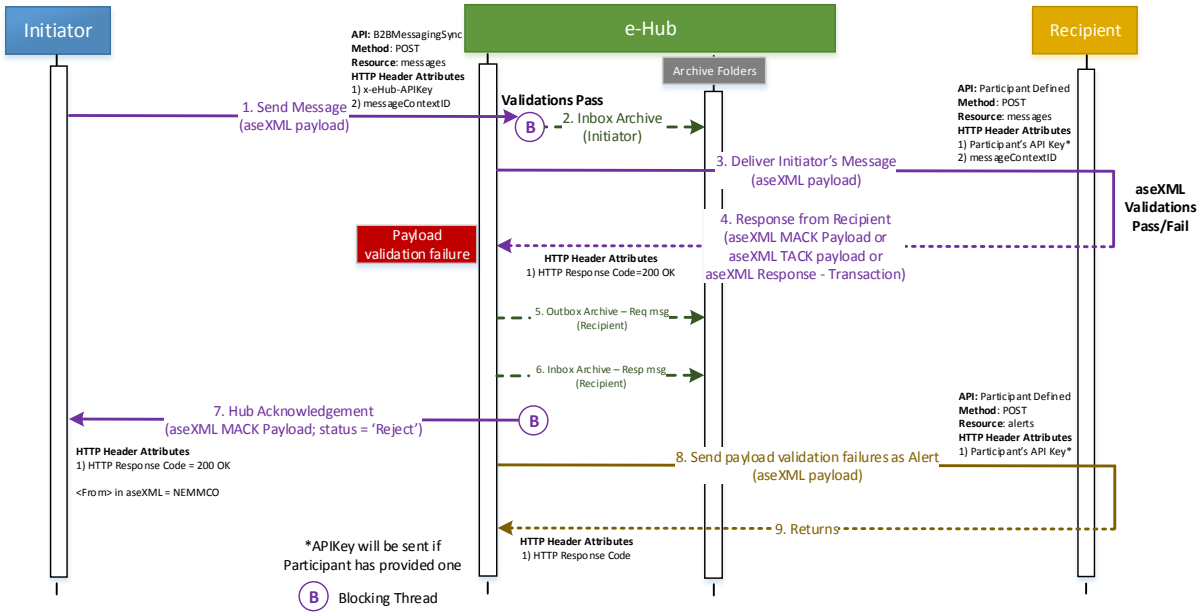
Figure 34 – Recipient HTTPS Rejection (sync)



- If the e-Hub receives a HTTP response code other than '200 OK' from the Recipient, the e-Hub responds to the blocking thread (to the Initiator) with the negative hub acknowledgement i.e. aseXML message NACK payload (refer section 10 for details on the event code and description).

9.4.10 Payload validation failure (Recipient) (sync)

Figure 35 – Payload Validation Failure (Recipient) (sync)

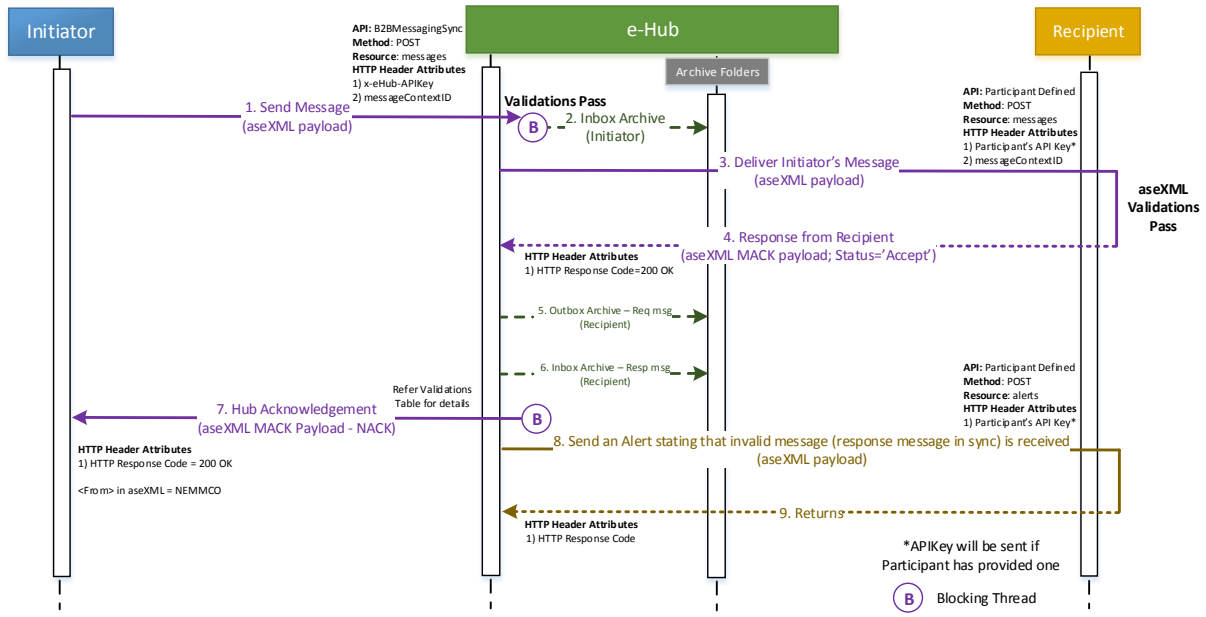


1. In the scenario above where the response from the Recipient fails payload validation at the e-Hub, the e-Hub will still archive the response from the Recipient, and return a negative hub acknowledgement to the Initiator. The e-Hub notifies the Recipient of the payload validation failure in the form of an alert, with details of the failure in the alert payload such as, e-Hub will call the Recipient's registered message management API using '/alerts' resource and aseXML payload; transaction type of PayloadExceptionAlert.
2. The Initiator and Recipient must resolve the issues and/or establish workarounds to process the corrected response from the Recipient.

© AEMO 2017

9.4.11 Incorrect message Payload – MACK Status = 'Accept' (sync)

Figure 36 – Incorrect Message Payload – MACK status ='Accept' (sync)



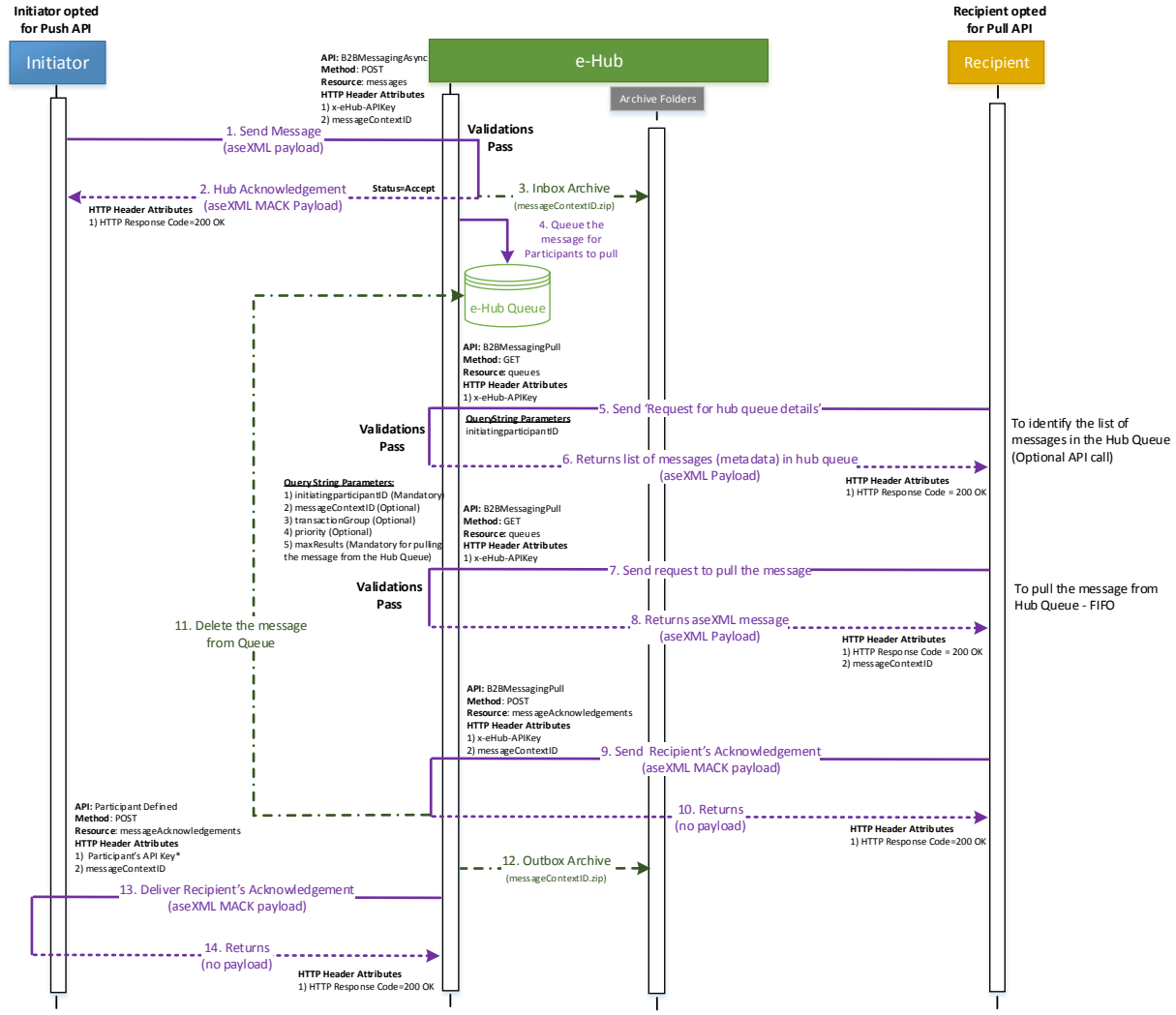
1. The following response payload is expected from the Recipient when the Initiator initiates the synchronous request:
   a. Message Acknowledgement payload with status = 'Reject'.
   b. Event-only payload.
   c. Transaction Acknowledgement payload with status = 'Accept' / 'Reject' / 'Partial'.
   d. Response payload.
2. In the event of receiving an incorrect message payload (for example, Message Acknowledgement payload with status = 'Accept'), the e-Hub will notify the Recipient of the validation failure in the form of an alert, with details of the failure in the aseXML payload i.e. e-Hub calls the Recipient's registered message management API using '/alerts' resource and aseXML payload; transaction type of PayloadExceptionAlert

Note: If the Initiator's request is read timed out (or) timed out due to network issues, it is up to the Initiator and Recipient to resolve any issues and/or establish workarounds to process the lost / unprocessed messages.

9.5 Push-Pull messaging

9.5.1 Normal processing (Recipient opted) (Pull)

Figure 37 – Normal Processing (Recipient opted) (pull)



9.5.1.1 Walkthrough

In the above scenario, the Initiator has registered for B2BMessagingAsync API services and the Recipient has registered for B2BMessagingPull API services:

1. The Initiator sends the message to the Recipient via the e-Hub. The e-Hub sends the hub acknowledgement to the Initiator after validating the incoming message.
2. Since the Recipient has registered for B2BMessagingPull API, the message is queued in the e-Hub queue.
3. B2BMessagingPull API & resource '/queues' can be utilised by the participants in the following ways.
    a. To view the metadata of the messages in the e-Hub queue.
    b. To pull the messages from the e-Hub queue.

4. The Recipient uses B2BMessagingPull API and resource '/queues' to view the meta- data of messages queued in the e-Hub queue. The e-Hub will provide the metadata of messages in the e-Hub queue if the query string parameter 'maxResults' is not passed when calling the B2BMessagingPull API.
5. The Recipient uses B2BMessagingPull API and resource '/queues' to pull the message from the e-Hub queue. Query string parameter 'maxResults' is mandatory to pull the message from e-Hub queue. The aseXML message/payload will be sent as the response with its reference messageContextID.
6. The Recipient can send additional query string parameters (in addition to maxResults) to pull specific types of messages from the e-Hub queue:
    a. messageContextID – The message(s) matching the messageContextID will be returned
    b. transactionGroup – The message(s) matching the Transaction Group will be returned
    c. priority – The message(s) matching the priority will be returned back

    One or all of the above query string parameters can be sent in the HTTP call. If there are multiple messages in the e-Hub queue matching the above criteria, the oldest message (FIFO) matching the requested criteria will be returned.

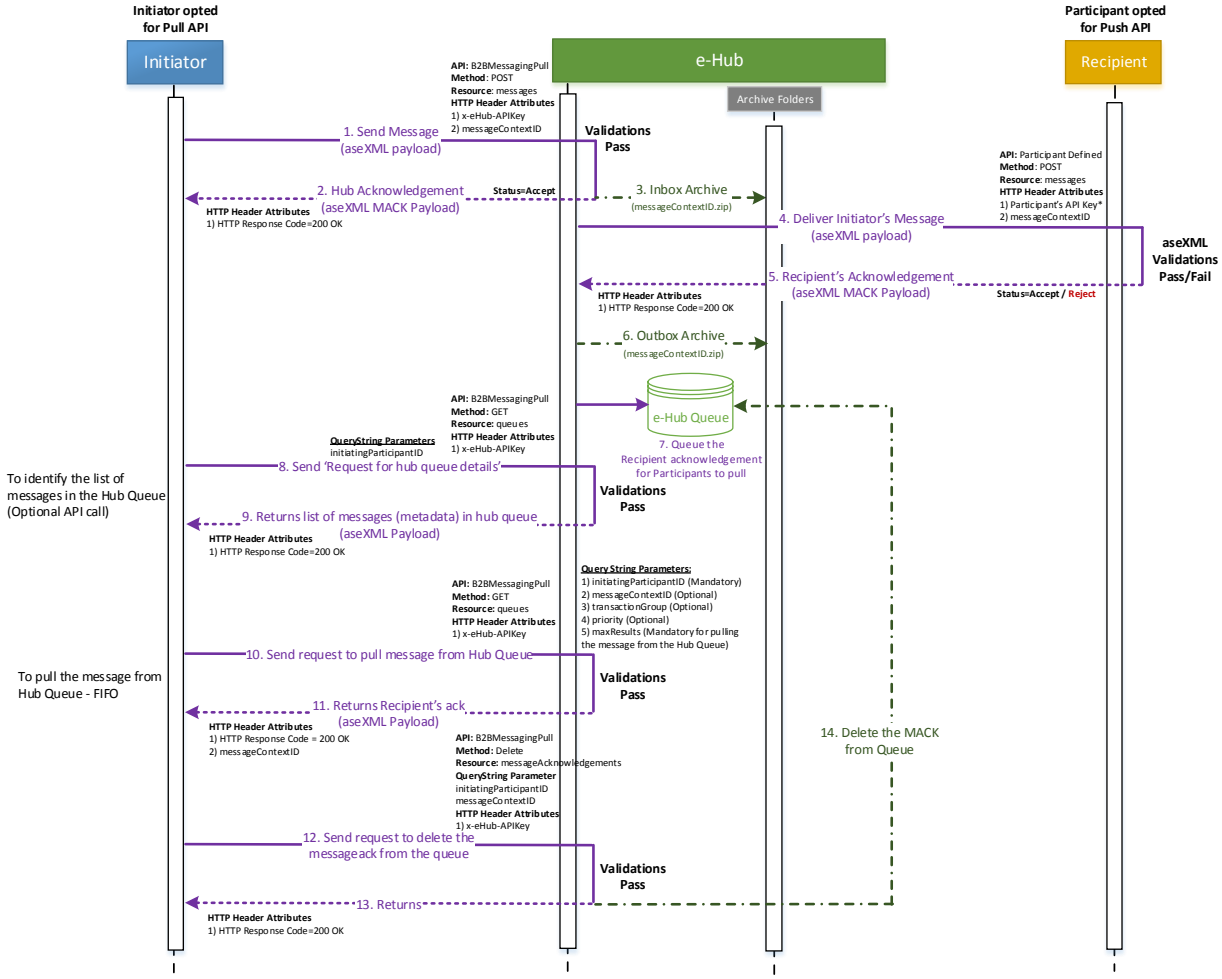
    If none of the above parameters are sent, the oldest message (FIFO) in the e-Hub queue will be returned.

    Note: Query string parameters can accept only one value i.e. transactionGroup can be set to 'SORD' but not a list of transaction groups such as 'SORD, CUST'
7. The Recipient is required to send Recipient message acknowledgement for each of the message pulled from the e-Hub queue (not applicable to the Message Acknowledgements pulled from the e-Hub).
8. The Recipient sends the message acknowledgement using B2BMessagingPull API and resource '/messageAcknowledgements'. messageContextID of the original message is sent (in request header) when posting the message acknowledgement.
9. The message that was pulled by the Recipient is deleted from the e-Hub queue only when the corresponding Recipient acknowledgement is received.

© AEMO 2017

9.5.2 Normal processing (Initiator opted) (Pull)

Figure 38 – Normal Processing (Initiator opted) (pull)



9.5.2.1 Walkthrough

In the above scenario, the Initiator has registered for B2BMessagingPull API services and the Recipient has registered for B2BMessagingAsync API services:

1. The Initiator sends the message to the Recipient via the e-Hub. The Initiator uses B2BMessagingPull API and resource '/messages' to send the message to the e-Hub.
2. The e-Hub validates the incoming message and sends the hub acknowledgement.
3. The message (message is transformed if required) is sent to the Recipient. The Recipient sends the acknowledgment.
4. Since the Initiator has registered for B2BMessagingPull API services, the Recipient acknowledgement is queued in the e-Hub queue.
5. The Initiator uses B2BMessagingPull API and resource '/queues' to view the meta- data of messages queued in the e-Hub queue. Refer section 9.5.1 for details.

6. The Recipient uses B2BMessagingPull API and resource '/queues' to pull the Recipient's acknowledgement from the e-Hub queue. Query string parameter 'maxResults' is mandatory to pull messages from the e-Hub queue. The aseXML message/payload will be sent as response with the reference messageContextID. In the above scenario, the e-Hub will send the Recipient acknowledgement that is queued in the e-Hub queue.
7. The Recipient can send additional query string parameters to pull specific types of messages from the e-Hub queue. Refer section 9.5.1 for details.
8. After processing the acknowledgment, the Initiator notifies the e-Hub that the Recipient acknowledgement has been processed. The Recipient uses B2BMessagingPull API, resource '/messageAcknowledgements' and DELETE method. The reference messageContextID is passed in the query string parameter. The e-Hub deletes the Recipient's acknowledgement from the e-Hub queue.

9.5.2.2 Summary

The participant has registered for B2BMessagingPull:

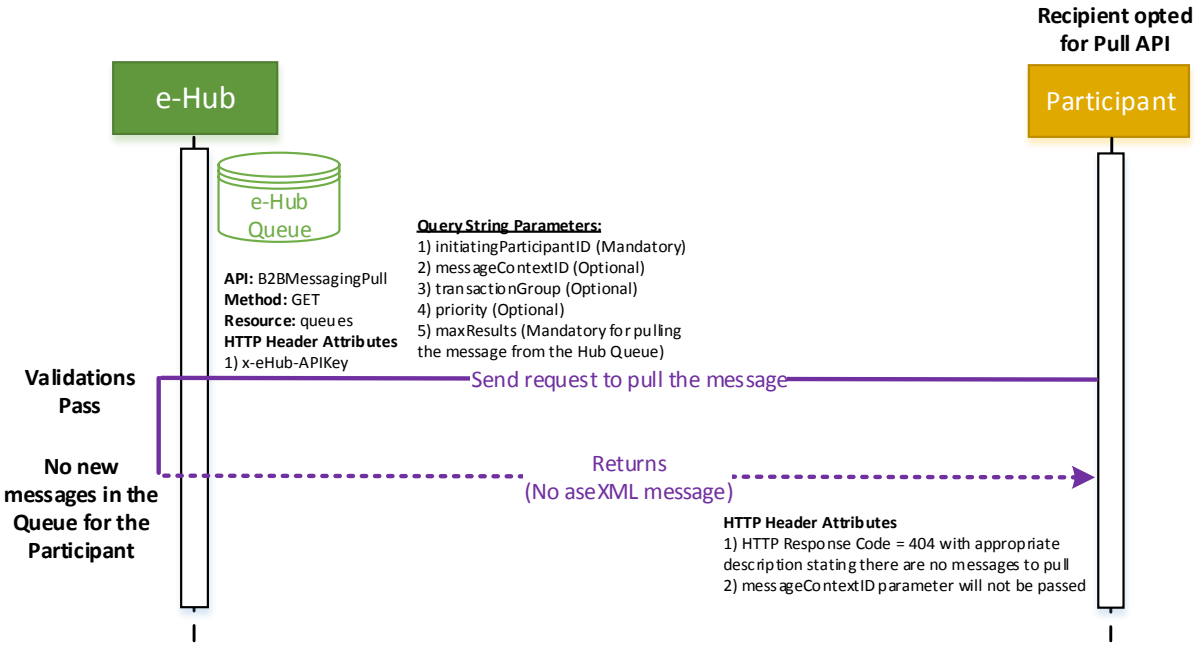
1. If a participant has pulled Transaction message or Transaction Acknowledgement message from the e-Hub queue, the participant is required to send Message Acknowledgement (MACK).
2. If a participant has pulled Message Acknowledgement message from the e-Hub queue, the participant is required to notify the e-Hub to delete the MACK from the e-Hub queue.

9.5.3 Exception Scenarios related to B2BMessagingPull API & '/messages' and 'messageAcknowledgements' resource

Refer to exception scenarios in Section 9.1.

9.5.4 Normal processing (empty queue) (Pull)

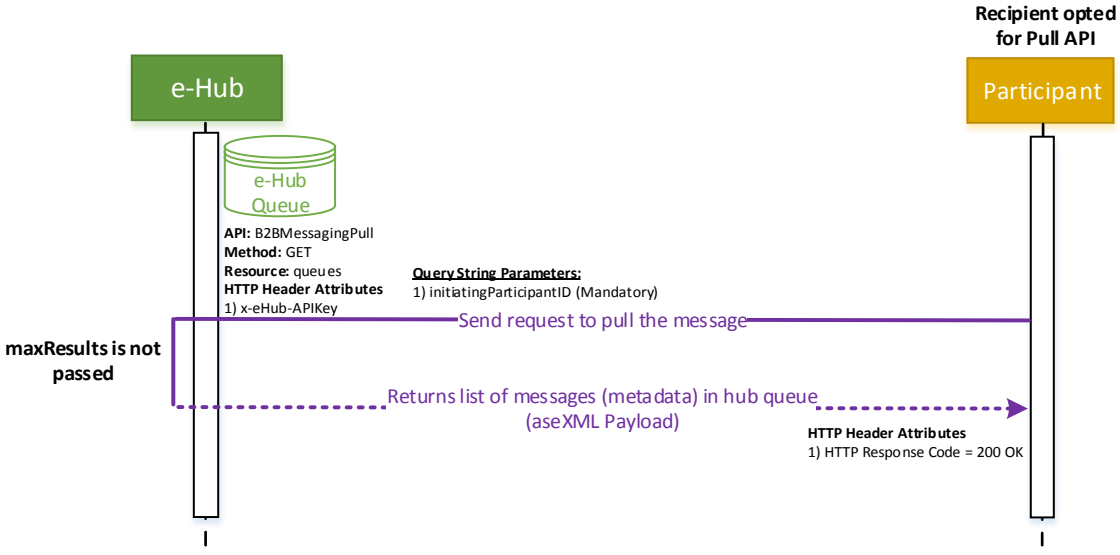
Figure 39 – Normal Processing (Empty queue) (pull)



- If there are no messages in the e-Hub queue to pull, the e-Hub will send HTTP response code of 404.

9.5.5 Normal processing (maxResults not passed) (Pull)

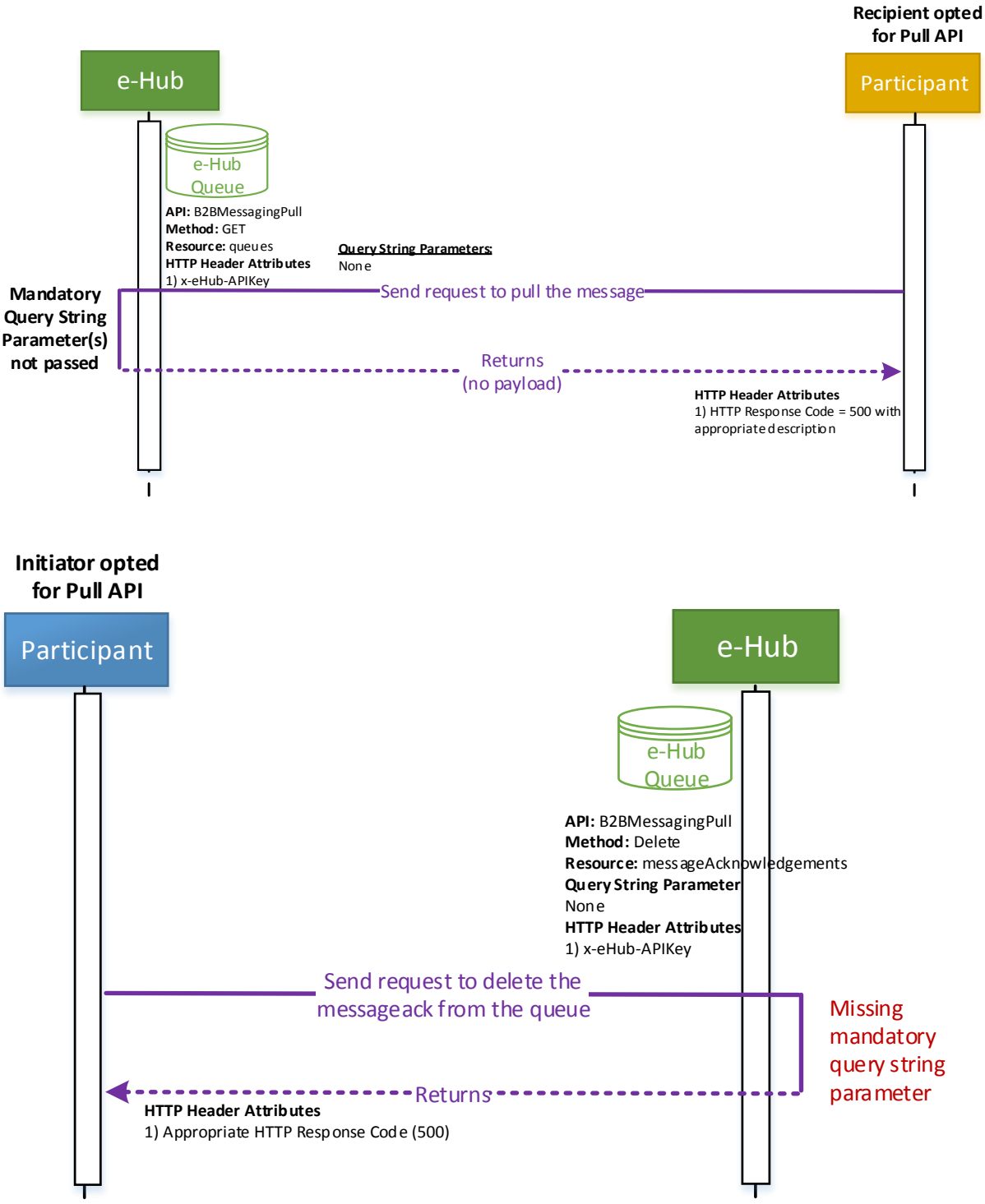
Figure 40 – Normal Processing (maxResults not passed) (pull)



- If participants call B2BMessagingPull API and resource '/queues' without passing the query string parameter 'maxResults', the e-Hub returns the metadata of messages in the e-Hub queue. The metadata of the messages is sent in an aseXML message; transaction type: HubQueueReport.

9.5.6 Invalid query string parameters (Pull)

Figure 41 – Invalid Query String Parameters (Recipient opted) (pull)

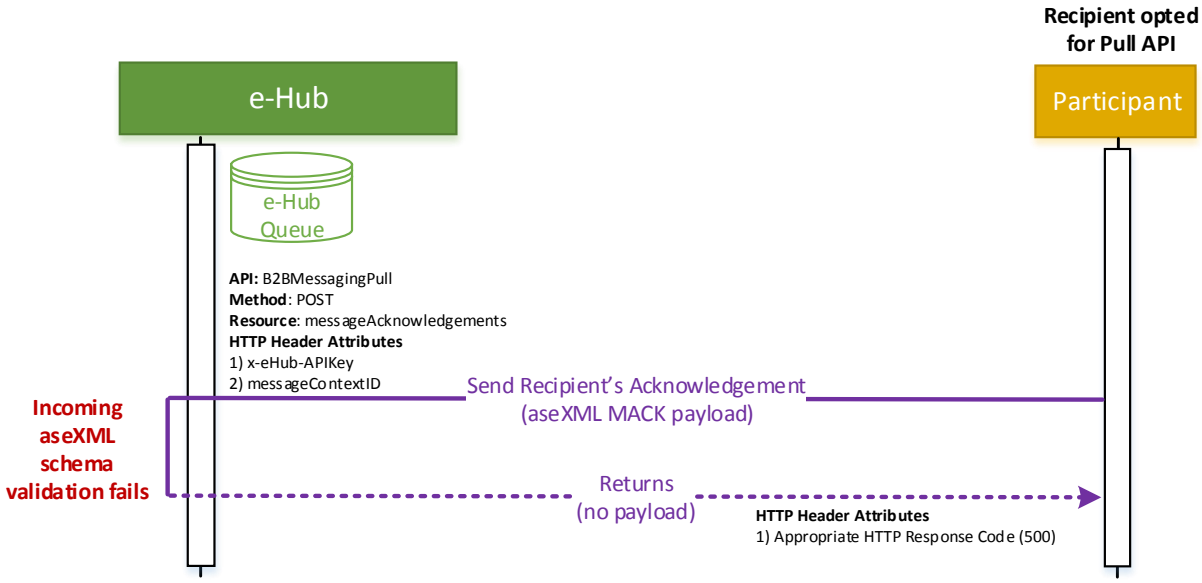


- If a mandatory query string parameter (initiatingParticipantID) is not passed when using the '/queues' or '/messageAcknowledgements' resource, return code of 500 with appropriate exception details is returned.

© AEMO 2017

9.5.7 Incoming aseXML schema validation failure (Pull)

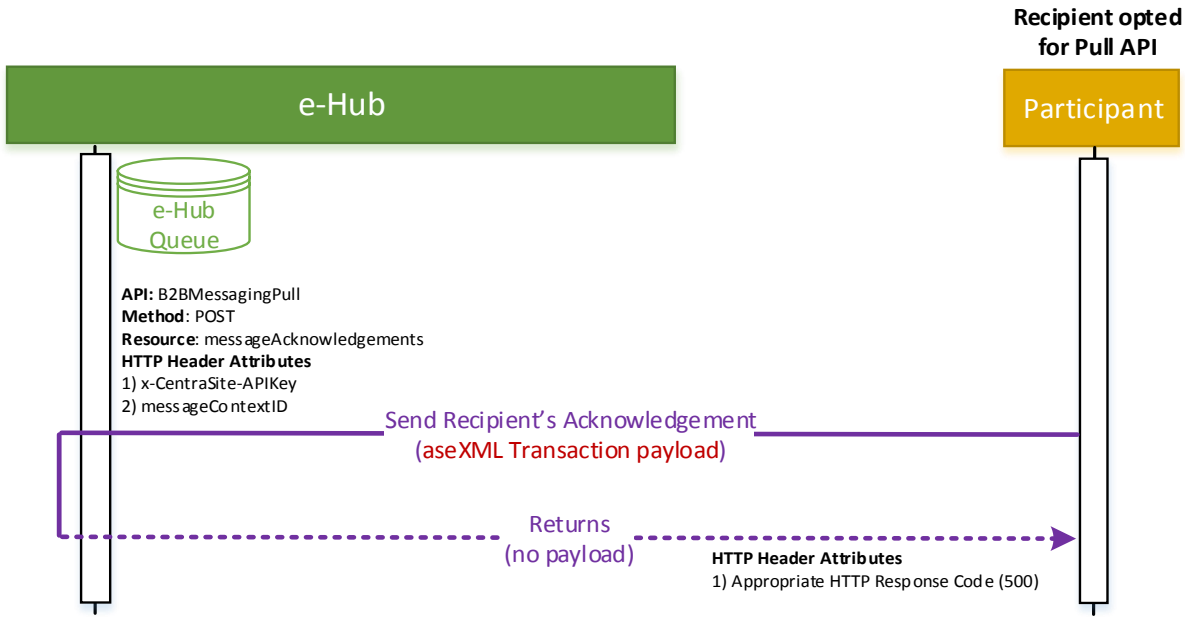
Figure 42 – Incoming aseXML Schema Validation Failure (pull)



- API: B2BMessagingPull, Resource: messageAcknowledgements - the e-Hub will return a HTTP response code of 500 with an exception description if the incoming payload is invalid.

9.5.8 Incorrect Payload (Pull)

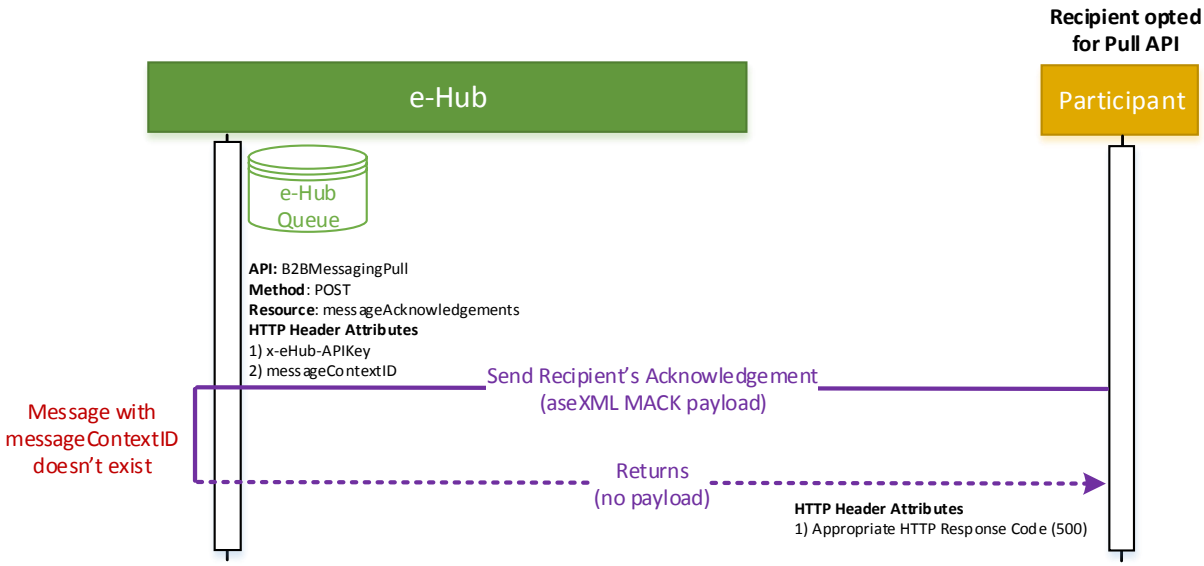
Figure 43 – Incorrect Payload (pull)



- API: B2BMessagingPull, Resource: messageAcknowledgements - the e-Hub will return a HTTP response code of 500 with an exception description if the incoming payload is invalid

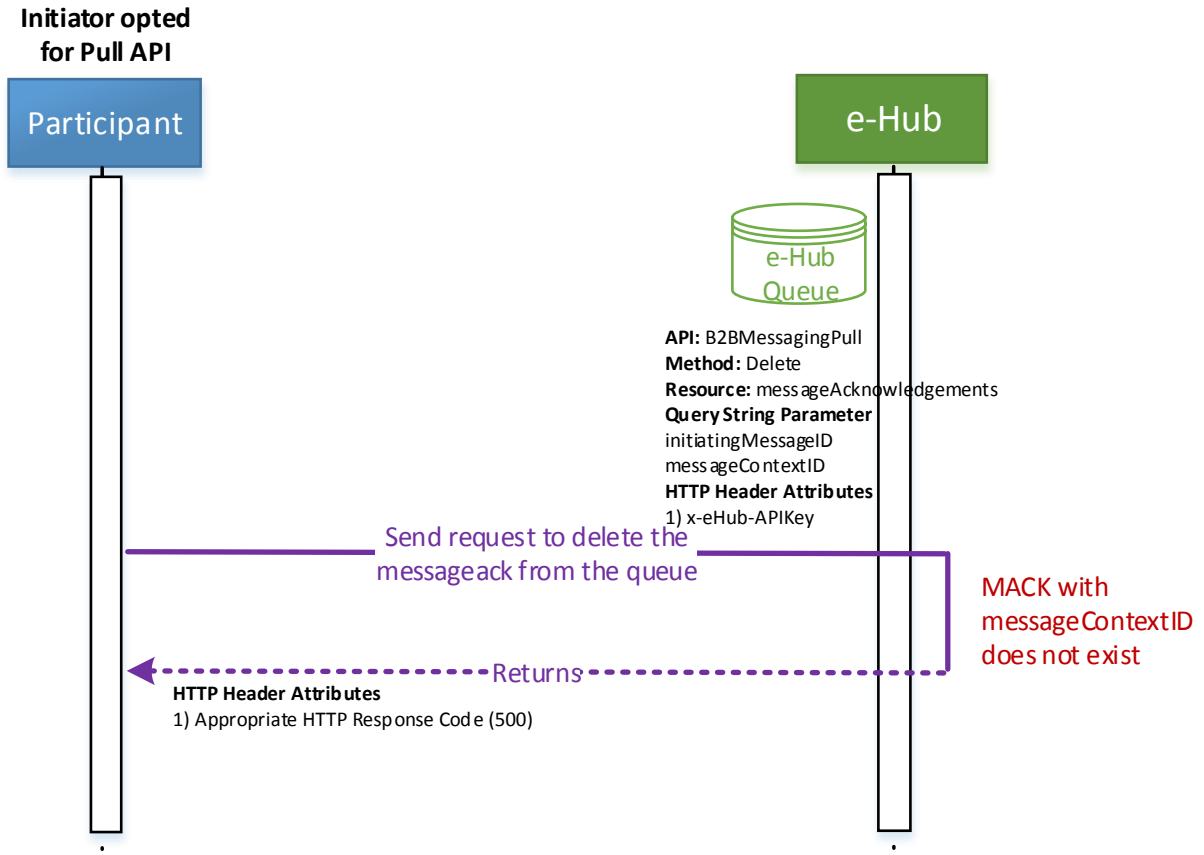
9.5.9 Message queue exception (Pull)

Figure 44 – Message Queue Exception (Recipient opted) (pull)



API: B2BMessagingPull, Resource: messageAcknowledgements, Method: POST - the e-Hub will return a HTTP response code of 500 with an exception description if the messageContextID does not exist in e-Hub queue for example, the message is acknowledged via the MSATS browser before the participant acknowledges using the API.

Figure 45 – Message Queue Exception (Initiator opted) (pull)



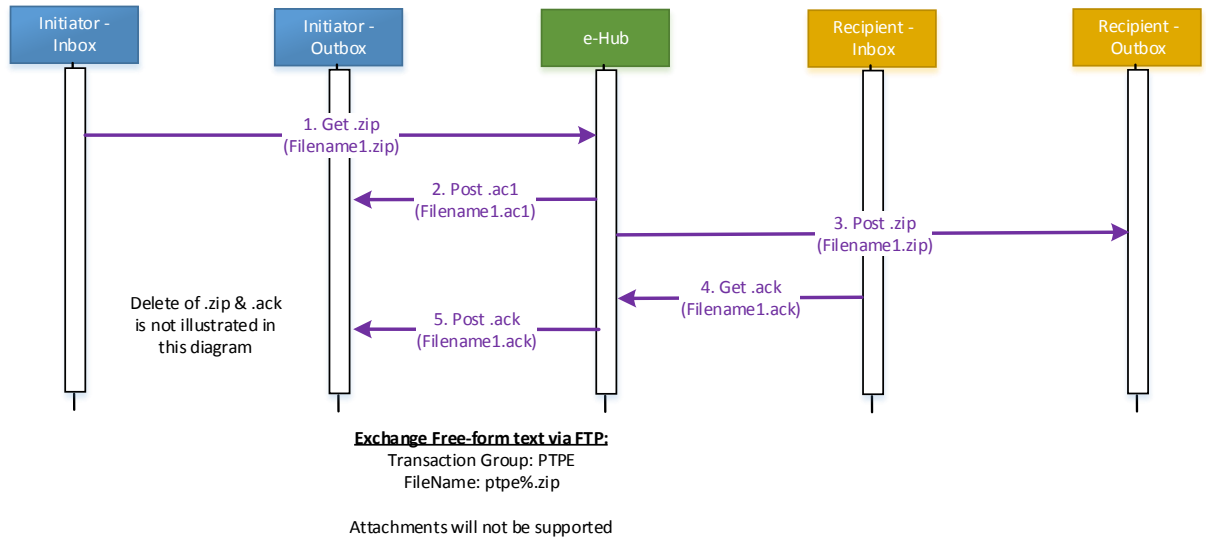
- API: B2BMessagingPull, Resource: messageAcknowledgements, Method: DELETE - the e-Hub will return a HTTP response code of 500 with an exception description if the messageContextID does not exist in e-Hub queue for example, the message has already been deleted via the MSATS browser

9.6 Peer-to-Peer (P2P) messaging

Free-form information can be exchanged using FTP protocol as illustrated below. The following transaction group and transaction type will be used in the aseXML file:

- Transaction Group: PTPE
- Transaction Type: <PTPDataExchange> & element <FreeFormData>

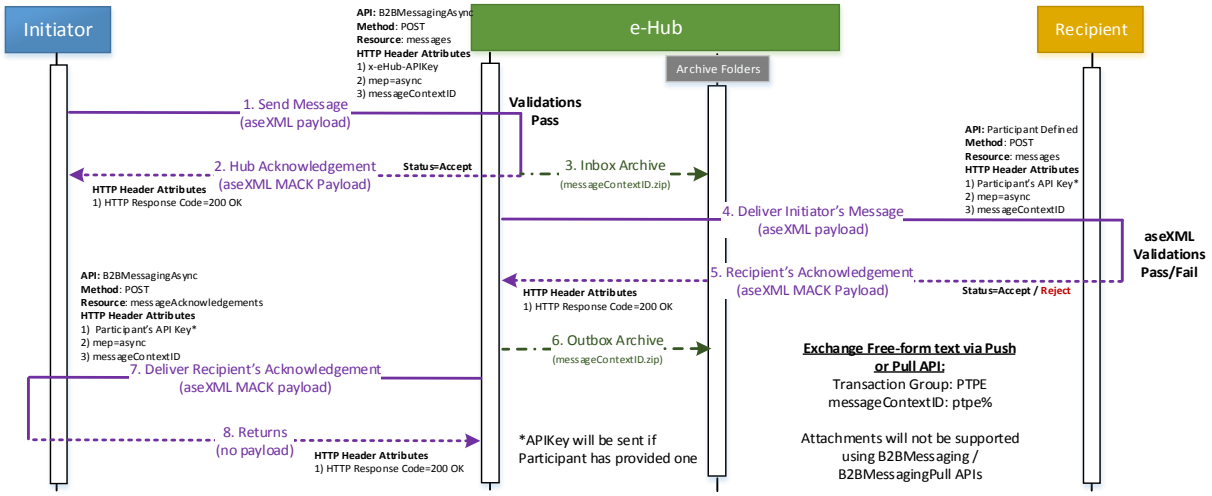
Figure 46 – Peer-to-Peer data exchange using FTP



Free-form information can be exchanged using B2BMessagingAsync (illustrated below), B2BMessagingSync and B2BMessagingPull API. The following transaction group and transaction type will be used in the aseXML file:

- Transaction Group: PTPE
- Transaction Type: <PTPDataExchange> & element <FreeFormData>

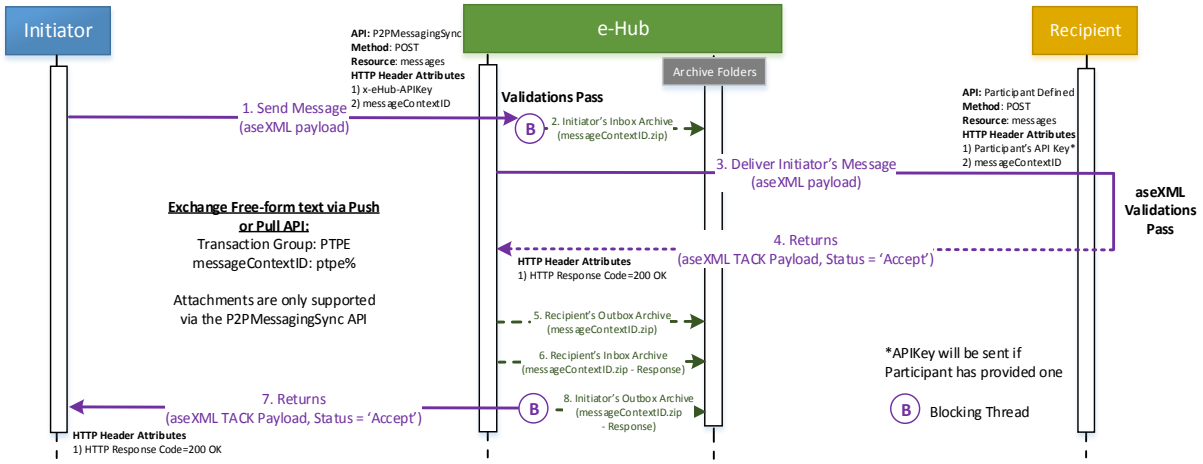
Figure 47 – Peer-to-Peer data exchange using B2BMessagingAsync API



Free-form information and/or attachments can be exchanged using the P2PMessagingSync API. The following section illustrates the scenarios related to P2PMessagingSync API.

9.6.1 Normal processing – scenario 1 (P2PMessaging)

Figure 48 – Peer-to-Peer data exchange using P2PMessagingSync API



1. The Initiator sends the aseXML message and/or attachments using the P2PMessagingSync API and resource '/messages'.
2. The e-Hub validates the incoming message and holds the initiating thread (blocking thread).
3. The e-Hub delivers the incoming message to the Recipient. The Recipient validates the incoming message and sends one of the following response:
    a. Message Acknowledgement, status = 'Reject' if the validations of the incoming message is invalid.
    b. Message Acknowledgement status = 'Accept' if the validations of the incoming message is valid and a Transaction Acknowledgement will be sent later.
    c. Transaction Acknowledgement, status = 'Reject' if the business validations of the incoming message fails.

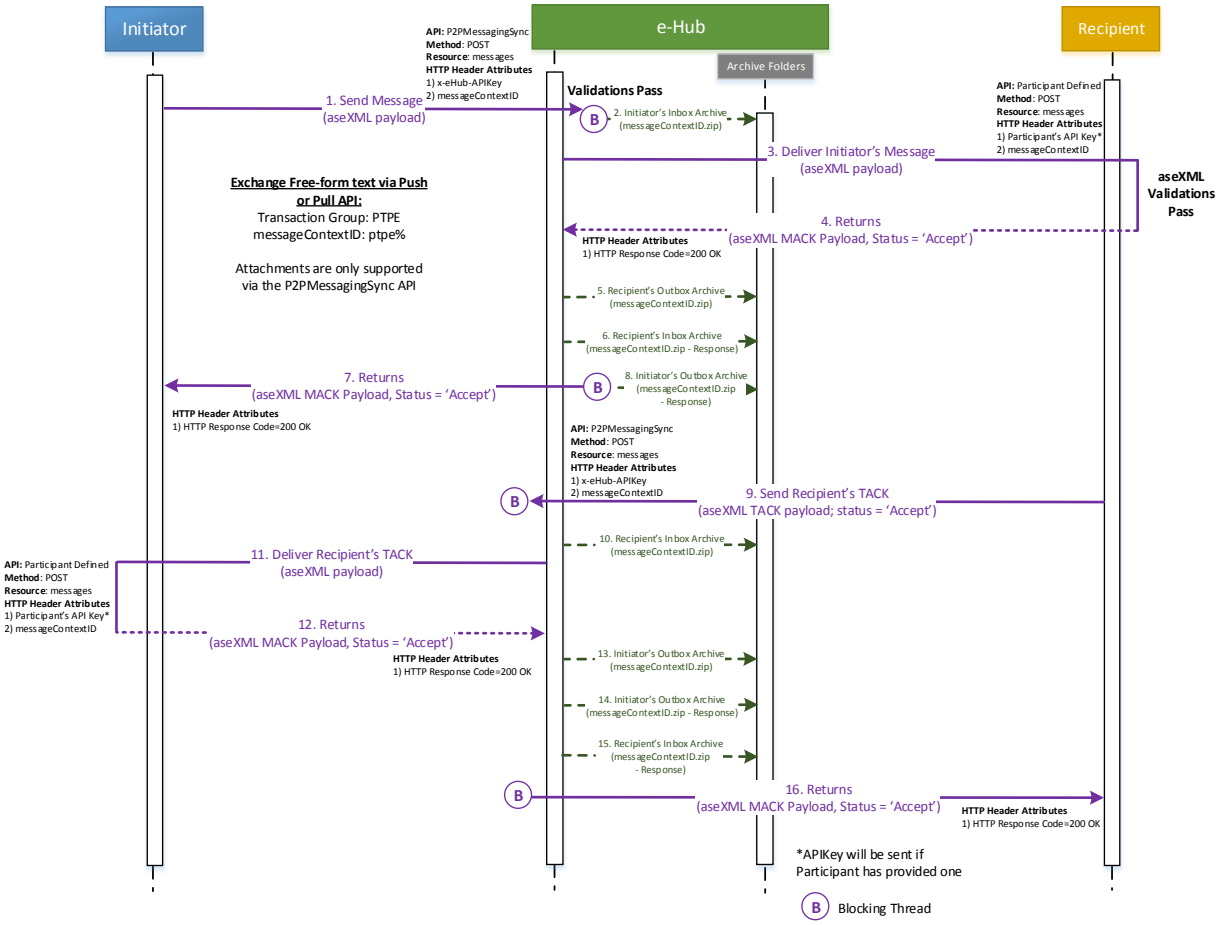
   d. Transaction Acknowledgement, status = ‘Accept’ if the business validations of the incoming message is valid.

4. The e-Hub responds to the blocking thread with the response received from the Recipient.
5. The incoming message and all the attachments are zipped and stored in (filename: messageContextID.zip).
   a. Initiator’s Inbox archive directory.
   b. Recipient’s Outbox archive directory.
6. The response message will be zipped and stored in (filename: messageContextID.zip):
   a. Recipient’s Inbox archive directory.
   b. Initiator’s Outbox archive directory.
7. If the Recipient sends attachments in the response, the e-Hub will not send those attachments to the Initiator nor will archive the attachments in the response message:
   a. If a participant calls P2PMessagingSync API to send attachments (for example, a .txt file and a .csv file), then the HTTP request must contain the following payloads.
   b. An aseXML message listing the attachments sent in the HTTP request (using Transaction Type: <PTPDataExchange> & element < AttachmentList>).
   c. Attachment#1: .txt file.
   d. Attachment#2: .csv file.
   e. The e-Hub will validate that the attachment list (names) in the HTTP request matches the attachment list in the aseXML file.

© AEMO 2017

9.6.2 Normal processing - scenario 2 (P2PMessaging)

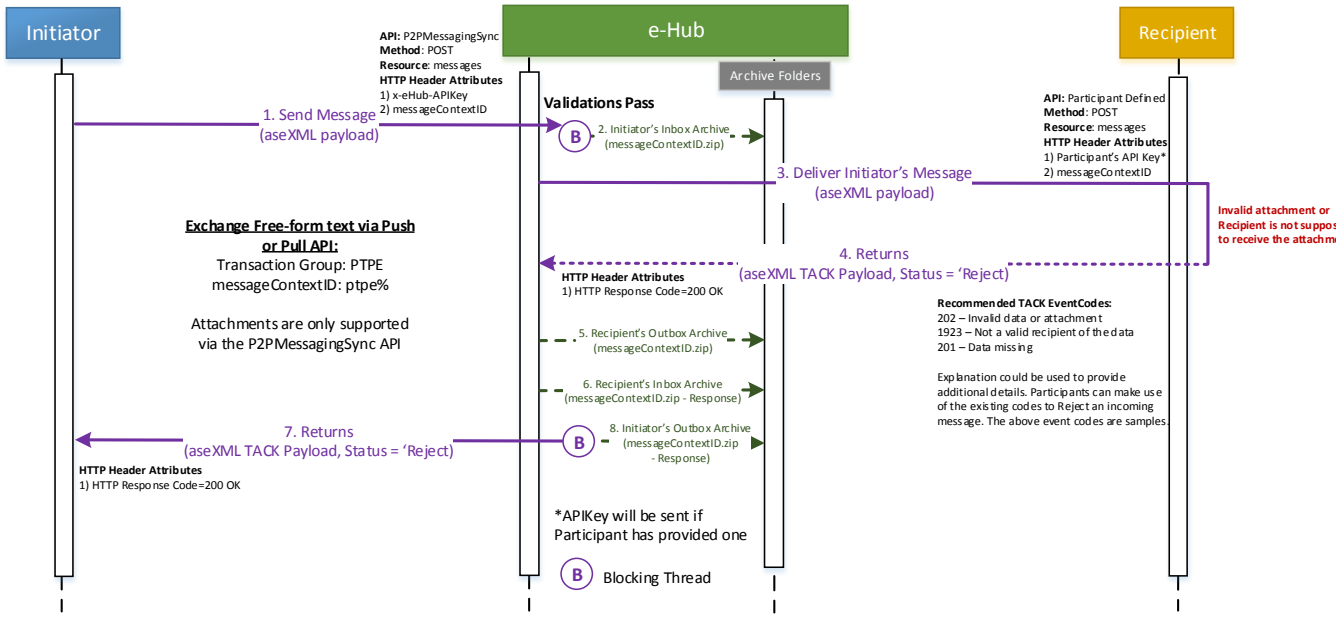
Figure 49 – Peer-to-Peer data exchange using P2PMessagingSync API – MACK accept & TACK accept



1. The response from the Recipient is MessageAcknowledgement (MACK), status = 'Accept'. If the participant is unable to send TransactionAcknowledgement (TACK) as a response to the incoming request, the participant can issue a MACK followed by a TACK
2. The Recipient sends MACK in response to the incoming P2P message. The Recipient then triggers another API call to send the corresponding TACK.

9.6.3 TACK status of Reject (P2PMessaging)

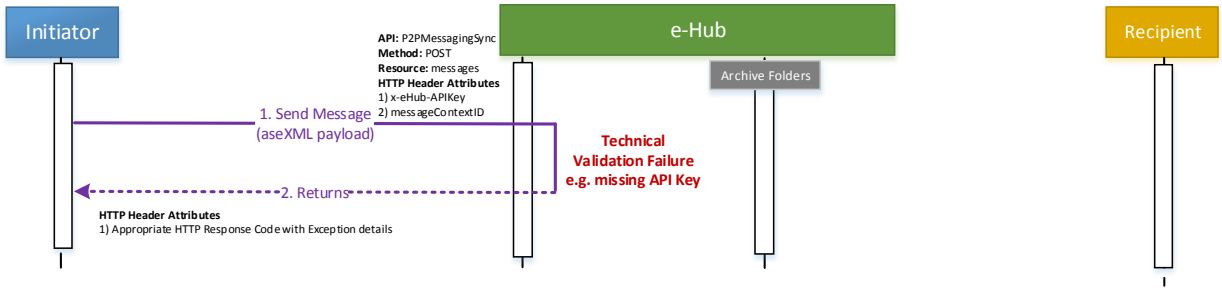
Figure 50 – Peer-to-Peer data exchange – TACK status of Reject



1. In the above scenario, the Recipient sends TACK message (status = Reject) for example, participant is not the intended recipient of the attachment(s).
2. The Recipient uses one of the existing TACK event codes and provides the details of the exception in aseXML <Explanation> element.

9.6.4 Technical validation failure (P2PMessaging)

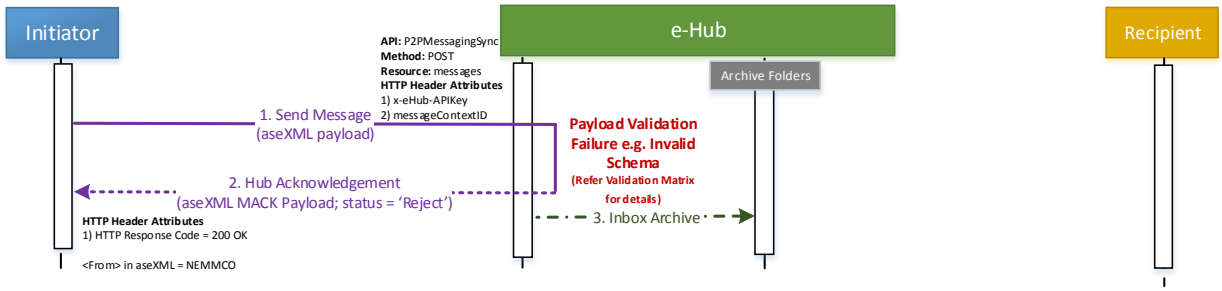
Figure 51 – Peer-to-Peer data exchange – Technical Validation Failure



In the event of a technical validation failure, the e-Hub returns a response with the appropriate HTTP Response Code and exception details in the response code description. The incoming message is not archived by the e-Hub.

9.6.5 Payload validation failure (P2PMessaging)

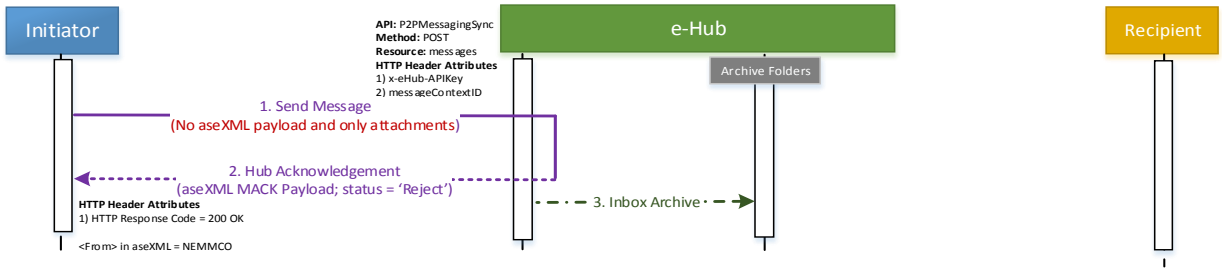
Figure 52 – Peer-to-Peer data exchange – Payload Validation Failure



1. Queuing of messages and protocol interoperability is not applicable to Peer-to-Peer messaging.
2. The e-Hub rejects the incoming messages if the Recipient is not registered for Peer-to-Peer services.

9.6.6 Request with only attachments (P2PMessaging)

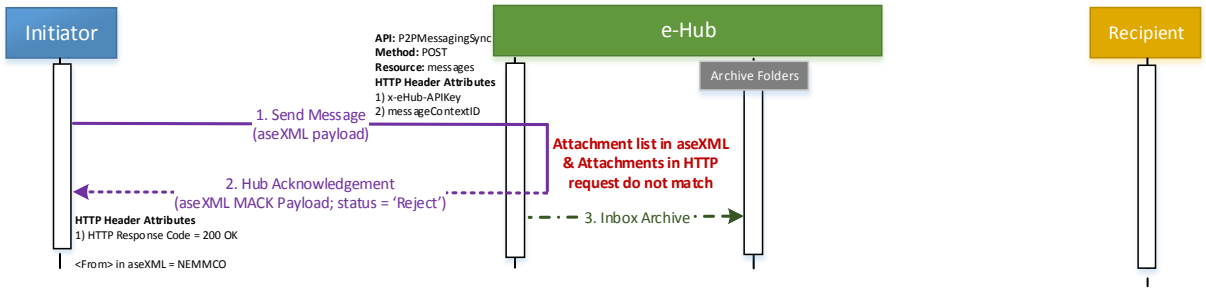
Figure 53 – Peer-to-Peer data exchange – Payload with only attachments



1. HTTP request contains only attachments and the aseXML payload is missing. The list of attachments are required to be sent in aseXML payload.
2. The e-Hub rejects the incoming messages if the HTTP request only carries attachments.

9.6.7 Attachment list in HTTP request & aseXML does not match (P2PMessaging)

Figure 54 – Peer-to-Peer data exchange – Attachment list mismatch

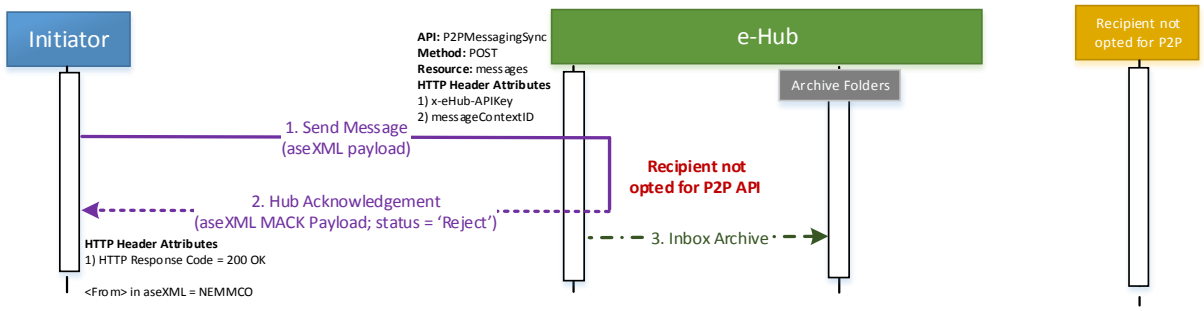


1. aseXML will contain the list of attachments that are sent in the HTTP request

2. The e-Hub rejects the incoming messages if the attachments (list of attachment names) in the HTTP request does not match the attachment list in the aseXML message

9.6.8 Recipient on FTP/not opted for P2P services (P2PMessaging)

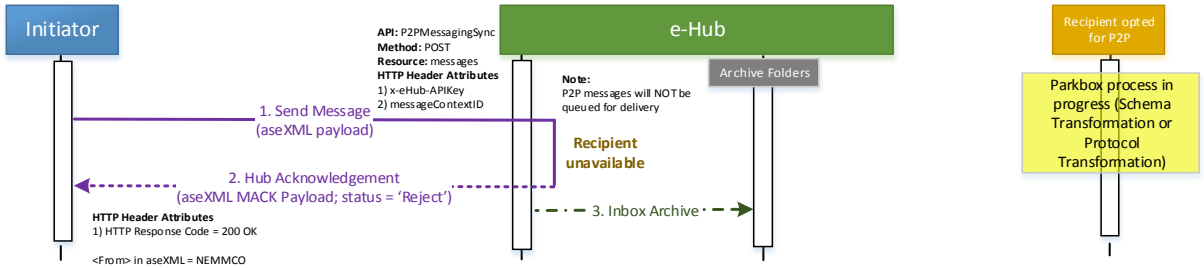
Figure 55 – Peer-to-Peer data exchange – Recipient not opted for P2P Services



- e-Hub rejects the incoming message if the Recipient has not registered for P2P services

9.6.9 Recipient unavailable – Parkbox (P2PMessaging)

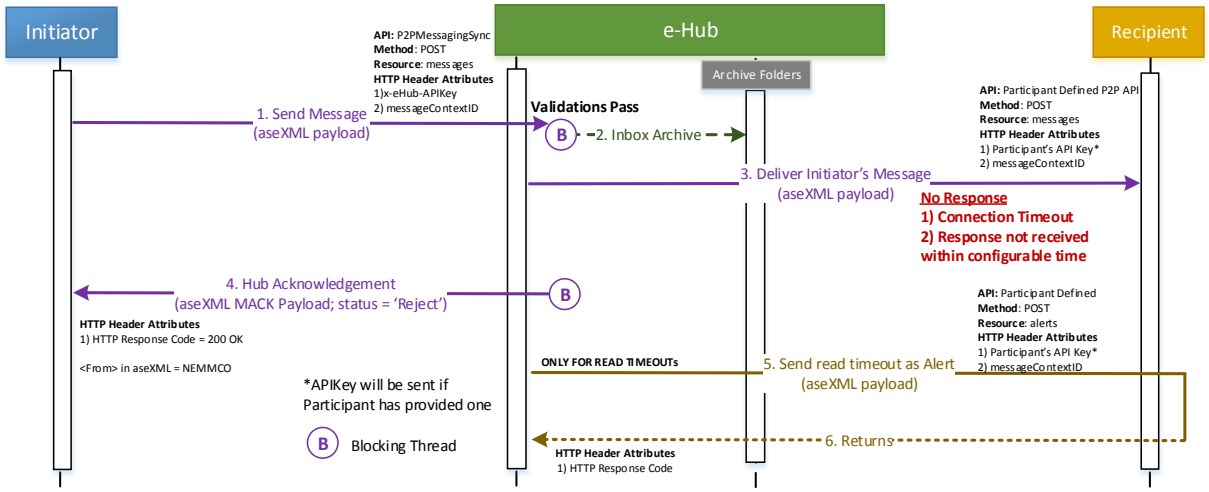
Figure 56 – Peer-to-Peer data exchange – Recipient Unavailable



1. Definition of parkbox process: The participant is in the process of updating their incoming version of the aseXML message (or) the opted protocol (FTP or API) using the B2B Transform Screen
2. If the Recipient is unavailable the message is rejected. Queuing is not applicable to P2P messaging.
3. It will be up to the Initiator and Recipient to resolve any issues and/or establish workarounds until the Recipient's system becomes available.

9.6.10 Recipient connection timeout (P2PMessaging)

Figure 57 – Peer-to-Peer data exchange – Recipient Connection Timeout



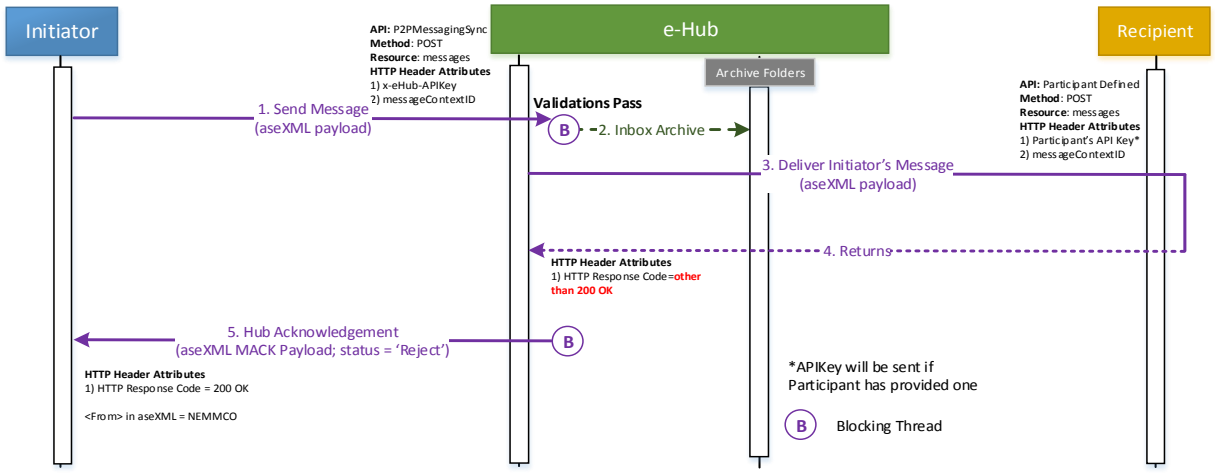
Definition:

1. Connection timeout: e-Hub is unable to connect to the Recipient's endpoint
2. Read timeout: the e-Hub is connected to Recipient's endpoint but the Recipient does not respond within the configured time

Rules:

1. In the event of a connection timeout or read timeout scenario, the e-Hub responds to the blocking thread (to the Initiator) with the appropriate exception details (i.e. appropriate event code) in aseXML message
2. In the event of read timeout, the e-Hub sends an alert to the Recipient stating that the connection has been terminated. The e-Hub will send this alert by calling the participant's registered hub message management API and using the '/alerts' resource. The alert will be sent in the aseXML message; transaction type: PayloadExceptionAlert
3. The Initiator and Recipient must resolve issues related to read timeouts i.e. determine the next steps and mechanism to re-deliver the response message.
4. Recipient HTTPS Rejection (P2PMessaging)

Figure 58 – Peer-to-Peer data exchange – Recipient HTTP Rejection



- If the e-Hub receives a HTTP response code other than '200 OK' from the Recipient, the e-Hub responds to the blocking thread (to the Initiator) with the negative hub acknowledgement i.e. aseXML message NACK payload (refer section 10 for details on the event code and description).

10 VALIDATIONS

See the attached draft validation spreadsheet included with this document.

© AEMO 2017

11 API SPECIFIC ASEXML PAYLOADS

A New Transaction Group HMGT – Hub Management is introduced to support message exchange between the e-Hub and participants related to the API protocol for example, participants requesting the e-Hub to send the list of current B2B stop files.

11.1 HubQueueReport

Participants can view metadata of the messages stored in the e-Hub queue. The metadata of the messages is made available to the participants via the HubQueueReport transaction type.

Table 78 – HubQueueReport

Field	Format	Use	Definition
Parameter Name	Varchar(100)	R	The query string parameter name sent in the HTTP request. Provides information to the participant on the list of query string parameters that were sent in the request. If there are no query string parameters in the incoming request (GET method), this field will not be populated. This field repeats for each query string parameter passed in the participant request. e.g. initiatingParticipantID
Parameter Value	Varchar(100)	R	The query string parameter value sent in the HTTP request. Provides information to the participant on the list of query string parameters that were sent in the request. If there are no query string parameters in the incoming request (GET method), this field will not be populated. This field repeats for each query string parameter passed in the participant request. e.g. 'AUSPG'
ResultCount	Number	M	The number of records returned in the list. If there are no messages in the e-Hub queue to report, ResultCount will be set to 0.
TransactionGroup	Varchar	R	Transaction Group of the original message in the e-Hub queue e.g. 'SORD'. If the message in the e-Hub queue does not contain 'TransactionGroup' element in the header, this field will not be supplied (e.g. event-only MACKs) This field repeats for each message in the queue.
Priority	Varchar	O	Priority of the message in the e-Hub queue. Allowed values: ▪ High ▪ Medium ▪ Low This field repeats for each message in the queue. If the message in the e-Hub queue does not contain 'Priority' element in the header, this field will not be supplied (e.g. event-only MACKs)
FromParticipantID	Varchar(10)	R	ID of the participant that sent the original message. This field repeats for each message in the queue. If the message in the e-Hub queue does not contain 'From' element in the header, this field will not be supplied (e.g. event-only MACKs)
MessageID	Varchar(36)	R	MessageID of the original message in the e-Hub queue. This field repeats for each message in the queue. If the message in the e-Hub queue does not contain 'MessageID' element in the header, this field will not be supplied (e.g. event-only MACKs)

Field	Format	Use	Definition
InitiatingMessageID	Varchar(36)	R	InitiatingMessageID if populated in the message stored in e-Hub queue This field repeats for each message in the queue. If the message in the e-Hub queue does not contain 'InitiatingMessageID' attribute in the <MessageAcknowledgement> element, this field will not be supplied (e.g. event-only MACKs)
MessageType	Varchar(50)	M	Type of message in the e-Hub queue Allowed values: ▪ Transaction Message ▪ Transaction Acknowledgement ▪ Message Acknowledgement If the message contains at least one MessageAcknowledgement element or is an event-only MACK ▪ Message Acknowledgement If the message contains TransactionAcknowledgement element but no MessageAcknowledgement element ▪ Transaction Acknowledgement If the message contains <Transactions> element ▪ Transaction Message This field repeats for each message in the queue.
MessageContextID	Varchar(50)	M	messageContextID of the messages in the e-Hub queue This field repeats for each message in the queue.
ReceivedDateTime	DateTime	M	Date and time that the original message was received by the e-Hub. This field repeats for each message in the queue.

11.2 HubFlowControlReport

Participants can retrieve the current standing list of stop files from the e-Hub. The stop file list will be made available to the participants via the HubFlowControlReport transaction type.

Table 79 – HubQueueReport

Field	Format	Use	Definition
ParameterName	Varchar(100)	R	The query string parameter name sent in the HTTP request. Provides information to the participant on the list of query string parameters that were sent in the request. If there are no query string parameters in the incoming request (GET method), this field will not be populated. This field repeats for each query string parameter passed in the participant request. e.g. initiatingParticipantID
ParameterValue	Varchar(100)	R	The query string parameter value sent in the HTTP request. Provides information to the participant on the list of query string parameters that were sent in the request. If there are no query string parameters in the incoming request (GET method), this field will not be populated. This field repeats for each query string parameter passed in the participant request. e.g. 'AUSPG'
ResultCount	Number	M	The number of records returned in the list. If there are no stop files to report, ResultCount will be set to 0.

Field	Format	Use	Definition
AlertType	Varchar(50)	M	Possible values: B2BStopFile This field has been added for future use. If alertType is not passed in the HTTP request, e-Hub will default to 'B2BStopFile' This field repeats for each of the alerts reported
ParticipantID	Varchar(10)	M	ID of the affected participant i.e. participant to whom stop file is issued This field repeats for each of the alerts reported
StopFileName	Varchar(80)	O	Filename of the Stop File. This field repeats for each of the alerts reported
Cause	Varchar(80)	M	Reason for the stop file e.g. 'Too many unacknowledged files/messages', 'Exceed high water mark' This field repeats for each of the alerts reported
StopDateTime	DateTime	M	Date and time the Stop File was created This field repeats for each of the alerts reported

11.3 HubFlowControlAlertNotification

The e-Hub issues alert notifications to the participants opted-in for APIs (for example, when a stop file is issued / removed for a participant) using the HubFlowControlAlertNotification transaction type.

Table 80 – HubQueueReport

Field	Format	Use	Definition
AlertType	Varchar(50)	M	Possible values: B2BStopFile This field has been added for future use.
ActionType	Varchar	M	Allowed values: ADD – When a new stop file is created REMOVE – When an existing stop file is removed
ParticipantID	Varchar(10)	M	ID of the affected participant i.e. participant against whom stop file is issued
StopFileName	Varchar(80)	O	Filename of the Stop File.
Cause	Varchar(80)	M	Reason for the stop file being added or removed, e.g. 'Too many unacknowledged files/messages', 'Exceed high water mark'
StopDateTime	DATETIME	R	Date and time the Stop File was created Populated when ActionType = 'ADD'

Field	Format	Use	Definition
RemovedDateTime	DATETIME	R	Date and time the Stop File was removed Populated when ActionType = 'REMOVE'

11.4 PayloadExceptionAlert

If the response payload sent by a participant fails validations, the validation failure is notified to the participant using the '/alerts' resource. The payload validation failure is notified using aseXML payload, transaction type 'PayloadExceptionAlert'.

Table 81 – HubQueueReport

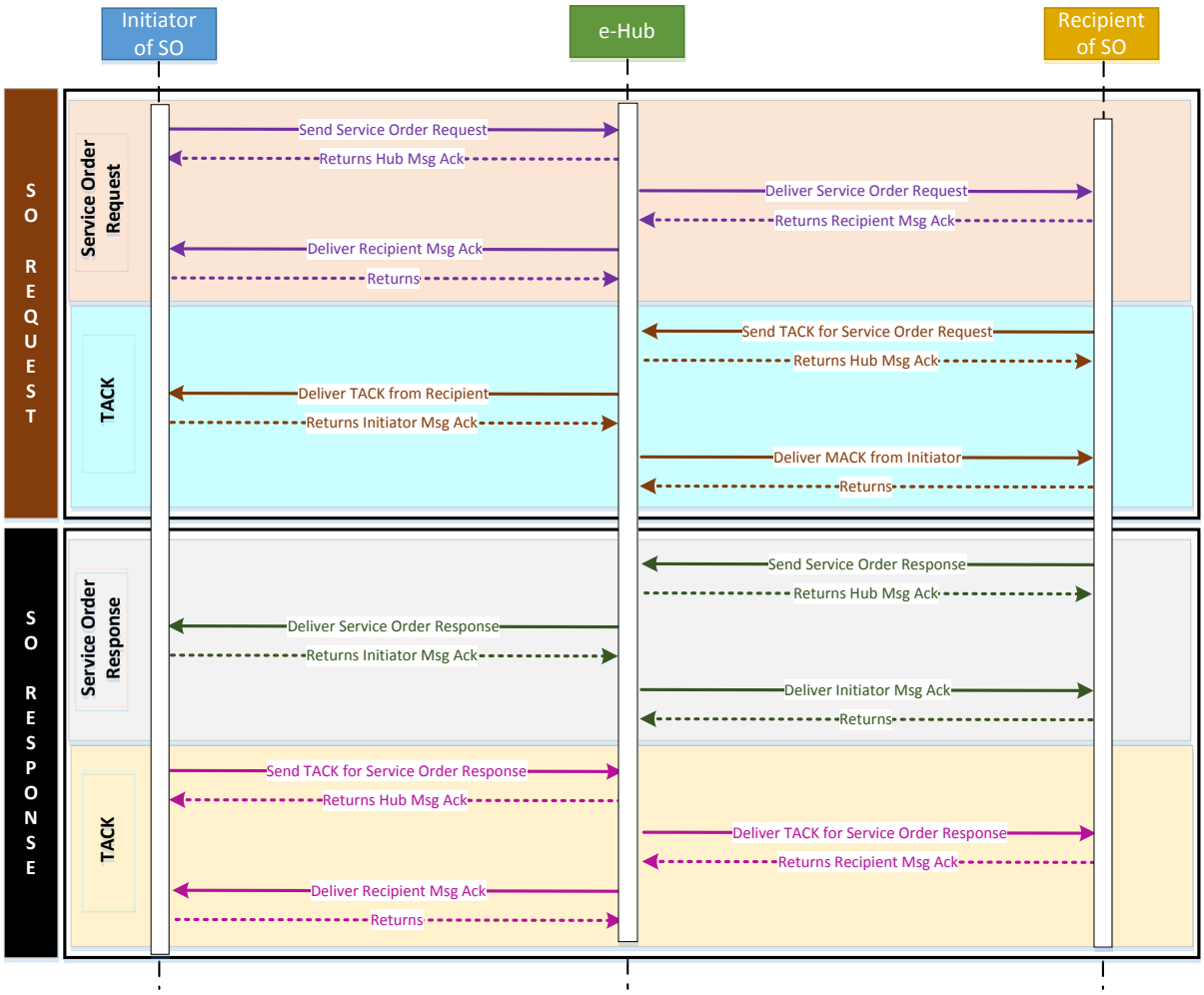
Field	Format	Use	Definition
Name	Varchar(80)	O	Name of the participant's API where the response payload validation failure was encountered
Resource	Varchar(80)	O	Name of the participant's API resource where the response payload validation failure was encountered
Method	Varchar(10)	O	Name of the participant's API method used where the response payload validation failure was encountered Possible values: ▪ GET ▪ POST ▪ DELETE ▪ PUT
MessageType	Varchar(50)	M	Allowed values: ▪ Transaction Message ▪ Transaction Acknowledgement ▪ Message Acknowledgement If the message contains at least one MessageAcknowledgement element or is an event-only MACK ▪ Message Acknowledgement If the message contains TransactionAcknowledgement element but no MessageAcknowledgement element ▪ Transaction Acknowledgement If the message contains <Transactions> element ▪ Transaction Message
MessageInitiator	Varchar(10)	M	ID of the participant that sent the erroneous message/transaction.
MessageID	Varchar(36)	R	The MessageID of the erroneous message/ transaction. If the response message does not contain MessageID (e.g. event only MACK) or the schema of the response message is invalid, this field will not be populated.
InitiatingMessageID	Varchar(36)	R	The InitiatingMessageID on the erroneous message. If the response message does not contain InitiatingMessageID (e.g. Transaction Message) or the schema of the response message is invalid, this field will not be populated.
MessageContextID	Varchar(50)	M	The MessageContextID of the erroneous message/ transaction.

Field	Format	Use	Definition
AlertDateTime	DateTime	M	Date and time the transaction was raised.
EventCode	Numeric(4)	M	Event Code as per validation table in section 10
Explanation	UNLIMITED VARCHAR	M	Details of the event

APPENDIX A. ASYNC SORD EXAMPLE

Example - Asynchronous message exchange model using APIs – Service Order process

Figure 59 – Asynchronous message exchange – Service Order Process (example)



APPENDIX B. E-HUB SYSTEM ADDRESSES

e-Hub - MSATS Browser and FTP

The MSATS systems can only be accessed over MarketNet.

System	Pre-production	Production
MSATS Browser	https://msats.preprod.nemnet.net.au/msats	https://msats.prod.nemnet.net.au/msats
Participant File Server	ftp://146.178.211.225	ftp://146.178.211.205

e-Hub - API Gateway and Portal

The e-Hub API systems can be accessed over the internet or MarketNet.

System	Pre-production	Production
API Portal	Internet https://apiportal.preprod.aemo.com.au MarketNet https://apiportal.preprod.marketnet.net.au	Internet https://apiportal.prod.aemo.com.au MarketNet https://apiportal.prod.marketnet.net.au
API Gateway	Internet https://apis.preprod.aemo.com.au:9319 MarketNet https://apis.preprod.marketnet.net.au:9319	Internet https://apis.prod.aemo.com.au:9319 MarketNet https://apis.prod.marketnet.net.au:9319

© AEMO 2017

APPENDIX C. OBTAINING AN SSL CERTIFICATE

This appendix outlines the process to create and obtain the required SSL certificates to communicate with AEMO's e-Hub API's.

Step 1. Determine whether to create a new certificate or use an existing certificate

If the participant is a *new* e-HUB participant or an *existing electricity* participant requesting a certificate:

a. The participant can choose to have one certificate **for multiple** participant ID's OR,

b. The participant can choose to have one certificate **for each** participant ID

If the participant is an *existing* Gas Hub participant:

a. The participant can request to use the *existing* Gas Certificate **for multiple** e-HUB participant ID's OR,

b. The participant can request to use the *existing* Gas Certificate **for each** e-HUB participant ID OR,

c. The participant can choose to have one *new certificate* **for multiple** participant ID's OR,

d. The participant can choose to have one *new certificate* **for each** participant ID

Step 2. Using an existing certificate

If creating a new certificate proceed to Step 3.

Participant sends an email to the AEMO Support Hub with the following information:

To:	supporthub@aemo.com.au
From:	The registered email address for requesting SSL certificates and API Keys. This email address is determined as part of registration process for e-Hub API services.
Subject:	AEMO HUB Participant ID Update
Body:	Dear AEMO Support Hub, Please update the following participant ID's with a previously generated certificate. The certificate to be used has a Thumbprint of <Thumbprint>. This certificate must be used for the following Participant ID's: - <Participant ID 1> - <Participant ID 2> - <Participant ID 3> *One or more Participant ID's may be listed.*

© AEMO 2017

Step 3. Generate a CSR (to request a matching certificate from AEMO)

If using an existing certificate proceed to Step 6.

Obtaining SSL certificates is achieved by submitting a Certificate-Signing-Request (CSR) to AEMO who will then manage certificate generation and distribution.

To generate a new CSR the participant will need to know their Organisation ID (e.g. 30187). This will be provided to the participant as part of the registration/accreditation process.

The CSR must not have the challenge password set.

The CSR should be generated with the following information, and saved to a file:

CN (common name)	<ORGANISATION ID>_<PreProd | Prod>_<Month><Year> e.g. 30187_PreProd_Jun2017 The length of the CN field is limited to 64 characters
OU	The OU and remaining fields in the subject attribute can be any description relating to your organisation
Public key algorithm	Must use RSA encryption with a key length of 1024 bits or preferably 2048 bits
Signature algorithm	Must use RSA encryption with a SHA1 hash

Step 4. Send the CSR to AEMO

The CSR file must be emailed to the AEMO Support Hub with the following information:

To:	supporthub@aemo.com.au
From:	The registered email address for requesting SSL certificates and API Keys. *This email address is determined as part of registration process for e-Hub API services.*
Subject:	AEMO CSR Request
Attachment:	CSR file

© AEMO 2017

Body:	Dear AEMO Support Hub, Please find attached a certificate signing request (CSR) to be signed by the AEMO Certificate Authority. This certificate must be used for the following Participant ID's: - <Participant ID 1> - <Participant ID 2> - <Participant ID 3> *One or more Participant ID's may be listed.*

Step 5. AEMO validates, generates and issues the certificate

AEMO will:

- Validate the CSR received from the participant
- Generate the SSL certificate by using the CSR
- Send the following SSL certificate chain to participant via email:
  - Participants' public certificate
  - e-Hub (production and pre-production) public certificate
  - CA certificate

Step 6. AEMO and Participant install the digital certificates

AEMO will apply the respective SSL certificate to one or more participant profiles in the e-Hub API Gateway.

The participant will apply the respective SSL certificate in their own API Gateway or system.

© AEMO 2017

APPENDIX D. OBTAINING AN API KEY

This appendix outlines the process to obtain an API Key required to be able to use the e-Hub market API's.

To be able to request API Keys participants must have the e-Hub API Portal login credentials provided by AEMO as part of the registration process.

Step 1. Login to the pre-production or production e-Hub API Portal

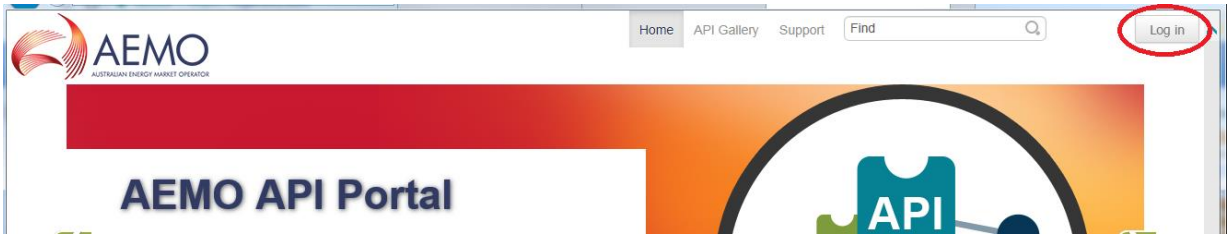
1. Access the AEMO e-Hub API Portal

API Keys are requested from AEMO's e-Hub API Portal. This can be accessed over the internet or from AEMO's MarketNet.

Pre-production	Internet https://apiportal.preprod.aemo.com.au MarketNet https://apiportal.preprod.marketnet.net.au
Production	Internet https://apiportal.prod.aemo.com.au MarketNet https://apiportal.prod.marketnet.net.au

2. Login using your AEMO issued credentials

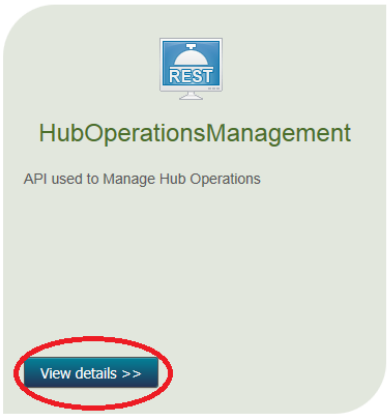
Click the "Log in" button on the portal.



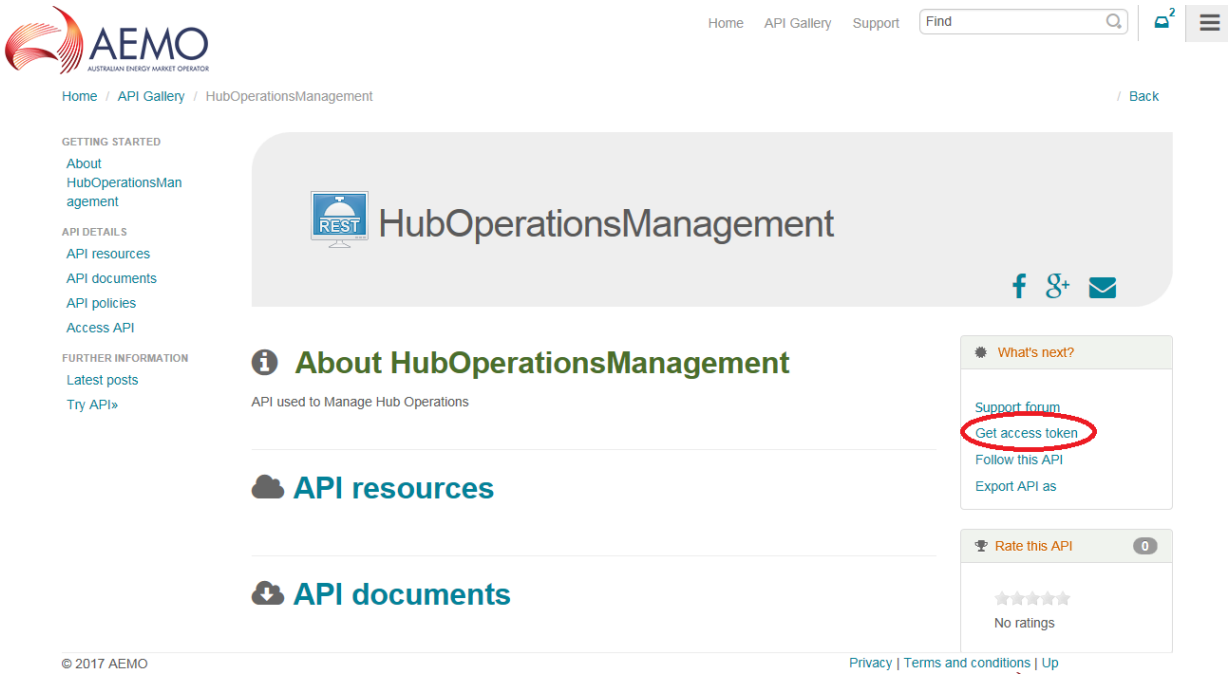
Step 2. Request the API Key

1. Click on the “API Gallery” menu item.

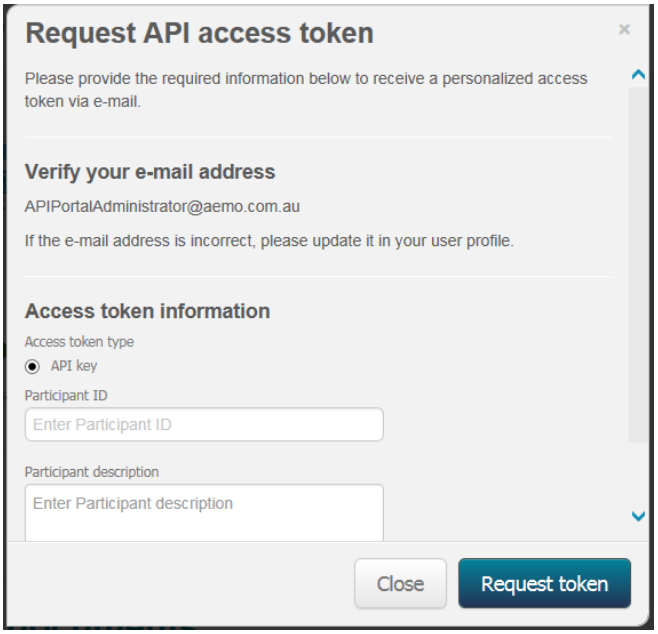
2. Click on the “View details >>” button for the API you would like a key for:



3. On the left hand “What’s next?” menu click the “Get access token” link



4. Complete the “Request API access token form”



Enter the following information:

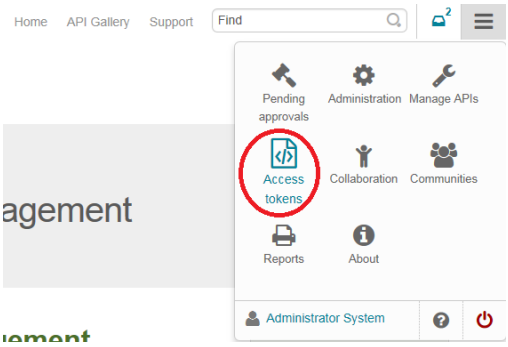
Participant ID	The registered Participant ID (PID) you require an API key for. This should be in uppercase. *AEMO will validate the request when it is received to ensure the account requesting the key is registered to do so.*
Participant description	This field is optional.

Step 3. Receiving the API Key

AEMO will send an email confirming the issuing or rejection in response to an API Key request to the registered email address of the e-Hub API Portal user.

Once approved and issued the API Keys will be able to be viewed in the API Portal.

1. Click the upper left menu icon
2. Click on the “Access tokens” link



The API Keys will be listed by Participant ID and then the API they are for.

The example Participant ID’s shown below are ACTEWNWO and NAGMO.

Home / Access tokens

Home API Gallery Support Find

Back

ACCESS TOKENS

ACTEWNWO

NAGMO

Access tokens

Manage access tokens

ACTEWNWO

API name	Access token	Type	Expiration date	Possible actions
HubMessageManagement	[redacted]	API key		Delete

NAGMO

API name	Access token	Type	Expiration date	Possible actions
HubMessageManagement	[redacted]	API key		Delete