TOTSCo

TOTSCo API Specification

A guide for developers

Version 2.0

TOTSCO

Contents

Figures

1 Introduction

The TOTSCo Hub API Specifications document contains the Letterbox and Directory API specifications as well as the definition of the message envelope structure. It is designed as a standard reusable interface; to facilitate a Hub and spoke message distribution framework; to support the adoption of near real time message processing and guaranteed message delivery. It also contains the security implementation for connecting to the TOTSCo Hub. This document is agnostic of any particular process documentation.

The intended audience of this document isrepresentatives of users who are responsible for the technical implementation of the communication between that user and the TOTSCo Hub.

1.1 Section intentionally left blank

1.2 Change log

Version **Date** **Changed By**	**Reason for change**
v1.0 20/07/2023 HUB design team	API Specification for the TOTSCo Hub. Issue 1.0 version
V1.1 18/08/2023	Updates to the below section - Section 2.1.8 Letter box responses - Section 2.2 Directory API Specification Section 3 – Security implementation. Inclusion of mutual TLS, API Key security options
V1.1a 08/11/2023	1) Section 3.2.2 API Key - Outbound Communication from Hub to RCP: Length of API Key supported for CP letterbox endpoint is 5000 characters. 2) section 2.1.8 Letterbox responses - Synchronous error response a) Inclusion of sample error response for 'Unclassified Authentication failure' sent for invalid API key. This was missed out in previous version. b) Internal error code 900902 mistakenly mentioned as 900900 in sample response for 'Invalid Credentials'. 3) Section 2.1.2 - URI format version number format change to keep in line with the actual URI links.

Version Date Changed By	Reason for change
V1.1a to v2.0	1) Multiple sections of the document removed or shortened to make the document more of a direct specification as the removed content was expositional and explanatory 2) References to RCP/MAP changed to more generic term of user throughout 3) Section 2.1.4 on version control moved to 2.3 as it applies to both letterbox and directory APIs 4) 2.1.5 clarification made that message delivery failures will not contain a source correlation ID, see 2.1.8.2 for more 5) 2.1.5 clarification made that it is best practice for correlation IDs to be unique though ultimately is the decision of the sender if not specified in the applicable industry process 6) 2.1.3 Letterbox version updated from v1 to v2. No functional differences apply beyond the addition of routingIDs not permissible in v1 and the error codes at point 9 below. 7) 2.1.8.1 updated to remove repetitive text and universal http response definitions. Process flow has been updated to validation process and all relevant error codes. 8) 2.1.8.1 Codes 403/9003 and 403/9001 corrected to say RCPID rather than RCPD ID. Document correction only, the errorText always stated RCPID. 9) 2.1.8.1 Error code list updated to include code 400/9017. The 256kb message size validation exists in v1, but the 400/400 error will be returned rather than 400/9017 10) 2.1.8.1 Code 401/900901 corrected. Document only correction, the description is now in line with what is returned 11) 2.1.8.1 Codes 503/700700, 500 and 503 added to the documentation. These also exist in v1, but were missed in the documentation. 12) 2.1.8.2 process flow has been updated to show each individual failure and clarifies which http code responses from users will result in message delivery failure notifications being generated. Codes 9013, 9014, 9015 and 9016 removed as there are no circumstances in which these can be returned. 13) 2.2 Directory version changed from v1 to v2. Functional differences specified at 2.2.2 being the ability to only return specific listTypes or specific process supported values, e.g. only returning values relevant to a single process 14) 2.2.2 Code 401/900901 corrected. Document only correction, the description is now in line with what is returned 15) 2.2.3 updated to include error responses for the directory API. 16) Throughout section 3, references changed from RCP portal to TOTSCo Account Management Portal. For more information on how to update credentials, please refer to the relevant user guide
GPLB Beta to V2.0	1) Multiple sections of the document removed or shortened to make the document more of a direct specification as the removed content was expositional and explanatory. Process specific references removed to make the document more generic. 2) References to RCP/MAP changed to more generic term of user throughout 3) 2.1.8.1 Code 401/900901 corrected. Document only correction, the description is now in line with what is returned 4) Section 2.1.4 on version control moved to 2.3 as it applies to letterbox and directory APIs 5) 2.1.8.1 updated to remove repetitive text and universal http response definitions. 6) 2.2.2 Code 401/900901 corrected. Document only correction, the description is now in line with what is returned 7) 2.2.3 updated to include error responses for the directory API. 8) Throughout section 3, references changed from RCP portal to TOTSCo Account Management Portal. For more information on how to update credentials, please refer to the relevant user guide

1.3 Contributing authors

Author	Organisation
Hub design team	Tech Mahindra Ltd.

1.4 Stakeholders and document approvals

Stakeholder Name and Title	Role	Reviewer/Approver/ Author/Contributor	Signature/Electronic Approval	Date
Tom Merritt	Business Analyst, TOTSCo	Approver		
David Norbury	Head of Delivery and Change, TOTSCo	Approver		
Manoj Upadhyay	Program Manager	Reviewer		
Niraj Suvarna	Solution Architect	Reviewer		
Vinod S	Infrastructure design	Reviewer		
Premanand Rao	Infrastructure design	Reviewer		
Talla Gopinadh	Security Design	Reviewer		
Rahul Kumar	TOTSCo Hub Design	Author/Contributor		
Sumathi Marimuthu	TOTSCO Hub Design	Author/Contributor		

1.5 Abbreviations and definitions

Abbreviation / term	Meaning / definition
TOTSCo	The One Touch Switching Company www.totsco.org.uk
TOTSCo Hub	This is the formal name used by TOTSCo to refer to the Hub which will provide services to users in support of applicable industry processes.
Error Code	The code that will be returned in a synchronous message as part of the Hub's validation of incoming messages.
Fault Code	The code that will be returned in an asynchronous message by the Hub to the sender after it has accepted and validated the message but has been unable to deliver the message to the recipient.
GPLB	Gaining Provider Led Business – Switching for Business process

Abbreviation / term	Meaning / definition
OTS	One Touch Switch process for residential customers
User	Any entity that uses the APIs specified in this document to send or receive messages or to access the directory

2 TOTSCo Hub integration specification

2.1 GPLB Letterbox API specification

The TOTSCo Hub letterbox API specification defines how messages will be sent to and received from the TOTSCo Hub.

2.1.1 Section intentionally left blank

2.1.2 URI format

The API URI format provided by the TOTSCo Hub and each user will conform to the following convention:

https://{fqdn}/letterbox/{version}/post

The elements of the URI are as follows:

URI Element	Description	Format
FQDN	The Fully Qualified Domain Name of the provider of the letterbox API. Please refer to the TOTSCO Hub FQDN and IP addresses document for the FQDN values used by the TOTSCo Hub. Each user will need to provide their FQDN as part of their Hub endpoint configuration and are encouraged to use similar naming conventions to the TOTSCo Hub environments.	Compliant with standard RFC 1035
Version	This is the version number of the letterbox API. This version will only ever change if there is a substantial update in the way messages are processed by the TOTSCo Hub. If a new version is introduced, the previous versions will remain in service for compatibility with existing processes.	vn (e.g. v2)

URI Element	Description	Format

2.1.3 API details

Field	Value
API Name	TOTSCo-LetterBoxAPI
Context	Letterbox
Version	v2
Resource	Post
Transport Level Security	https, TLS 1.3
Port	The TOTSCo Hub will expose the letterbox API using the standard port 443 for https. It is recommended that users also expose their API on port 443, but an alternative port number may be specified in the endpoint configuration if required.
Request Format	application/json
Request Headers	Authorization, Accept, Content-Type: text/plain; charset=UTF-8
Tags	Totsco

2.1.4 Section intentionally left blank

.

2.1.5 Envelope elements

Every message sent through the Hub letterbox API must have an envelope and a message body.

- This document defines the envelope.
- The message format documents for the relevant industry process define the body – this document does not repeat those specifications.

The example below shows a completed envelope, with sample values including audit information. The "_messageBody" would be where the specific message content will be defined – the body is not processed by the TOTSCo Hub.


```
{
   "envelope": {
      "source": {
         "type": "RCPID",
         "identity": "BBCD",
         "correlationID": "10266c25-1861-49d7-9157-436bc47fa746"
      },
      "destination": {
         "type": "RCPID",
         "identity": "BCBX",
         "correlationID": "ca2ba334-df49-46f4-9853-5c75c73fcc9a"
      },
      "routingID": " businessSwitchMatchFailure",
      "auditData": [
         {
            "name": "auditFieldName",
            "value": "auditFieldValue"
         }
      ]
   },
   "_messageBody": {
```

```
        "_comment": "The real message body would appear here in plain text".
    }
}
```

The following table defines each element of the envelope:

JSON element	Description	Format	Notes
Envelope	A container defining the delivery information for any associated message.	Object	Required
Source	A container defines the originator of the message and represents the return address for any message requiring a response.	Object	Required
Destination	A container representing the destination of the message and used by the TOTSCo Hub to identify the correct recipient letter box to deliver it to.	Object	Required
source/type destination/type	The name of the directory list where the identity can be found and validated. E.g., "RCPID" for switching messages.	String	Required
source/identity destination/identity	The identity of the sending or receiving entity for the message as defined in the directory list selected. E.g., the RCPID. *Note: The RCPIDs allocated by TOTSCo to users will be in the format of four alphabetic characters (XXXX) without vowels. There is an exception to this as 'TOTSCO' is the RCPID that is the sender of message delivery failure notifications. Please refer to section 2.1.8.2 Asynchronous message delivery failure notifications for more information on message delivery failure notifications.*	String	Required
source/correlationID destination/correlationID	A string of characters that the message originator will recognise and allow matching of a reply to a request message. In a source element, the correlationID must always be provided, the format can be anything the originator chooses to support their messaging process but should be sufficiently unique to allow correlation of response with request over a reasonable period. *Note: a message delivery failure returned by the TOTSCo Hub will not contain a source correlationID. Please see*	String	Required /Optional

JSON element	Description	Format	Notes
	2.1.8.2 Asynchronous message delivery failure notifications for more information. In a destination element, the correlationID would only be populated when the message is being sent in response to a message previously sent to you (for example, a confirmation or a failure message in GPLB would contain a destination correlationID, but a request message need not contain one). In that case the destination correlationID will be the same value as the source correlationID that was sent by the original sender of the message – i.e., it is being reflected to them. The maximum length of the correlationID is 256 characters. *Note: At the time of publication, there is no requirement for correlationIDs to be unique within any applicable process. However, for OTS, the OTS Industry Process Group recommended that users use a unique source correlationID for every message in the format of a UUID.*		
routingID	The routingID that the Hub will use to route the message to the recipients desired destination. Each messaging specification will have its own requirements for how this value is populated, but the value must be supported by the Hub. Please refer to the applicable message specification document for a full list of supported values for routingID for the applicable process.	String	Required
auditData	A list of name value pairs that TOTSCo will use for auditing and reporting. Each messaging specification will have its own requirements for audit data. An example is the error code for a failure response.	Array	Optional
auditData/name	The text name of the property being provided for auditing. The maximum length of the 'name' is 256 characters.	String	Required
auditData/value	The value associated to the named entity above. The maximum length of the 'value' is 256 characters.	String	Required
_messageBody	This is the message to be sent to the recipient. The actual element name should be based on the message being sent. Please refer to the applicable message	String	Required

JSON element	Description	Format	Notes
	specification document for the messages supported for the relevant process.		

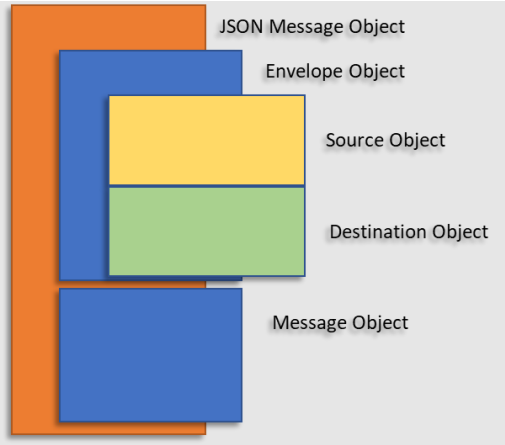


Figure 1 – Post Office JSON Message Envelope Structure

2.1.6 Auditing requirements

The audit Data in the envelope must be populated according to the following rules when sending switch messages.

- faultCode – If a failure message is being sent through the Hub, the fault code in that failure message must also be replicated in the audit data.

Here is an example showing a failure message sent through the Hub:

```
{
   "envelope": {
      "source": {
         "type": "RCPID",
         "identity": "RYBL",
         "correlationID": "10266c25-1861-49d7-9157-436bc47fa746"
      },
      "destination": {
         "type": "RCPID",
         "identity": "RYMN",
         "correlationID": "ca2ba334-df49-46f4-9853-5c75c73fcc9a"
      },
      "routingID": "residentialSwitchMatchFailure",
      "auditData": [

         {
            "name": "faultCode",
            "value": "1103"
         }
      ]
   },
   "residentialSwitchMatchFailure": {
      "faultCode": "1103",
      "faultText": "Account not found".
   }
}
```

2.1.7 Section intentionally left blank

2.1.8 Letterbox responses

2.1.8.1 Synchronous error responses

The letterbox API REST interface is synchronous, meaning that when a message is sent to the letterbox it will reply within the same communication session. That reply does not contain a JSON message on a successful post as its purpose is only to acknowledge receipt of the message being delivered to the Hub letter box. However, on a failure, a small JSON error structure will be returned describing the nature of the error.

The letterbox APIs will acknowledge message receipt with a HTTP 202 response code, the definition of which is as follows: "*The request has been accepted for processing, but the processing has not been completed. The request might or might not be eventually acted upon and may be disallowed when processing occurs.*"

Where the TOTSCo Hub is the recipient of a message sent by a user, the 202 responses will be sent once the Hub has validated the message. The Hub will only validate the message envelope and not the contents of the message body. The Hub will validate the message envelope for the below:

- Validated the security mechanism (OAuth credentials/API key, mTLS certificate)
- Validated the size of the message.
- Validated the format of the message envelope structure (valid JSON structure as per the API specification)
- Validated the contents of the envelope.
- Verified that the sender is authorised to send on behalf of the defined source.
- Verified that the source and destination is valid and status is Active.

For the error responses, a JSON message will be returned. Examples of the error responses for different scenarios are given in the tables below:

JSON element	Description	Format
errorCode	A numeric description of the specific error as defined in table above. *Note: Integer is used here, and throughout the document, to indicate that the value returned will be a numeric value. However, it will be returned within quotation marks as per the examples in the following table, e.g. "9000" rather than 9000.*	Integer
errorText	A description that represents the nature of the error and can be used by the message originator to determine remedial action.	String
code	A numeric description of the specific http error as defined in table above	Integer
type	Type of the error	String
message	A definition of the error	String
nextAccessTime	Refers to the next scheduled time when API will be accessed.	Timestamp
description	A description of the specific nature of the error with suggested remedial action.	String

The following table gives the sample response the Hub will generate in the event of an error.

Code	Message	Description	Error response

400	Bad Request	Rejected message size – If the message size is more than 256Kb, the message will be rejected.	{
			"errorCode": "9017",
			"errorText": "Request message size limit is exceeded. Maximum allowed bytes are 256000.
			}

400	Bad Request	Schema validation failed in the Request: [Path '/directory'] Object has missing required properties ([\"listID\"])	{
			"code": "400",
			"message": "Bad Request",
			"description": " Schema validation failed in the Request: [Path '/directory'] Object has missing required properties ([\"listID\"]), "
			}

400	Bad Request	Unknown or invalid source Type.	{
			"errorCode": "9002",
			"errorText ": "Unknown or invalid source Type."
			}

400	Bad Request	Unknown or invalid source	{
			"errorCode": "9003",
			"errorText": "Unknown or invalid source ID."
			}

400	Bad Request	Unknown or invalid destination Type	{
			"errorCode": "9000",
			"errorText ": "Unknown or invalid destination Type."
			}

400	Bad Request	Unknown or Invalid destinationID	{
			"errorCode": "9001",
			"errorText ": "Unknown or invalid destination ID."
			}

400	Bad Request	No routingID is mapped with Source	{
			"errorCode": "9010",
			"errorText ": "No routingID is mapped with Source RCP."
			}

400	Bad Request	Unknown or invalid routing ID.	{
			"errorCode": "9012",
			"errorText ": "Unknown or invalid routing ID."
			}

401	UNAUTHORIZED ERROR	Source type and ID not permitted from originating location	{
			"errorCode": "9004",
			"errorText ": "Source type and ID not permitted from originating location."
			}

401	Unauthorized	Invalid Credentials. Make sure you have provided the correct security credentials.
		Note: code 900901 is an internal process code and not an error code. This message is sent for invalid OAuth2 token
			{
			"code": "900901",
			"message": "Invalid Credentials",
			"description": "Invalid Credentials. Make sure you have provided the correct security credentials."
			}

401	Unauthorized	Invalid Credentials. Make sure your API invocation call has a header.
		Note: code 900902 is an internal process code and not an error code. This message is sent for invalid API-key
			{
			"code": "900902",
			"message": "Missing Credentials",
			"description": "Invalid Credentials. Make sure your API invocation call has a header: 'Authorization : Bearer ACCESS_TOKEN' or 'Authorization : Basic ACCESS_TOKEN' or 'apikey: API_KEY'"
			}

403	Forbidden	(900908) – Resource forbidden
		Note: code 900908 is an internal process code and not an error code. This message is sent when the source is not subscribed/registered in the Hub
			{
			"code": "900908",
			"message": "Resource forbidden",
			"description": "Resource forbidden"
			}

403	Forbidden Error	Unknown or invalid source account status.	{
			"errorCode": "9003",
			"errorText": "Source RCPID account status is not valid"
			}

403	Forbidden Error	Unknown or Invalid destination account status	{
			"errorCode": "9001",
			"errorText ": "Destination RCPID account status is not valid."
			}

404	Not Found	No matching resource found for given API Request	{
			"code": "404",
			"type": "Status report",
			"message": "Runtime Error",
			"description": "No matching resource found for given API Request"
			}

405	Method Not Allowed	Method not allowed for given API resource	{
			"code": "405",
			"type": "Status report",
			"message": "Runtime Error",
			"description": "Method not allowed for given API resource"
			}

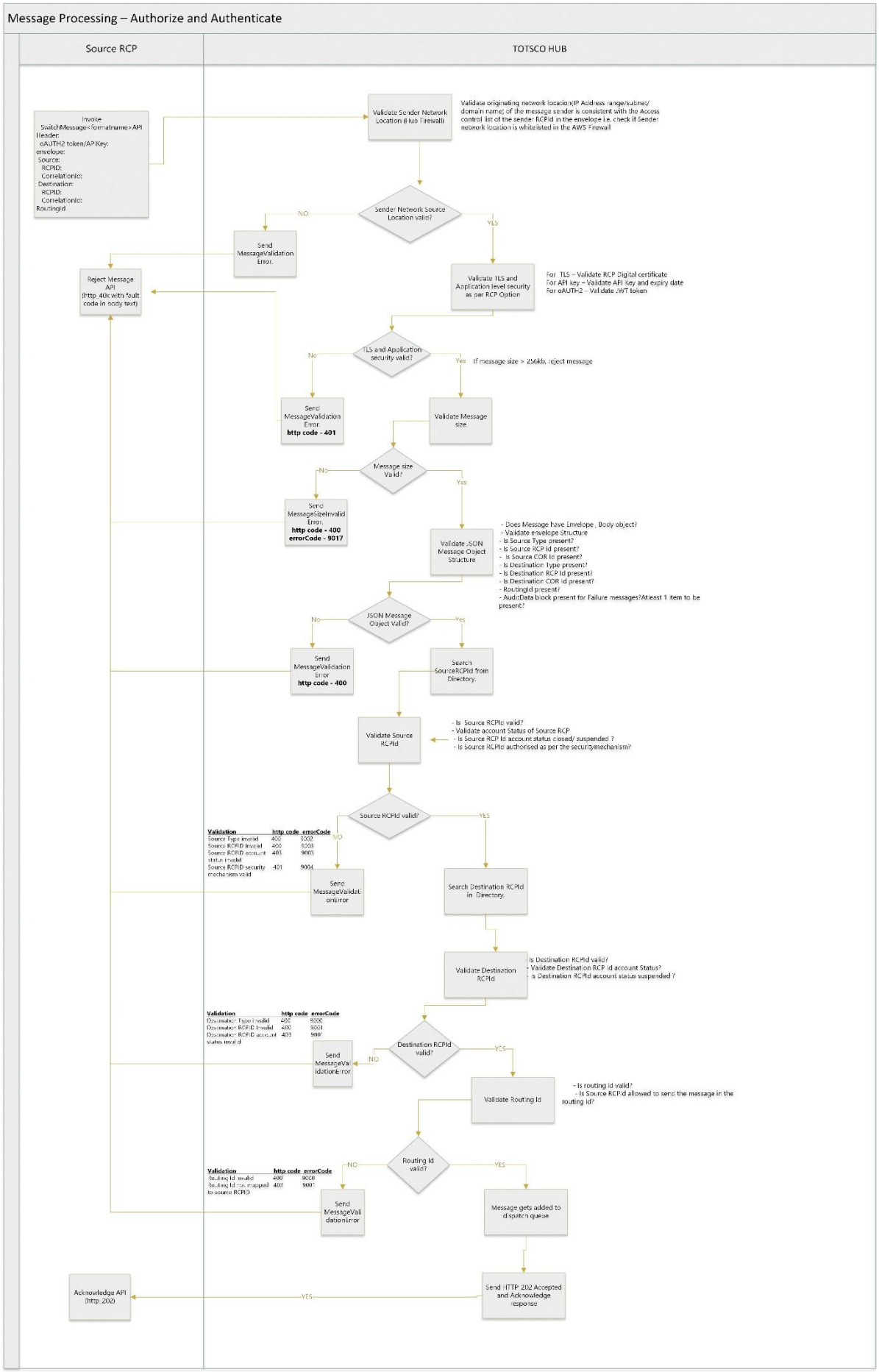
429	Too Many Requests	Hub exceeded the quota. You can access API after YYYY-MMM-DD xx: xx:xx+xxxx UTC.
		
		Note: code 900804 is an internal process code and not an error code. The date and time mentioned above will be one minute after first message arrives.
			{
			"code": "900804",
			"message": "Message throttled out",
			"description": "Hub exceeded the quota. You can access API after 2023-May-18 04:43:00+0000 UTC",
			"nextAccessTime": "2023-May-18 04:43:00+0000 UTC"
			}

500	INTERNAL SERVER ERROR	An unexpected error has occurred while processing the request, Error connecting to the back end.	{
			"code": "101503",
			"type": "Status report",
			"message": "Runtime Error",
			"description": "Error connecting to the back end"
			}

503	API Blocked	This API has been blocked temporarily. Please try again later or contact the system administrators.
		
		Note: code 700700 is an internal process code and not an error code.
			{
			"code": "700700",
			"type": "API blocked",
			"description": "This API has been blocked temporarily. Please try again later or contact the system administrators."
			}

500	Blocked	Internal server error returned by the Hub firewall when recognized as an attack and access denied.	Standard HTML Page with Blocked message.
503	Server Unavailable	When there are no backend pool servers active or if user tries to access any URL path which does not exist.	Standard HTML Page with Server Unavailable message.

The below diagram shows the flow of the message validation when it is received in the Hub.


Message Processing – Authorize and Authenticate
Source RCP
TOTSCO HUB
Invoke SwitchMessage<formatname> API
Header:
oAUTH2 token/API Key:
envelope:
Source:
RCPID:
CorrelationId:
Destination:
RCPID:
CorrelationId:
RoutingId
Validate Sender Network Location (Hub Firewall)
Validate originating network location(IP Address range/subnet/ domain name) of the message sender is consistent with the Access control list of the sender RCPId in the envelope i.e. check if Sender network location is whitelisted in the AWS Firewall
Sender Network Source Location valid?
NO
YES
Send MessageValidation Error.
Reject Message API (Http 40x with fault code in body text)
Validate TLS and Application level security as per RCP Option
For TLS – Validate RCP Digital certificate
For API key – Validate API key and expiry date
For oAUTH-2 – Validate JWT token
TLS and Application security valid?
No
Yes
If message size > 256kb, reject message
Send MessageValidation Error. http code - 401
Validate Message size
Message size Valid?
No
Yes
Send MessageSizeInvalid Error. http code - 400 errorCode - 9017
Validate JSON Message Object Structure
- Does Message have Envelope , Body object?
- Validate envelope Structure
- Is Source Type present?
- Is Source RCP Id present?
- Is Source COR Id present?
- Is Destination Type present?
- Is Destination RCP Id present?
- Is Destination COR id present?
- RoutingId present?
- AuditData block present for Failure messages?Atleast 1 item to be present?
JSON Message Object Valid?
No
Yes
Send MessageValidation Error http code - 400
Search SourceRCPId from Directory.
Validate Source RCPId
- Is Source RCPId valid?
- Validate account Status of Source RCP
- Is Source RCP Id account status closed/ suspended ?
- Is Source RCPId authorised as per the securitymechanism?
Source RCPId valid?
YES
NO
Validation | http code | errorCode
Source Type invalid | 400 | 9002
Source RCPID Invalid | 400 | 9003
Source RCPID account status invalid | 403 | 9003
Source RCPID security mechanism valid | 401 | 9004
Send MessageValidationError
Search Destination RCPId in Directory.
Validate Destination RCPId
Is Destination RCPId valid?
Validate Destination RCP id account Status?
Is Destination RCPId account status suspended ?
Validation | http code | errorCode
Destination Type invalid | 400 | 9000
Destination RCPID Invalid | 400 | 9001
Destination RCPID account status invalid | 403 | 9001
Destination RCPId valid?
NO
YES
Send MessageValidationError
Validate Routing Id
- Is routing id valid?
- Is Source RCPId allowed to send the message in the routing id?
Routing Id valid?
NO
YES
Validation | http code | errorCode
Routing Id invalid | 400 | 9020
Routing Id not mapped to source RCPID | 403 | 9001
Send MessageValidationError
Message gets added to dispatch queue
Send HTTP 202 Accepted and Acknowledge response
Acknowledge API (http 202)
YES

2.1.8.2 Asynchronous message delivery notifications

In the event of the TOTSCo Hub being unable to deliver a message to its intended recipientthe TOTSCo Hub will create a message back to the originator of the message in the following format.


```
{
   "envelope": {
      "source": {
         "type": "RCPID",
         "identity": "TOTSCO"
      },
      "destination": {
         "type": "RCPID",
         "identity": "BTYD",
         "correlationID": "10266c25-1861-49d7-9157-436bc47fa746"
      },
      "routingID": "messageDeliveryFailure",
      "auditData": [{
            "name": "originalDestinationType",
            "value": "RCPID"
         },
         {
            "name": "originalDestination",
            "value": "BRQD"
         },
         {
            "name": "originalRoutingID",
            "value": "businessSwitchMatchRequest"
         },
         {
            "name": "faultCode",
            "value": "9005"
         }
      ]
   },
   "messageDeliveryFailure": {
      "code": "9005",
      "text": "Unable to deliver the message to the destination, no valid route.",
      "severity": "failure"
   }
}


{
   "envelope": {
      "source": {
         "type": "RCPID",
         "identity": "TOTSCO"
      },
      "destination": {
         "type": "RCPID",
         "identity": "RYBL",
         "correlationID": "10266c25-1861-49d7-9157-436bc47fa746"
      },
      "routingID": "messageDeliveryFailure",
      "auditData": [{
            "name": "originalDestinationType",
            "value": "RCPID"
         },
         {
            "name": "originalDestination",
            "value": "RYMN"
         },
```

```
            {
               "name": "originalRoutingID",
               "value": "residentialSwitchMatchRequest"
            },
            {
               "name": "faultCode",
               "value": "9006"
            }
         ]
      },
      "messageDeliveryFailure": {
         "code": "9006",
         "text": "Unable to deliver the message to the destination, rejected, invalid message
format.",
         "severity": "failure"
      }
}


{
   "envelope": {
      "source": {
         "type": "RCPID",
         "identity": "TOTSCO"
      },
      "destination": {
         "type": "RCPID",
         "identity": "BTYD",
         "correlationID": "10266c25-1861-49d7-9157-436bc47fa746"
      },
      "routingID": "messageDeliveryFailure",
      "auditData": [{
            "name": "originalDestinationType",
            "value": "RCPID"
         },
         {
            "name": "originalDestination",
            "value": "BRQD"
         },
         {
            "name": "originalRoutingID",
            "value": "businessSwitchMatchRequest"
         },
         {
            "name": "faultCode",
            "value": "9007"
         }
      ]
   },
   "messageDeliveryFailure": {
      "code": "9007",
      "text": "Recipient rejected message.",
      "severity": "failure"
   }
}
{
   "envelope": {
      "source": {
         "type": "RCPID",
         "identity": "TOTSCO"
      },
      "destination": {
         "type": "RCPID",
         "identity": "RYBL",
         "correlationID": "10266c25-1861-49d7-9157-436bc47fa746"
      },
      "routingID": "messageDeliveryFailure",
```

```
        "auditData": [{
              "name": "originalDestinationType",
              "value": "RCPID"
           },
           {
              "name": "originalDestination",
              "value": "RYMN"
           },
           {
              "name": "originalRoutingID",
              "value": "residentialSwitchMatchRequest"
           },
           {
              "name": "faultCode",
              "value": "9008"
           }
        ]
     },
     "messageDeliveryFailure": {
        "code": "9008",
        "text": "Unable to deliver the message to the destination, timed out.",
        "severity": "failure"
     }
  }
```


The message Delivery Failure body describes a notification to the sender of the original message of a failure to deliver the message. The source information will represent the TOTSCO Hub, and the audit data will contain the original intended message recipient and destination. The originator's correlation ID will be returned in the destination information. Note that the source does not contain a correlationID, the messageDeliveryFailure cannot be replied to as it is a notification, and therefore no correlationID is required.

The content of this message then describes the notification information.

JSON element	Description	Format
messageDeliveryFailure	Container for messages that have failed delivery to the destination. These messages will be sent from the post office.	Object
code	A number that represents the nature of the fault and can be used by the message originator to determine remedial action.	Integer
text	A description of the associated response code	String
severity	An indicator of the nature of the message about the processing of the originators' message. Currently the only applicable value is "failure".	String

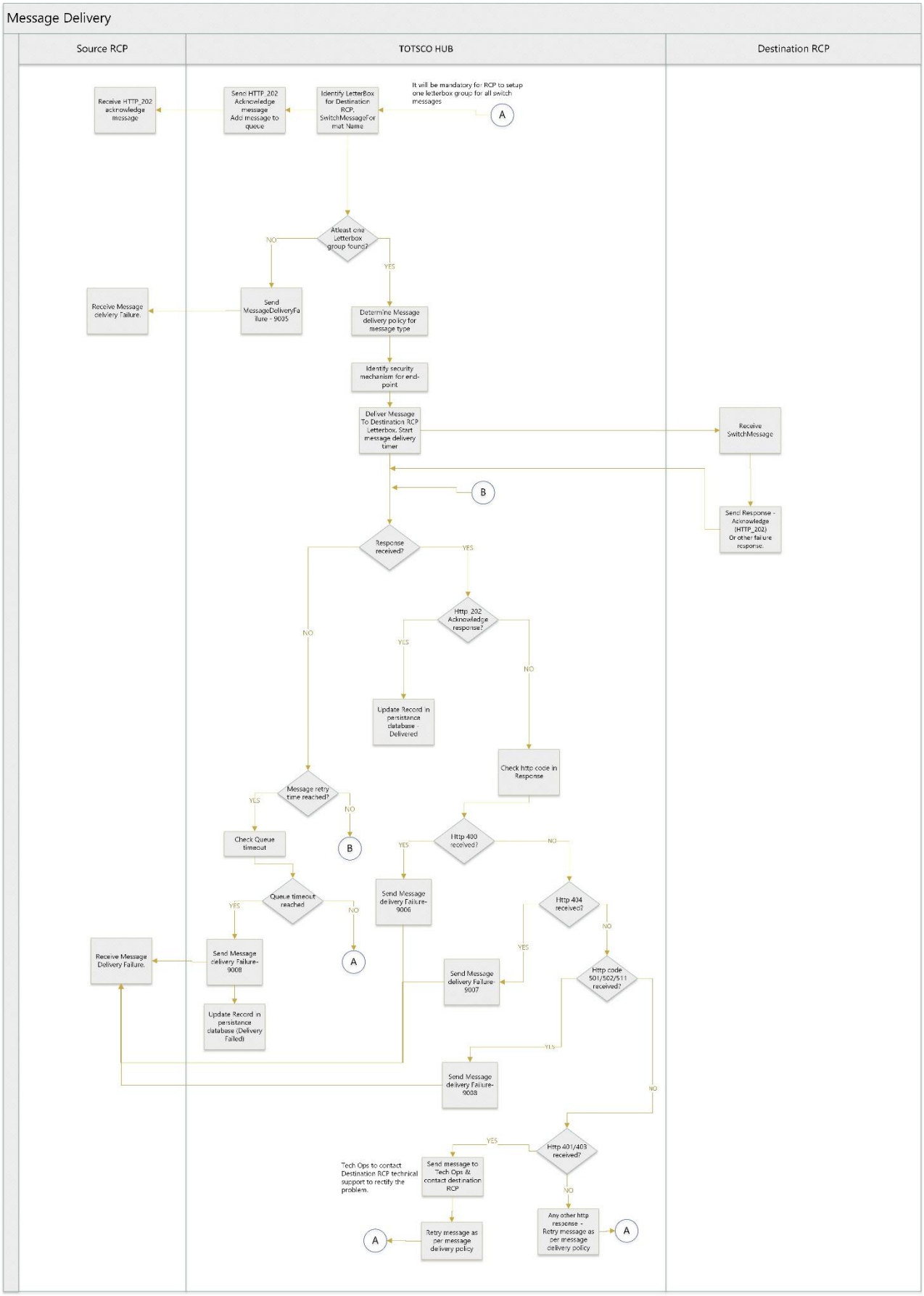
The following table defines the list of response codes the post office will generate in the event of a message delivery failure.

Code	Text	Severity
9005	Unable to deliver the message to the destination, no valid route.	failure
9006	Unable to deliver the message to the destination, rejected, invalid message format. *Note: message sent if destination endpoints reject message and returns http_400 error response.*	failure
9007	Recipient rejected message.	failure

	Note: message sent if destination endpoints reject message and returns http_404 error response.	
9008	Unable to deliver the message to the destination, timed out. *Note: message sent if the destination endpoint returns http_501, 502 or 511 response. Message also sent if the original message times out in line with the message delivery policy.*	failure

Please note that only an http_202 response will be treated by the Hub as a successful delivery. The Hub will stop attempting to deliver a message and will return one of the above message delivery failure notifications if an http_400, 404, 501, 502 or 511 response is received. If any other http response is received, or if no http response is received, the Hub will continue to deliver the message in line with the relevant message delivery policy until it times out or one of the aforementioned http responses is received.

The below diagram shows the flow of the message delivery after is has been validated and assigned to delivery queue.


Message Delivery
Source RCP
TOTSCO HUB
Destination RCP
Receive HTTP_202 acknowledge message
Send HTTP_202 Acknowledge message Add message to queue
Identify LetterBox for Destination RCP, SwitchMessageFormat Name
It will be mandatory for RCP to setup one letterbox group for all switch messages
A
Atleast one Letterbox group found?
NO
YES
Receive Message delivery Failure.
Send MessageDeliveryFailure - 9005
Determine Message delivery policy for message type
Identify security mechanism for end-point
Deliver Message To Destination RCP Letterbox, Start message delivery timer
Receive SwitchMessage
Send Response - Acknowledge (HTTP_202) Or other failure response.
B
Response received?
YES
NO
Http 202 Acknowledge response?
YES
NO
Update Record in persistance database - Delivered
Check http code in Response
Message retry time reached?
YES
NO
Check Queue timeout
B
Http 400 received?
YES
NO
Queue timeout reached
YES
NO
Send Message delivery Failure-9006
Http 404 received?
NO
YES
Receive Message Delivery Failure.
Send Message delivery Failure-9008
A
Send Message delivery Failure-9007
Http code 501/502/511 received?
Update Record in persistance database (Delivery Failed)
YES
Send Message delivery Failure-9008
NO
YES
Http 401/403 received?
Tech Ops to contact Destination RCP technical support to rectify the problem.
Send message to Tech Ops & contact destination RCP
NO
Any other http response - Retry message as per message delivery policy
A
A
Retry message as per message delivery policy

2.2 Directory API specification

The TOTSCo Hub maintains a central directory of all entities involved in the sending and receiving of messages using the TOTSCo Hub. To be able to send message via the Hub, all users need access to the directory to obtain the directory list.

2.2.1 Directory API details

Field	Value
API Name	TOTSCo-DirectoryAPI
Context	/directory
Version	v2
Description	To send message via the Hub, all users need access to the directory to obtain the directory list.
Tags	Totsco
Transport Level Security	https, TLS1.3
HTTP Method	GET
Resources	/entry
Request Format	text/plain
Request Header	Authorization, Accept, Content-Type: text/plain; charset=UTF-8

The directory API returns two types of directory information.

1. For a specified identity of a specified list type Hub
2. For all identities of a specified list type

Note: list type is enumerated, at the time of publication there is only one value – RCPID.

2.2.2 For a specified identity of a specified list type Hub

To get the details of a specified RCPID, the listID and identityID with “RCPID” as value will need to be passed as query parameter in GET method as per below format.

```
https://{fqdn}/directory/{version}/entry?listType={listType}&identity={identity}
```

The elements of the URI are as follows:

URI Element	Description	Format/example
FQDN	The Fully Qualified Domain Name of the provider of the directory API. The TOTSCo Hub FQDN will be provided by TOTSCo.	Compliant with standard RFC 1035
Version	This is the version number of the directory API. If a new version is introduced, the previous versions will remain in service for compatibility with existing processes.	vn (e.g. v2)

<table>
<tr><td></td><td></td><td></td></tr>
<tr><td>listType</td><td>This mandatory value specifies the entity list types to be included in the results.

For example, RCPID.</td><td>Enumerated String
(e.g., RCPID)</td></tr>
<tr><td>identity</td><td>This is an optional query string parameter.

Permitted values for identity parameter (if provided) will be:
- ‘all’
- the relevant process name – e.g. ‘OTS’ and ‘GPLB’
- <specific identity value> - e.g. a specific RCPID

If this query string parameter is not specified, then all identities of a given list type will be returned.

If this query string parameter is present and its value is “all” then all identities of a given list type will be returned.

If this query string parameter is present and its value is anything other than “all” then the directory entry for the identity of the specified list type will be returned.

Please see the below examples assuming a list type of RCPID and using OTS and GPLB as supported processes:

<table>
<tr><td>Identity</td><td>Output</td></tr>
<tr><td>‘all’</td><td>All RCPIDs with Process supported = ‘OTS’ and All RCPIDs with Process supported = ‘GPLB’</td></tr>
<tr><td>Blank</td><td>All RCPIDs with Process supported = ‘OTS’ and All RCPIDs with Process supported = ‘GPLB’</td></tr>
<tr><td>not provided</td><td>All RCPIDs with Process supported = ‘OTS’ and All RCPIDs with Process supported = ‘GPLB’</td></tr>
<tr><td>‘OTS’</td><td>All RCPIDs with Process supported = ‘OTS’</td></tr>
<tr><td>‘GPLB’</td><td>All RCPIDs with Process supported = ‘GPLB’</td></tr>
<tr><td>Invalid identity</td><td>http code : 404 – Invalid identity</td></tr>
<tr><td><valid RCPID></td><td>
For provided RCPID</td></tr>
</table>
</td><td>String
(e.g., RGXD)</td></tr>
</table>

Response Code:

The following table defines the list of response codes the HUB will generate in the event of a success/error processing Directory API call.

Code	Message	Description	Sample response
200	Ok	Request is valid and the Hub could retrieve entry(s) as per query string	Please refer to the section 2.2.4 for the directory response structure
400	Bad Request	Schema validation failed in the Request: [Path '/directory'] Object has missing required properties ([\"listID\"])	{ "code": "400", "message": "Bad Request", "description": " Schema validation failed in the Request: [Path '/directory'] Object has missing required properties ([\"listID\"]), " }
401	Unauthorized	Invalid Credentials. Make sure you have provided the correct security credentials. *Note: code 900901 is an internal process code and not an error code.*	{ "code": "900901", "message": "Invalid Credentials", "description": "Invalid Credentials. Make sure you have provided the correct security credentials." }
404	Not Found	Invalid identityID, identityID not available in directory Hub.	identityID not found.

2.2.3 Directory API Response Structure

a) Sample directory response structure if all identities are requested


```
{
  "list": [
    {
      "listType": "RCPID",
      "identity": [
        {
          "id": "RTYQ",
          "name": "Xenon Telecom",
          "processSupport": [
            {
              "process": "OTS",
              "status": "ACTIVE"
            }
          ],
          "resource": [
            {
              "name": "salesAssistURL",
              "type": "URL",
              "value": "https://xenon.com/sales"
            },
            {
              "name": "customerAssistURL",
              "type": "URL",
              "value": "https://xenon.com/OTS"
            }
          ]
        },
        {
          "id": "BTYD",
          "name": "Xenon Telecom - Business",
          "processSupport": [
            {
              "process": "GPLB",
              "status": "ACTIVE"
            }
```

```
          ],
          "resource": [
            {
              "name": "salesAssistURL",
              "type": "URL",
              "value": "https://xenon.com/GPLBsales"
            },
            {
              "name": "customerAssistURL",
              "type": "URL",
              "value": "https://xenon.com/GPLB"
            }
          ]
        },
        {
          "id": "RYBL",
          "name": "Bands Telecom UK Ltd.",
          "processSupport": [
            {
              "process": "OTS",
              "status": "ACTIVE"
            }
          ],
          "resource": [
            {
              "name": "salesAssistURL",
              "type": "URL",
              "value": "https://bandstelecom.co.uk/OTS"
            },
            {
              "name": "customerAssistURL",
              "type": "URL",
              "value": "https://bandstelecom.co.uk/sales"
            }
          ]
        },
        {
          "id": "BRQD",
          "name": "Jive Virtual Networks Ltd.",
          "processSupport": [
            {
              "process": "GPLB",
              "status": "ACTIVE"
            }
          ],
          "resource": [
            {
              "name": "salesAssistURL",
              "type": "URL",
              "value": "https://jivevirtual.com/Business"
            },
            {
              "name": "customerAssistURL",
              "type": "URL",
              "value": "https://jivevirtual.com/customersupport"
            }
          ]
        }
      ]
    }
  ]
}
```


b) Sample directory response structure if a single process identity (e.g. GPLB) is requested


```
{
  "list": [
    {
      "listType": "RCPID",
      "identity": [
        {
          "id": "BTYD",
          "name": "Xenon Telecom - Business",
          "processSupport": [
```

```
            {"process": "GPLB",
               "status": "ACTIVE"
            }
               ],
          "resource": [
            {
              "name": "salesAssistURL",
              "type": "URL",
              "value": "https://xenon.com/GPLBsales"
            },
            {
              "name": "customerAssistURL",
              "type": "URL",
              "value": "https://xenon.com/GPLB"
            }
          ]
        },
        {
          "id": "BRQD",
          "name": "Jive Virtual Networks Ltd.",
          "processSupport": [
               {"process": "GPLB",
               "status": "ACTIVE"
               }
               ],
          "resource": [
            {
              "name": "salesAssistURL",
              "type": "URL",
              "value": "https://jivevirtual.com/Business"
            },
            {
              "name": "customerAssistURL",
              "type": "URL",
              "value": "https://jivevirtual.com/customersupport"
            }
          ]
        }
      ]
    }
  ]
}
```


c) Sample directory response if listType is not passed


```
{
    "code": "404",
    "description": "Invalid ListType, ListType cannot be empty"
}
```


d) Sample directory response if invalid listType is requested


```
{
    "code": "404",
    "description": "Invalid ListType, ListType is not available in directory hub"
}
```


e) Sample directory response if invalid identity is requested


```
{
    "code": "404",
    "description": "Invalid identity, identity not available in directory hub"
}
```


Response elements of the JSON document will be as follows.

JSON element	Description	Format	Notes
list	An array of the lists for the directory information	Object array	Required
list/listType	The list type associated to the contained identities. *Note: At the time of publication, the directory API supports the value of "RCPID". However, other listTypes may be applicable in the future*	String	Required
list/identity	An array of all the identity objects applicable to the list type	Object array	Required
identity/id	The value assigned to an entity for the purposes of messaging via the TOTSCo Hub.	String	Required
identity/name	Value to identify the trading name of the organization	String	Required
identity/processSupport	An array of objects defining what industry processes this identity supports.	String	Optional
processSupport/process	The name of the industry process. *Note: At the time of publication, the directory API supports the values of "OTS" and "GPLB". However, other processes may be applicable in the future*	String	Required
processSupport/status	A value indicating the production status of this process. Current supported values are "ACTIVE" and "SUSPEND".	String Enumerated	Required
identity/resource	A list of objects containing resources applicable to the identity.	Array	Optional
resource/name	The names of the resources will be a set of industry agreed values to represent a use or function. Samples include "CustomerAssistURL", "SalesAssistURL".However, other names may be applicable	String	Required
resource/type	This value provides a type to represent the resource supplied. Many resources may only support a single type, others may support multiple. This value specifies how to interpret the value provided.	String	Required
resource/value	This is the value of the named resource.	String	Required

The recommendation is to use this service nightly, or at the most weekly, to request a full list of all the latest information.

If you have received a message from an unknown source ID, then you would request that single identity from the directory to update your local cache.

2.3 Version Control

The TOTSCo Hub will support API versioning, up to a maximum of five API versions. The current version along with four previous versions. For any major or minor changes, the API version will be updated and notified through TOTSCo communication.

3 Security Implementation

The TOTSCo Hub will support two levels of security and authentication so that both the Hub and the user can ensure that the API requests come from a legitimate source and that the requesting client is authorized to send the requested data.

1. **Transport layer security (TLS)** - To provide confidentiality and integrity of messages in transit. The TLS implementations supported will include both standard TLS and mutual TLS (mTLS).
2. **Application-level security** – To enforce access control at API gateways, so that API calls are authorised. The Hub will support the implementation of OAuth2.0 and API-key protocols.

The Hub as the server will support the following combinations of TLS and application-level security when clients connect to it.

1. OAuth 2.0 + standard TLS
2. OAuth 2.0 + mutual TLS
3. Web API Keys + standard TLS
4. Web API Keys + mutual TLS
5. mutual TLS, without authentication, but only when the user provides their own Certificate.

For user's own server-side implementations, the following valid combinations of TLS and application-level security will be supported by the Hub.

1. OAuth 2.0 + standard TLS
2. OAuth 2.0 + mutual TLS
3. Web API Keys + standard TLS
4. Web API Keys + mutual TLS
5. mutual TLS without authentication, but only when the user provides their own Certificate.
   The self-signed approach can avoid the need for a further authentication process as the issuing of the certificate itself is secure and is an equivalent in security levels to Web API Keys.

Note: For mutual TLS, a user can choose to either provide the certificate that the Hub should present when connecting to the user's own systems, or it can use the standard Hub certificate as they so choose.

Users can opt for any security mechanism from the above combination options at the time of configuring their source network and delivery endpoints. Security information should be configured per endpoint, and a user can choose to use different security mechanisms per endpoint if they so wish.

There are some responsibilities to ensure this security implementation is meaningful:

- Users must ensure that the FQDN they configure to connect to the Hub is from a valid source, such as documentation retrieved directly from the TOTSCo Account Management Portal.
- Users must ensure that the FQDN they configure for their endpoint(s) are valid for their organisation.
- The Hub will ensure the implementation respects the values configured, and that requests for changes are not accepted from unverified parties.

3.1 Transport Layer Security (TLS)

API requests to send messages to the Hub (e.g., User to Hub, or Hub to User) will use HTTPS. The Hub will expose its API using port 443. It is recommended that users do the same, but if they choose to use an alternative port as part of their endpoint configuration.

The version of TLS supported will be 1.3. This will be backwards compatible with version TLS 1.2.

The Hub will support both standard and mutual TLS (mTLS) methods described below.

3.1.1 Standard TLS

In Standard TLS for Client-Server setup, the server has a chain of certificate, while the client does not. When a client makes a connection to the server, the chain of certificate of the server will be presented back to the client. The client then verifies the presented chain of certificate against the CA Root certificate that has been configured in the clients' certificate store. If both certificates match, the TLS handshake is made, and the Server can send messages to Client using the encrypted TLS connection.

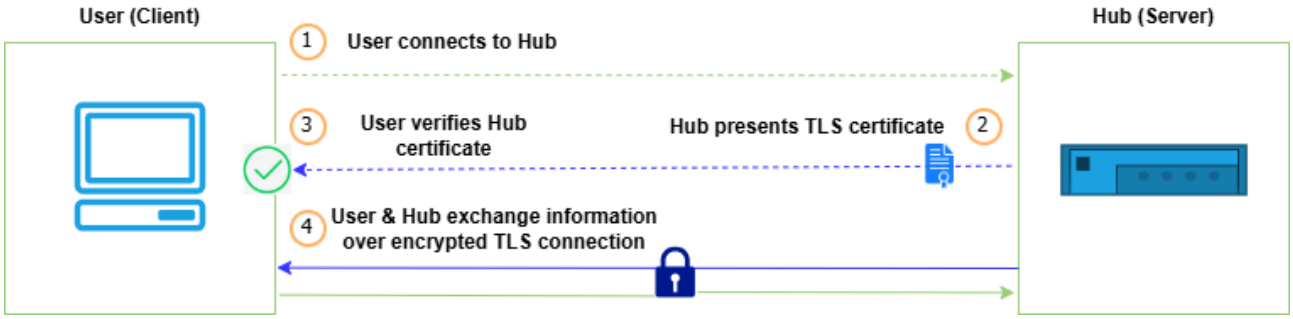
Connection from User to TOTSCo Hub

The Hub will have a Root CA Signed RSA Certificate implemented to ensure users can trust they are accessing the TOTSCo Hub. The details of this certificate will be shared with users during the setup process. TOTSCo will take full responsibility for ensuring there is always a full certified and valid certificate implemented.

Scenario-1:
If the user's client Trust store has Public CA Root Certificates pre-configured, then it is not required to configure/import the Hub's certificate.

Scenario-2:
If the user's client Trust store does not have Public CA Root Certificates pre-configured, then the user will need to request the Hub's Root certificate. The TOTSCo TechOps team will share the Hub certificate over e-mail. This chain of certificate will need to be configured by the user in their client system.

Message process



1. When a message is being sent to the Hub, the user's client application will connect to it by calling the Hub letter box API. The API URL will be shared with the user during onboarding.
2. When the Hub receives the connection from the user client, the Hub will present its TLS certificate to the user client.

3. The user client will verify the presented Hub's certificate with the CA Root certificate stored in its trusted certificate store.
4. If both chain of certificates match, the TLS handshake is made, and the user client can send messages to Hub using the encrypted TLS connection.

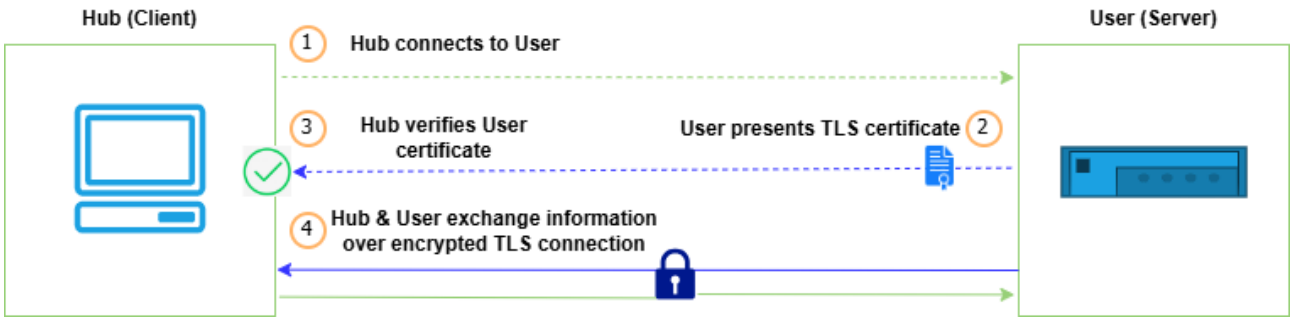
Connection from Hub to the User

The Hub will have a preconfigured list of Public Root CA signed Certificate. Refer to Appendix B for the list. It will be TOTSCo's responsibility to renew the Public Root CA signed Certificates before expiry.

Scenario-1:
If the user has implemented a chain of certificate issued by a Public Certificate authority (CA) mentioned in Appendix B, then the user will not need to share their chain of certificate with TOTSCo. It is each user's responsibility to ensure they have a certified and valid certificate in place to ensure safe communication between their client and the Hub.

Scenario-2:
If the user has implemented a chain of certificate issued by a Private CA or is using a Self-Signed certificate, then the user will need to send the certificate to the TOTSCo TechOps Team, along with its expiry date. The TOTSCo TechOps team will store the certificate in the Blue Marble CRM against the user's individual client record. TOTSCo will send an email notification regarding renewal of the user certificate before the expiry date.

Message Process



1. When a connection is being made from the Hub to the user's endpoint server, the Hub will connect to the user's endpoint server by calling the endpoint letter box API. The full URL for the user's API will need to be shared by the user at the time of onboarding, so this can be defined within the TOTSCo Hub.
2. When the user's endpoint server receives the connection from the Hub, the user's endpoint will present its chain of certificate.
3. The Hub will verify the user's presented chain of certificate with the Root CA certificate in its trusted certificate store.
4. If the chain of certificate match, the TLS handshake will complete successfully, and the Hub will proceed to send any valid messages through to the user's letterbox, using the encrypted TLS connection.

3.1.2 Mutual TLS

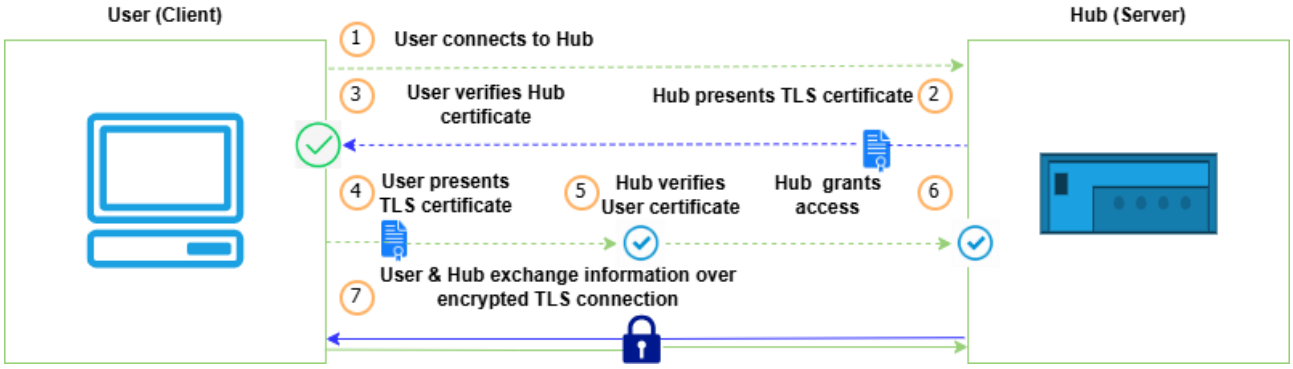
Mutual TLS or mTLS is a method for mutual authentication used in a Zero Trust security framework. Both the client and server have a certificate, which is used by both parties to authenticate each other. In Mutual TLS, the chain of certificates of the Hub and the user's endpoint server will be exchanged during configuration and used for authentication whenever a connection is made.

Users can choose how they wish to deploy a certificate for their own platform, the chosen certificate will need to be either sent or details of what is being used to TOTSCo, so this can be configured in the Hub. The Hub will then make use of the relevant certificate when connecting to the user's endpoint.

Connections from a User to the Hub

1. The TOTSCo TechOps Team will share the Hub's chain of certificate and expiry date with each user through an email. This will be shared with the user when configuring the source network and if the user has opted for the mutual TLS security option. The chain of certificate will be shared for each source network that the user needs configuring in the Hub. These can be the same or unique certificates as the user would prefer to implement.
2. The user will import the Hub's chain of certificate into its Trusted Certificate Store. The user should give the certificate an alias name in format <Certificate Name>_<DD-MM-YYYY> during import activity, where <DD-MM-YYYY> will be the expiry date of the chain of certificate.
3. The user must share their chain of certificate and the expiry date for each of the user's source networks with the TOTSCo Hub TechOps team. The certificate should have a minimum 1-year validity.
4. The Hub TechOps Team will import the user's chain of certificate (s) into the Hub Servers.

Message Process



When a connection is being made to the Hub, the user will connect to the Hub by calling the Hub letter box API. When the Hub receives the connection from the user client, it will present the Hub's chain of certificate back to the connecting client.

1. The user client will verify the certificate presented by the Hub against the one stored in the user's trusted certificate store.
2. If the Hub certificate is valid, the user client will present its chain of certificate to the Hub.
3. The Hub will verify the presented client certificate with the registered certificate for the specified endpoint client. If the certificates match, the Hub will complete the encryption handshake and grant access to the user client.
4. The user client will now be able to send messages through the Hub API within the encrypted TLS connection.

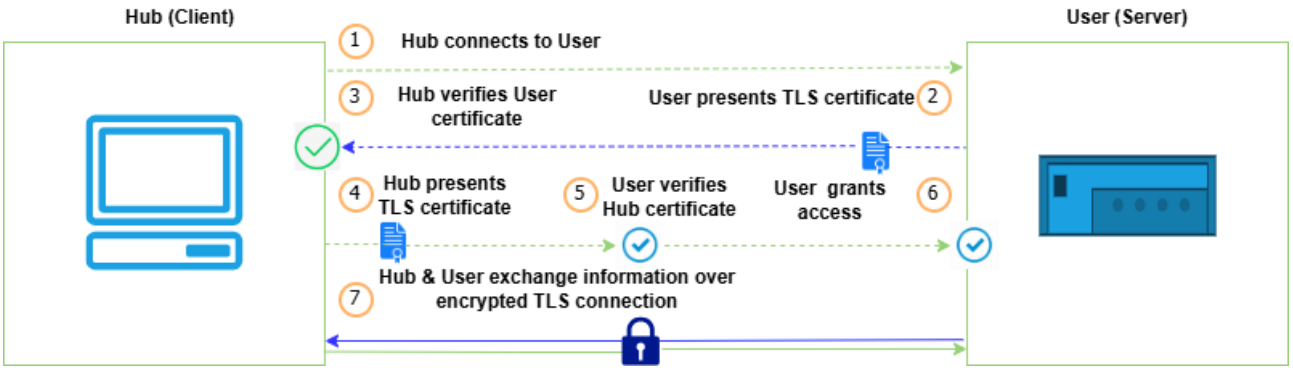
Maintaining validity of the chain of certificate.

- The Hub TechOps team will provide the Hub's new chain of certificate to each registered user, 30 days prior to expiry. The new certificate can be imported into the user trusted store before the expiry of an old certificate. The new certificate will take over immediately after expiry of any old one without any disruption. For best practice, each user should remove any expired certificates once the new one is being used.
- All registered users should send any new chain of certificates and their expiry date to the TOTSCo Hub TechOps team ideally 30 days prior to the existing certificates expiry. New certificates will be imported into the Hub trusted store at least 10 days prior to the expiry of the existing certificate. TOTSCo will remove any expired certificates as soon as they the new certificate takes over as part of its security model.

Connections from the Hub to the User

1. The Hub TechOps Team will share the user's chain of certificate and expiry date with the user via email. This will be done when the user configures the destination endpoint for receiving the messages from the Hub and triggers if the user has opted for mutual TLS security option.
2. The user must import the Hub's chain of certificate into its Trusted Store. The user should give the certificate an alias name in the format <Certificate Name>_<DD-MM-YYYY> during import activity, where <DD-MM-YYYY> is the expiry date of the certificate.
3. The user will need to share their chain of certificate and expiry date with the Hub TechOps team when setting up the user's destination endpoints. The certificates will need to be shared for each destination endpoint that the user will be configuring in the Hub. The certificate should have a minimum 1-year validity.
4. The TOTSCo TechOps team will import the user's chain of certificate(s) into the Hub's Trusted Certificate Store. The TechOps team will give the chain of certificate an alias name in the format <Certificate Name>_<DD-MM-YYYY> during the import activity, where <DD-MM-YYYY> will be the expiry date of the certificate.

Message Process



1. When a connection is being made from the Hub to the user's endpoint, the Hub will connect to the endpoint API. The API URL will have been shared by the user at the time of onboarding and setting up the endpoint in the Hub.
2. When the user's endpoint server receives the connection from the Hub, it will present the user certificate stored, to the Hub.

3. The Hub will verify the presented user's chain of certificate with the certificate for the specified endpoint stored in the Hub's trusted certificate store.
4. If the user's certificate is validated, the Hub will likewise present its certificate to the user's server.
5. The user's server will verify the Hub certificate matches the one stored in its Trusted Certificate Store. If the chain of certificate is verified, the encryption handshake will successfully complete, and the user's server will grant access to the TOTSCo Hub.
6. The Hub will then send the valid messages for the user to the endpoint server through the encrypted TLS connection.

Maintaining validity of the certificate.

1. The Hub will auto trigger reminder email communication to the user a month prior to the user's destination endpoint or Hub's certificate expiry.
2. The Hub TechOps team will provide the Hub's new digital public certificate to the user 30 days prior to expiry.
3. A user must send new chain of certificates through to the Hub TechOps team ideally 30 days prior to expiry in order to ensure there is no disruption to the user's usage of the TOTSCo Hub.

3.2 Application-level security

The API application-level authentication process ensures that individual messages within the communication stream are validated as genuine and identified to the specific sending user and Hub.

The Hub will support both the usage of **OAuth2.0** and **API-key** protocols as outlined below.

3.2.1 oAUTH2.0

OAuth provides token-based validation and authorisation for all communication between a user and the TOTSCo Hub.

Inbound Communication from User to the Hub:

Registered users will need to use an OAuth2 client to request an Access Token to successfully submit calls to the Letter Box API within the Hub. However, these need to be done via a user's OAuth 2.0 supporting client / application as an intermediary between the user's API systems and the TOTSCo OAuth 2.0 server.

Each user will be provided the following information:

- Client ID
- Secret key
- OAuth2 token generation webservice URL for each of the source locations configured by the user for sending messages to the Hub.

The Token generation URL, Client Id and Client Secret must be stored securely by the user.

The user's system will use the OAuth2 token generation URL to request an access token from the TOTSCo OAuth 2.0 server, validated by the correct client Id and secret key being provided. Please refer to section 3.2.1.1 for the process to generate access token.

Once the authentication process has successfully completed, the user can connect to the Hub services. The OAuth2 token remains valid for a period of 1 hour. Any messages after the 1-hour period will require a new

OAuth2 authentication call to be made and the generation of a new token to ensure that following messages are successfully delivered to the Hub.

Outbound Communication from TOTSCo Hub to User:

The TOTSCo OAuth 2.0 client will request an access token from the receiving user's OAuth 2.0 server for validation. Upon successfully completing the authentication process, the connection is authorised to the user's server and the Hub can then send any messages to the user's endpoint for processing.

To support the OAuth2 inbound authorisation mechanism, each registered user will need to have the following elements in place:

- Deployment of an OAuth2.0 Server
- Create a client id for TOTSCo that is then shared
- Generate a secret key for TOTSCo to be shared
- A token key generation webservice URL. (This will need to include all of the user's specified endpoints that are configured to receive messages from the TOTSCo Hub.)

The Hub will securely store each registered user's clientid and secret key, token key generation webservice URL in the Hub platform. When a message needs to be pushed to a receiving user's endpoint, the Hub will create a connection to the registered token key generation webservice URL, using the credentials of the client Id and secret key to generate a valid OAuth2 access token. This token will then be added to the https header of the message being pushed to the receiving user's endpoint. The user's endpoint will then be able to read the header and validate the credentials to ensure the identity and transaction is genuine.

3.2.1.1 Process to request and generate OAuth2 Access Token from TOTSCo Hub.

3.2.1.1.1 Access Token Request

Given below is format of OAuth2 access token request.

Request URL: https://{fqdn}/oauth2/token

FQDN: fully qualified domain name with the host and port e.g., https://host:port

Use the provided request URL to generate OAuth access token using POST method.

Required Parameters:

Header Parameter		**Comments**
Field	Value	
Authorization	Basic <Base64-encoded client_Id:client_secret>	Authorization: <auth-scheme> <authorization-parameters> The auth-scheme will be used as Basic and authorization parameter will be the base64 encoded value of "client id" and "client secret" provided by Hub.
Content-Type	application/x-www-form-urlencoded	
Body parameter		
Field	Value	

grant_type	client_credentials	**grant_type** and its value **client_credentials** need to be passed inside **x-www-form-urlencoded.**

Field	Description	Type	Notes
client_id	Client Id (will be provided by TOTSCo Hub during onboarding)	String	Required
client_secret	Client Secret (will be provided by TOTSCo Hub)	String	Required

3.2.1.1.2 Access Token Response

In a successful authorization grant type, the response will hold the access token and expiry time. Below is an example of a successful response:

HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

```
{
          "access_token":"{OAuth2 Access Token}",
          "token_type": "Bearer",
          "scope":"default",
           "expires_in":3600.
 }
```

Field	Description	Type	Default Value
access_token	Access token will be used to call the API.	String	-
token_type	Token type describes the type of the token.	String	Bearer
scope	Scope of the access token.	String	default
expires_in	Indicates the validity of token in seconds.	Integer	3600

3.2.1.1.3 Exception

Below exceptions would be sent in OAuth2 token generation response:

Error Code	Description	Root Cause
400	Bad Request	error found when invalid request value passed
401	Unauthorised	Incorrect client credentials provided
404	Not Found	when incorrect token generation URL is passed
405	Method Not Allowed	when incorrect method type is passed
415	Unsupported Media Type	when mandatory parameters are not passed

3.2.2 API Key

The Hub will support the implementation of an API Key for authentication of the connection between the Hub and the user. The API key is the simplest form of application-based security that can be configured for the TOTSCo Hub API. The Hub uses a self-contained JSON Web Token (JWT) as the API key, the API Key provides consented access and restricts actions of what the client app can perform on resources on behalf of the user.

Inbound Communication from User to Hub:

For inbound communication between a user's source network and the TOTSCo Hub, an API Key will be provided to the user by Hub TechOps team. The user will need to store the API Key securely. The user will be provided with an API Key for each of the source locations specified by the user that will need to send messages to the HUB

The API Key, which is a JWT token, will be base64 encoded and will be in below format:

```
base64(header).base64(payload).base64(signature)
```

The user's client application will be able to use the API Key when connecting to the Letter Box API. The API key has a validity of 6 months. If using the API key for connection security, it is the responsibility of the user to ensure the API key token being used is valid and not expired, before posting any messages to the Hub. The Hub TechOps will notify any user that has chosen to use the API key when these are due to expire and ensure that new API keys are sent out well ahead of the expiry date, so the user can change the keys over. The user will have to ensure that the renewed API-key is updated in any client systems connecting to the TOTSCo Hub.

Connect with letter box API Using API Key

There are two ways that the user's client application can call the letter box API using API Key.

1. Pass API Key as a header

Sample Format:

```
POST /post HTTP/1.1
https://{fqdn}/letterbox/{version}
apikey: <API_key_value>
…
```

where <API_key_value> is the HUB provided API key

2. Pass as a query parameter.

Sample Format:

POST /post?apikey=<url encoded API key value> HTTP/1.1
https://{fqdn}/letterbox/{version}
…

Please note that the API Key must be encoded using a URL encoder before passing as a query parameter in the API Request

Outbound Communication from Hub to the User:

Registered users will need to provide the API key to Hub at the time of onboarding or subsequently whenever the keys are changed. The maximum length of the user's API key should have maximum size of 256 characters. This will need to be provided for every destination end-point that will be configured by the user in the Hub. The Hub will store the API Key in its secure database.

Before making a connection to a user's destination endpoint, the Hub will validate the API expiry date. If the API key has not been renewed the message will not be sent.
This token will be sent in the http header of the message to the user'sendpoint.

Sample Format:

POST /post HTTP/1.1
https://{fqdn}/letterbox/{version}
apikey: <API_key_value>
…

where <API_key_value> is the user's provided API key for the destination endpoint.

The API key provided by the user should have a validity of 6 months and the user will have to ensure that the API key tokens are not expired. The user will be required to maintain the validity of the API-key for their destination endpoints and notify Hub Tech-ops of the renewed API key at least 30 days before the expiry.

4 Section intentionally left blank

5 Appendix A – Section intentionally left blank

6 Appendix B – List of CA Root certificates

actalisauthenticationrootca
addtrustexternalca
addtrustqualifiedca
affirmtrustcommercialca
affirmtrustnetworkingca
affirmtrustpremiumca
affirmtrustpremiumeccca
amazonrootca1
amazonrootca2
amazonrootca3
amazonrootca4
baltimorecybertrustca
buypassclass2ca
buypassclass3ca
camerfirmachambersca
camerfirmachamberscommerceca
camerfirmachambersignca
certumca
certumtrustednetworkca
chunghwaepkirootca
comodoaaaca
comodoeccca
comodorsaca
digicertassuredidg2
digicertassuredidg3
digicertassuredidrootca
digicertglobalrootca
digicertglobalrootg2
digicertglobalrootg3
digicerthighassuranceevrootca
digicerttrustedrootg4
dtrustclass3ca2
dtrustclass3ca2ev
entrust2048ca
entrustevca
entrustrootcaec1
entrustrootcag2
entrustrootcag4
geotrustglobalca
geotrustprimaryca

geotrustprimarycag2
geotrustprimarycag3
geotrustuniversalca
globalsignca
globalsigneccrootcar4
globalsigneccrootcar5
globalsignr3ca
globalsignrootcar6
godaddyclass2ca
godaddyrootg2ca
haricaeccrootca2015
haricarootca2015
identrustcommercial
identrustpublicca
letsencryptisrgx1
luxtrustglobalroot2ca
luxtrustglobalrootca
quovadisrootca
quovadisrootca1g3
quovadisrootca2
quovadisrootca2g3
quovadisrootca3
quovadisrootca3g3
secomscrootca1
secomscrootca2
securetrustca
sslrooteccca
sslrootevrsaca
sslrootrsaca
starfieldclass2ca
starfieldrootg2ca
starfieldservicesrootg2ca
swisssigngoldg2ca
swisssignplatinumg2ca
swisssignsilverg2ca
teliasonerarootcav1
thawteprimaryrootca
thawteprimaryrootcag2
thawteprimaryrootcag3
ttelesecglobalrootclass2ca
ttelesecglobalrootclass3ca
usertrusteccca

usertrustrsaca
utnuserfirstobjectca
verisignclass3g3ca
verisignclass3g4ca
verisignclass3g5ca
verisignuniversalrootca
wso2carbon
xrampglobalca

End of Document