TOTSCo

TOTSCo Hub API Specifications v1.1

A guide for developers

18 August 2023

TOTSCo

Contents

Figures

TOTSCO

1 Introduction

This TOTSCo Hub API Specifications document complements and supports the One Touch Switch (OTS) Message Specification and the OTS Industry Process document. This documentation is available from www.totsco.org.uk.

This document contains the definition of the message envelope structure as well as the letterbox API specification and Directory API specification.

The intended audience of this document is:

- Representatives of communications providers (RCPs) who are responsible for the technical implementation of the communication between that RCP and the TOTSCo Hub.
- Representatives of Tech Mahindra, the vendor chosen by TOTSCo to implement the Hub.

1.1 Communications providers and third-party integrators

The TOTSCo Hub will initially provide services for retail RCPs to exchange messages in support of the OTS process. In the future the Hub may provide services in support of other processes requiring message exchange between RCPs who may not be the retailers. So, this document used the generic term of RCP.

Note that retail RCPs may choose to engage an agent, such as a managed access provider (MAP). Any reference in this document to a requirement against an RCP would apply to a MAP providing services to an RCP.

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

1.3 Contributing authors

Author	**Organisation**
Dave Stubbs	Virgin Media
Niall Gillespie	BT
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**
Richard Steele	CTO, TOTSCo	Approver		

Tom Merritt	Business Analyst, TOTSCo	Approver		
David Norbury	Head of Delivery, TOTSCo	Approver		
Jason Bird	CSO, TOTSCo	Approver		
Alex Simmons	Technical lead, TOTSCo	Approver		
Niraj Suvarna	Enterprise Architect	Reviewer		
Rahul Mehta	Solution Architect	Reviewer		
Nimesh Soni	Solution Architect	Reviewer		
Premanand Rao	Infrastructure design	Reviewer		
Shailesh Pingle	Delivery Manager	Reviewer		
Vishal Dumbre	Security Design	Reviewer		
Rahul Kumar	TOTSCo Hub Design	Author/Contributor		
Vikash Prasad	TOTSCo Hub Design	Author/Contributor		
Nitesh Barnwal	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 RCPs in support of the OTS process, and possibly for other industry processes in the future. TOTSCo have chosen Tech Mahindra to implement and operate the TOTSCo Hub.
CP	Communications provider This is a term defined by Ofcom in their General Conditions of Entitlement as a "means a person who provides an Electronic Communications Network or an Electronic Communications Service".
RCP	Retail RCP. This term was first defined in the OTS Industry Process to define those RCPs who provide services at the retail level to end-users, both consumer and business.
MAP	Managed Access Provider This is a commonly used term within the UK telecoms industry to refer to parties who provide integrations services to RCPs but are not themselves RCPs.
Error Code	The code that will be returned in a synchronous message as part of the Hub's validation of incoming messages.

Abbreviation / term	Meaning / definition
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.

2 TOTSCo Hub integration specification

The TOTSCo Hub provides the mechanisms to deliver messages from one party to another in an environment where it is impractical for all parties to talk to each other directly. The parties will be RCPs or their agents such as MAPs.

The analogy of a post office is appropriate as TOTSCo are the agent who will accept a sender's message and will be responsible for delivering it on their behalf to the intended recipient. Senders do not need to find a way to deliver the message directly. In architecture parlance, this is a Hub and spoke mechanism as opposed to point to point.

Any messaging system requires standards to ensure interoperability, and to that end all messages sent via the TOTSCo Hub will be represented in JSON format and delivered using REST APIs.

Messages are made up of:

- an envelope containing the delivery data needed for the TOTSCo Hub to route the message to the correct destination, including a return address for replies and failures,
- and a message body.

The Hub does not need to know anything about the message body – that information is only for the sender and recipient to know and understand.

2.1 Letterbox API specification

The TOTSCo Hub letterbox API specification defines how messages will be sent to and received from the TOTSCo Hub. The API specification is separate from the documentation of the message formats for the industry processes (e.g., OTS), as the Hub does not need to know anything about the message format itself. The Hub acts on a routingID which may or may not be related to the message format. Examples of current routing IDs are described in §4 of this document.

The requirement is that both the TOTSCo Hub and the TOTSCo Hub users (the RCPs) all implement the same API specification. This makes it simpler to implement the messaging protocols in a uniform way, as well as supporting the ability to perform peer to peer testing.

A sample definition of the letterbox API specification can be found at the following URL:
https://app.swaggerHub.com/apis/TOTSCO/letterbox/0.4.0

The letterbox provides a mechanism to deliver a message via the TOTSCo Hub to an identified recipient. The TOTSCo Hub will only process the message envelope, to understand what/who it is for and to be able to process it correctly and will not process the message body (other than to pass it on to the recipient).

For example, certain types of messages sent to the Hub will be subject to a delivery timer with backoff and fallout policies. The policies for delivery messages will be industry-agreed and will be defined in the Hub and not specified by the sender.

Please note, the letterbox API specification is not specific to any single messaging or industry process. 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; and where peer to peer interfaces and mechanisms (e.g., email, SFTP, etc) would be impractical or lack the security or functionality that the Hub provides.

2.1.1 The letterbox post API Interface

The letterbox post API takes a JSON message from an authorised source and delivers it to an identified destination. The information needed to route that message is defined within the message envelope contained within the JSON message.

In summary:

- Messages are delivered using a "push push" model – i.e., the source RCP will push a message to the TOTSCo Hub, and the Hub will onwards push the message to the destination RCP.
- Message exchange between RCPs via the TOTSCo Hub is asynchronous, so the "push push" model applies to both requests in one direction and responses in the other direction.
- Messages will be pushed using https post with TLS v1.3.
- OAuth2 token will used be used to authorise the sender of each https request to the recipient.

When the TOTSCo Hub receives a message from an RCP (or a MAP), the OAuth2 credentials of the sender of the message will be matched against the source information in the envelope to ensure the message originates from an authorised sender and is not being spoofed.

Similarly, when an RCP (or a MAP) receives a message from the Hub, they will also check that the OAuth2 credentials to ensure the message originated from the Hub and is not being spoofed.

2.1.2 URI format

The API URI format provided by the TOTSCo Hub, and each RCP (or MAP) 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 RCP will need to provide their FQDN (possibly for their chosen MAP) as part of their Hub endpoint configuration.	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	n.n Initially "1.0"

URI Element	Description	Format
	Hub. If a new version is introduced, the previous versions will remain in service for compatibility with existing processes.	

Each RCP/MAP will need to provide their FQDN values and are encouraged to use similar naming conventions as the TOTSCO HUB environments.

2.1.3 API details

Field	Value
API Name	TOTSCo-LetterBoxAPI
Context	letterbox
Version	1.0
Resource	post
Transport Level Security	https, TLS 1.3
Port	The TOTSCo Hub will expose their API using the standard port 443 for https. It is recommended that RCPs / MAPs 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 Version Control

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.

e.g. If there are some minor changes, API version will be upgraded from version 1.0 to 1.1 and for any major changes the API version will be upgraded from version 1.0 to 2.0.

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 envelope contains the addressable information needed to identify and authenticate the message's originator – the "source", and the intended recipient – the "destination".

The envelope also contains a “routingID” which the Hub will use to determine how and where to send the message. Every message specification will define how the routingID should be populated. The Hub will also use the routingID to determine the delivery policy for the message.

Finally, there is an array element called “auditData” that must be used to provide information to TOTSCo for reporting to Ofcom and industry, where defined in each industry process.

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": "RBCD",
         "correlationID": "XYZ987"
      },
      "destination": {
         "type": "RCPID",
         "identity": "RCBX",
         "correlationID": "ABC123"
      },
      "routingID": " residentialSwitchMatchFailure",
      "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 OTS messages.	String	Required

JSON element	Description	Format	Notes
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 value.	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. 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. In that case the correlationID will be the value that was sent by the original sender of the message – i.e., it is being reflected to them.	String	Required /Optional
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 and published in the directory.	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	String	Required
auditData/value	The value associated to the named entity above.	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 §4 for the messages supported for the process described in this document.	String	Required

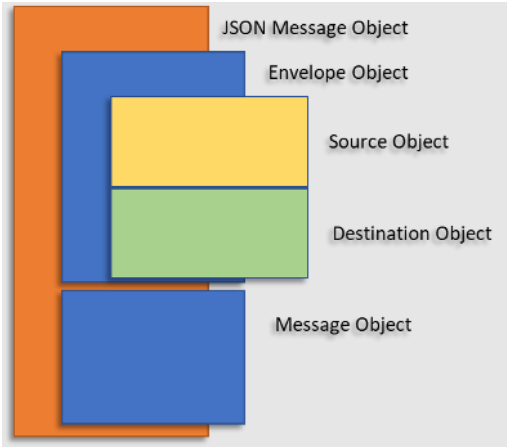


Figure 1 – Post Office JSON Message Envelope Structure

The container structure of a Post Office message is displayed above, the message object is separated from the envelope to allow changes in the content of either structure without affecting each other.

The reason every message must have a source correlation ID is that, as well as the recipient of the message being able to reply to you, in the event of a failure to deliver a message the TOTSCo Hub can return a delivery failure notification, even if the message was a response message. You can get delivery failures for response messages as well as request messages.

2.1.6 Auditing requirements

To allow TOTSCo to support the Ofcom reporting requirements for OTS, the audit Data in the envelope must be populated according to the following rules when sending OTS 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 by losing RCP.

```
{
   "envelope": {
      "source": {
         "type": "RCPID",
         "identity": "RBCD",
         "correlationID": "XYZ987"
      },
      "destination": {
         "type": "RCPID",
         "identity": "RCBA",
         "correlationID": "ABC123"
      },
      "routingID": "residentialSwitchMatchFailure",
      "auditData": [

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

2.1.7 Message formats

The TOTSCo Hub API specification defines the envelope of the JSON message only, and the API for sending the messages. These are the only parts of the JSON message the Hub will be responsible for understanding. Each message will also contain a message body, and this is the element that the recipient of the message must understand. Both parts together form the entire JSON message.

There may be hundreds of supported message bodies the Hub will deliver, those defined in this document in section §4 relate only to the process described by this document. Each industry process utilising the TOTSCo Hub (e.g., OTS) will have its own message format document that defines the message body.

Should any change be made to the API or envelope specification, that change will apply to every messaging process, so the API and envelope specification must be agnostic of those processes.

2.1.8 Letterbox responses

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 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 an RCP (or MAP), the 202 responses will be sent once the Hub has validated the message:

- Validated the security mechanism (OAuth credentials/API key, mTLS certificate)
- Validated the format of the message (e.g., valid JSON message)
- Validated the contents of the envelope.
- Verified that the sender is authorised to send on behalf of the defined source.
- Verified that the source RCP and destination RCP is valid.

If the message fails to be accepted by the API, various 400 errors may result subject to the OAuth processing of the API, or validation of the received message. For example, 401 – Unauthorised, or 403 – Forbidden for authentication errors, or in the event of a JSON format failure, error 400 – Bad request will be returned. To rectify the API error, please refer to the error description sent in the API response and resend the message with the appropriate changes.

The following table defines the list of http codes for the different errors during validation of the message.

HTTP Status Code	Exception Name / Code	Error Description
400	BAD REQUEST	The API cannot convert the payload data to the underlying data type. The data is not in the expected format. A required field has not been supplied. A validation error occurred.
401	UNAUTHORIZED ERROR	The request requires authentication and valid credentials were not provided

HTTP Status Code	Exception Name / Code	Error Description
403	FORBIDDEN ERROR	Valid credentials have been provided, but the authorisation level is not sufficient for the request
404	OPERATION NOT FOUND	The requested resource was not found
405	Method Not Allowed	The service does not support the HTTP method (e.g., POST, GET)
429	Too Many Requests	Hub exceeded the quota. You can access API after YYYY-MMM-DD xx: xx:xx+xxxx UTC. Note: The Hub has a limit of handling 67K incoming messages per minute. Any messages received beyond the limit will be rejected with the error code. The date and time mentioned above will be one minute after first message arrives.
500	INTERNAL SERVER ERROR	An unexpected error has occurred while processing the request
501	Not Implemented	The HTTP method is not supported by the server and cannot be handled.
502	Bad Gateway	The server got an invalid response while working as a gateway to get the response needed to handle the request.
503	Service Unavailable	The server cannot handle the request for a service due to temporary maintenance.
504	Gateway Timeout	The server is acting as a gateway and cannot get a response in time for a request.

The synchronous acceptance of the message must extract the source element as a minimum to determine if the sender is valid and authorised to be sending messages into the Hub. An invalid source will result in a 401 or 402 error.

The following table defines the list of http codes and error codes the Hub will generate in the event of a validation error before it has accepted the message for delivery. Please note that the validation will be done in sequence one at a time and the error response will be sent accordingly.

No.	Validation	http code	Error code
1	oAUTH2 token validation	401	NA
2	API Key validation	401	NA
3	mTLS Digital certificate validation	401	NA
4	Validate JSON Message Object structure	400	NA
5	Validate Message Envelope attributes	400	NA
6	Validate if Source Type is valid	400	9002
7	Validate if Source RCPD Id is valid	400	9003
8	Validate if Source RCPD ID account status is valid	403	9003
9	Validate if Destination Type is valid	400	9000
10	Validate if Destination Id is valid	400	9001

No.	Validation	http code	Error code
11	Validate if Destination RCPD ID account status is valid	403	9001
12	Validate Source type and ID permitted from originating location	401	9004
13	Validate if Routing Id is mapped to the Source RCP	400	9010
14	Validate RoutingID	400	9012

For the error responses, the following JSON will be returned, there will be no envelope.

```
{
    "errorCode": "<errorCode",
    "errorText": "<errorText "
}
```

The content of this message is as follows.

JSON element	Description	Format
errorCode	A numeric description of the specific error as defined in table above	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
message	A definition of the error	String
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	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 RCP Id.	{ "errorCode": "9003", "errorText": "Unknown or invalid source Id." }

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

400	Bad Request	Unknown or Invalid destination RCP Id	{
			"errorCode": "9001",
			"errorText ": "Unknown or Invalid destination."
			}

400	Bad Request	No routingID is mapped with Source RCP	{
			"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	Access failure for API: /directory/v1.0, version: v1.0 status: (900901) - 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": "Access failure for API: /directory/v1.0, version: v1.0 status: (900901) - Invalid Credentials. Make sure you have provided the correct security credentials.”
			}

401	Unauthorized	Access failure for API: /directory/v1.0, version: v1.0 status: (900900) - 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 Error	Unknown or invalid source RCP Id account status.	{
			"errorCode": "9003",
			"errorText": "Source RCPD ID account status is not valid."
			}

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

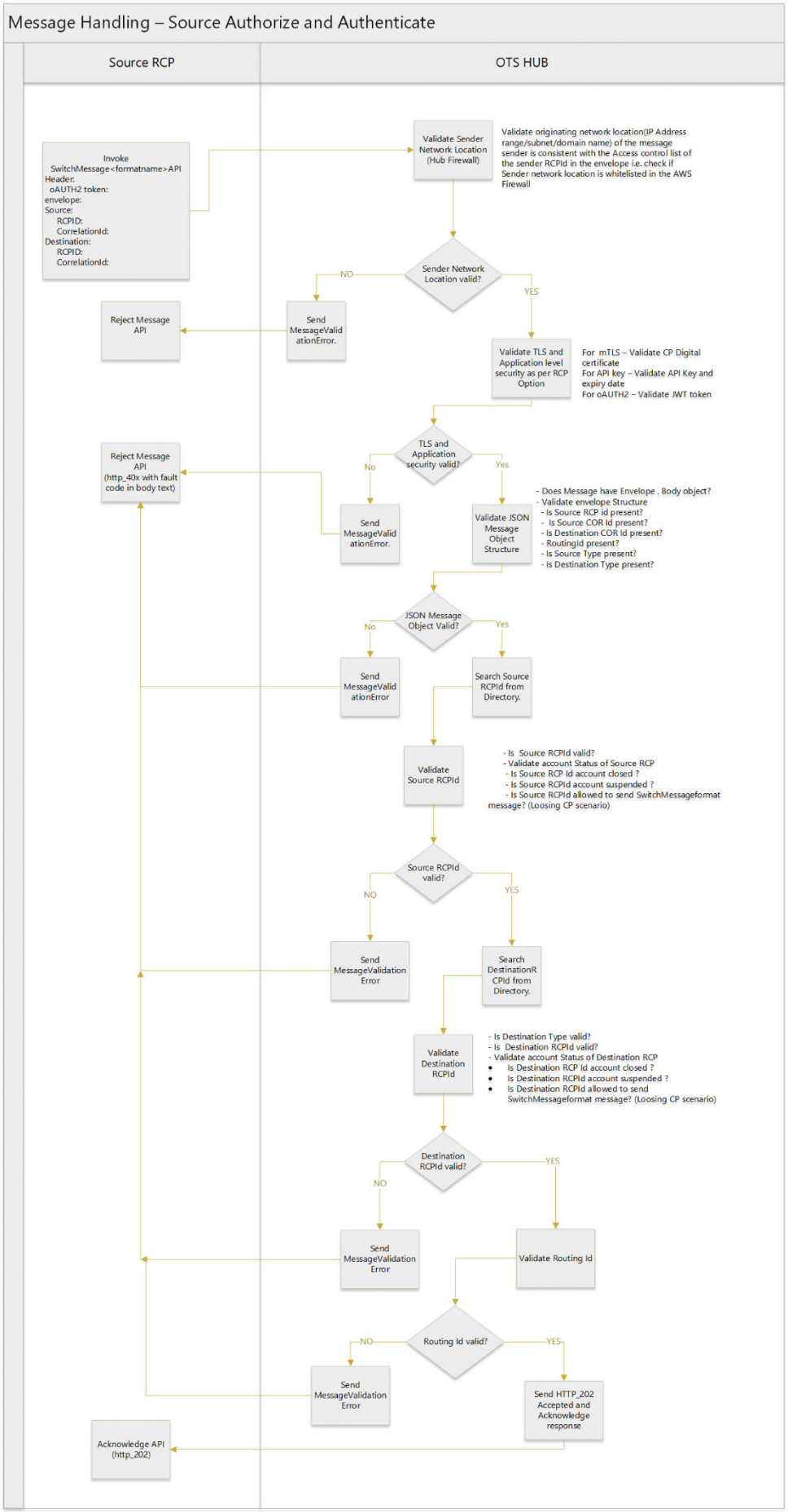
404	Not Found	No matching resource found for given API Request	{
			"code": "404",
			"type": "Status report",
			"message": "Runtime Error",
			"description": "No matching resource found for given API Request"
			}

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"
			}

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


Message Handling – Source Authorize and Authenticate
Source RCP
OTS HUB
Invoke SwitchMessage<formatname>API
Header:
oAUTH2 token:
envelope:
Source:
RCPID:
CorrelationId:
Destination:
RCPID:
CorrelationId:
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 Location valid?
NO
YES
Send MessageValidationError.
Reject Message API
Validate TLS and Application level security as per RCP Option
For mTLS – Validate CP Digital certificate
For API key – Validate API Key and expiry date
For oAUTH2 – Validate JWT token
TLS and Application security valid?
No
Yes
Reject Message API (http_40x with fault code in body text)
Send MessageValidationError.
Validate JSON Message Object Structure
- Does Message have Envelope , Body object?
- Validate envelope Structure
- Is Source RCP id present?
- Is Source COR Id present?
- Is Destination COR Id present?
- RoutingId present?
- Is Source Type present?
- Is Destination Type present?
JSON Message Object Valid?
No
Yes
Send MessageValidationError
Search Source RCPId from Directory.
Validate Source RCPId
- Is Source RCPId valid?
- Validate account Status of Source RCP
- Is Source RCP Id account closed ?
- Is Source RCPId account suspended ?
- Is Source RCPId allowed to send SwitchMessageformat message? (Loosing CP scenario)
Source RCPId valid?
NO
YES
Send MessageValidation Error
Search DestinationRCPId from Directory.
Validate Destination RCPId
- Is Destination Type valid?
- Is Destination RCPId valid?
- Validate account Status of Destination RCP
Is Destination RCP Id account closed ?
Is Destination RCPId account suspended ?
Is Destination RCPId allowed to send SwitchMessageformat message? (Loosing CP scenario)
Destination RCPId valid?
NO
YES
Send MessageValidation Error
Validate Routing Id
Routing Id valid?
NO
YES
Send MessageValidation Error
Send HTTP_202 Accepted and Acknowledge response
Acknowledge API (http_202)

Asynchronous post office faults and messages

In the event of the post office being unable to deliver a message to its intended recipient, or if status update messages are sent because of a delivery policy, the post office 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": "RCBD",
            "correlationID": "ABC123"
        },
        "routingID": "messageDeliveryFailure",
        "auditData": [{
                name: "originalDestinationType",
                value: "RCPID"
            },
            {
                name: "originalDestination",
                value: "RMNK"
            },
            {
                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 originators 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.

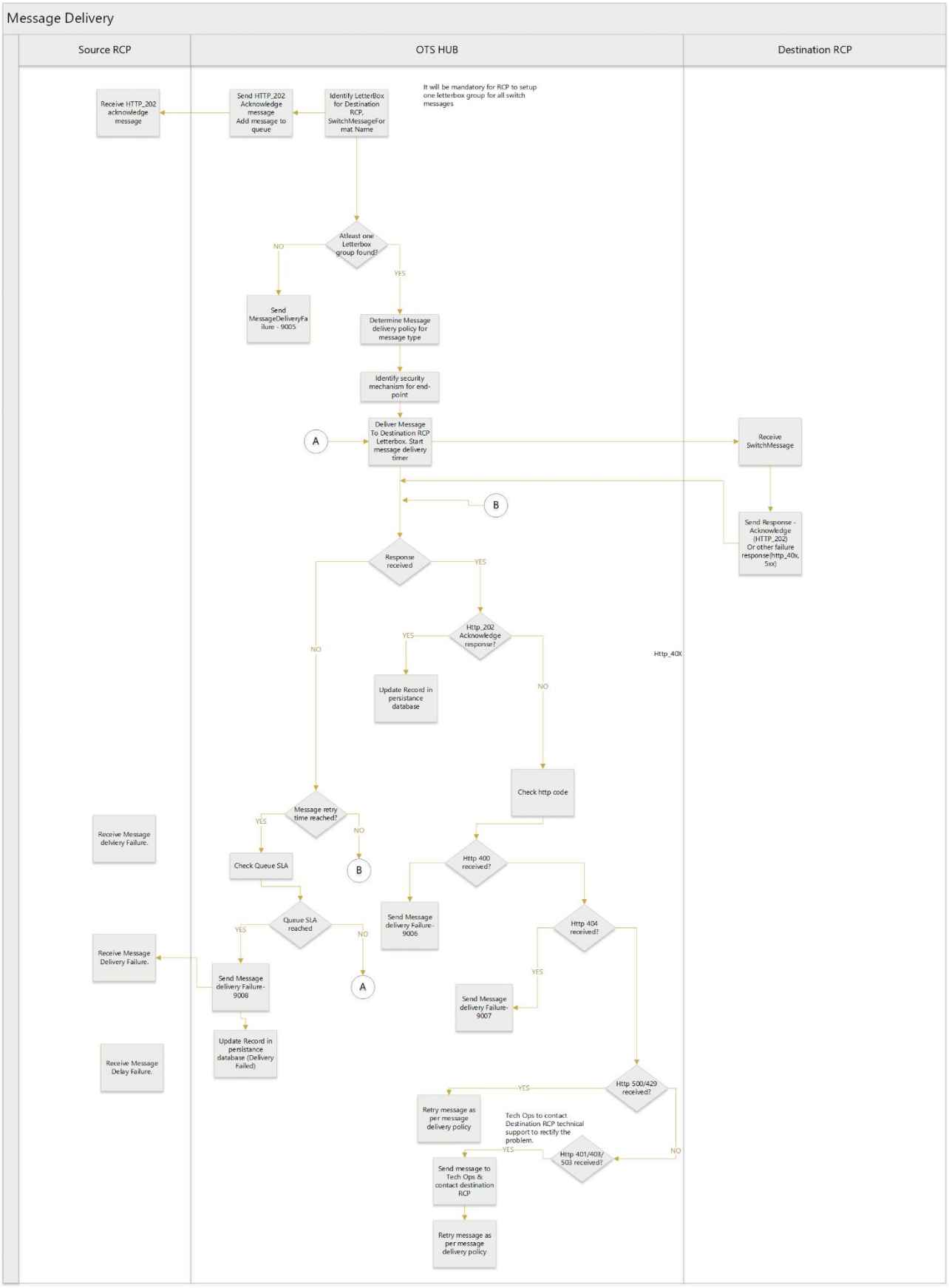
JSON element	Description	Format
messageDeliveryFailure	Container for messages that have failed delivery to the destination. These messages will be sent from the post office.	Object
Code	A number that represents the nature of the fault and can be used by the message originator to determine remedial action.	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. Values will include, “information”, “warning”, “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. Note: message sent if destination endpoints reject message and returns http_404 error response.	Failure
9008	Unable to deliver the message to the destination, timed out. Note: when destination connection is down and returns http_5xx error or timed out after multiple retries	Failure
9013	Unable to deliver message to the destination - Invalid API Key	Failure
9014	Unable to deliver message to the destination - API Key expired	Failure
9015	Unable to deliver message to the destination - Digital certificate invalid	Failure
9016	Unable to deliver message to the destination - Digital certificate expired	Failure

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


Message Delivery
Source RCP
OTS HUB
Destination RCP
It will be mandatory for RCP to setup one letterbox group for all switch messages
Receive HTTP_202 acknowledge message
Send HTTP_202 Acknowledge message Add message to queue
Identify LetterBox for Destination RCP, SwitchMessageFor mat Name
Atleast one Letterbox group found?
NO
YES
Send MessageDeliveryFa ilure - 9005
Determine Message delivery policy for message type
Identify security mechanism for end-point
A
Deliver Message To Destination RCP Letterbox. Start message delivery timer
Receive SwitchMessage
B
Send Response - Acknowledge (HTTP_202) Or other failure response(http_40x, 5xx)
Response received
YES
NO
Http_202 Acknowledge response?
YES
NO
Http_40X
Update Record in persistance database
Check http code
Message retry time reached?
YES
NO
Receive Message delivery Failure.
Check Queue SLA
B
Http 400 received?
Send Message delivery Failure- 9006
Queue SLA reached
YES
NO
Http 404 received?
Receive Message Delivery Failure.
Send Message delivery Failure- 9008
A
YES
Send Message delivery Failure- 9007
Update Record in persistance database (Delivery Failed)
Receive Message Delay Failure.
Http 500/429 received?
YES
NO
Retry message as per message delivery policy
Tech Ops to contact Destination RCP technical support to rectify the problem.
YES
Http 401/403/ 503 received?
NO
Send message to Tech Ops & contact destination RCP
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	1.0
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, currently 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. 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.	n.n (e.g., 1.0)
listType	This mandatory value specifies the entity list types to be included in the results.	Enumerated String

	For example, RCPID.	(e.g., RCPID)
identity	This is optional query string parameter. 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 other than “all” then directory entry for that identity of the specified list type will be returned.	String (e.g., RGXD)

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	Access failure for API: /directory/v1.0, version: v1.0 status: (900901) - 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": "Access failure for API: /directory/v1.0, version: v1.0 status: (900901) - 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


```
{
  "list": [
    {
      "listType": "RCPID",
      "identity": [
```

```
        {
          "id": "RGXD",
          "name": "Xenon Telecom",
          "processSupport": [
            {
              "process": "OTS",
              "status": "Active"
            }
          ],
          "resource": [
            {
              "name": "customerAssistURL",
              "type": "URL",
              "value": "https://xenon.com/OTS"
            },
            {
              "name": "salesAssistURL",
              "type": "URL",
              "value": "https://xenon.com/sales"
            }
          ]
        }
      ]
    }
  ]
}
```


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.	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 RCP 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. Current supported values are “OTS”.	String	Required

JSON element	Description	Format	Notes
processSupport/status	A value indicating the production status of this process. Current supported values are "Active" and “Suspend”. A full list of the values and their uses will be described in a future update.	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”.	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 messaged from an unknown source ID, or that message contains a signing key that you do not know, then you would request that single identity from the directory to update your local cache.

3 Security Implementation

The Hub will communicate with multiple RCP source networks and endpoints to sending and receiving messages. A multi-level and robust security mechanism will be implemented, so that both parties can ensure that the API requests come from a legitimate source and that the requesting client is authorized to send the requested data. The Hub will support two levels of security and authentication.

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 RCP provides their own Certificate.

For RCPs 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 RCP 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, an RCP can choose to either provide the certificate that the Hub should present when connecting to the RCPs own systems, or it can use the standard Hub certificate as they so choose.

CPs 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 an RCP can choose to use different security mechanisms per endpoint if they so wish.

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

- CPs / MAPs must ensure that the FQDN they configure to connect to the Hub is from a valid source, such as documentation retrieved directly from the Hub website.
- CPs / MAPs must ensure that the FQDN they configure for their endpoint(s) via the Hub RCP portal are valid for their organisation.
- The Hub will ensure the implementation respects the values configured via the Hub RCP portal, 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., RCP/MAP to Hub, or Hub to RCP/MAP) will use HTTPS. The Hub will expose its API using port 443. It is recommended that RCPs / MAPs do the same, but if they choose to use an alternative port as part of their endpoint configuration, this can be specified within the RCP portal.

The version of TLS supported will be 1.3. As all parties will be building new endpoints for implementation of OTS, 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

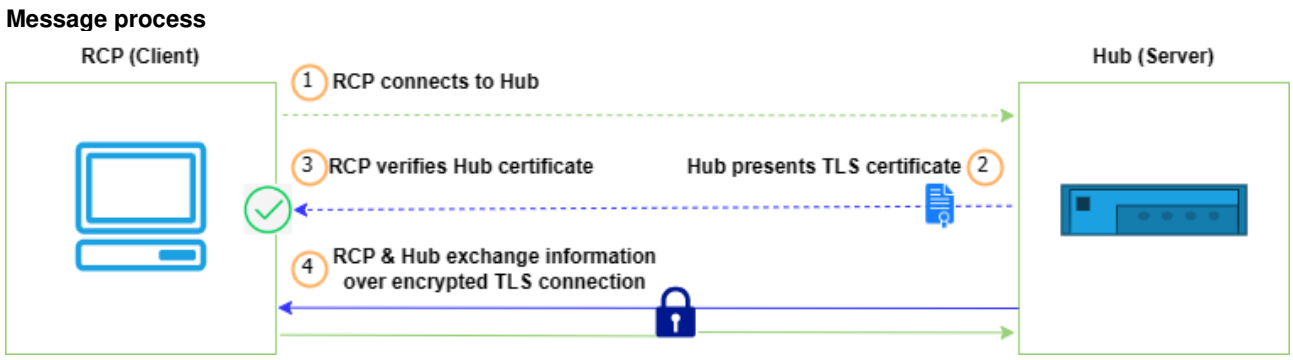
In Standard TLS for Client-Server setup, the server has a digital certificate, while the client does not. When a client makes a connection to the server, the digital certificate of the server will be presented back to the client. The client then verifies the presented digital certificate against the 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 RCP to TOTSCo Hub

The Hub will have a Root CA Signed RSA Certificate implemented to ensure CPs can trust they are accessing the TOTSCo Hub. The details of this certificate will be shared with RCP's 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 RCP'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 RCP's client Trust store does not have Public CA Root Certificates pre-configured, then the RCP will need to request the Hub's Root certificate. The TOTSCo TechOps team will share the Hub certificate over e-mail. This digital certificate will need to be configured by the RCP in their client system.



1. When a message is being sent to the Hub, the RCPs client application will connect to it by calling the Hub letter box API. The API URL will be shared with the RCP during onboarding.

2. When the Hub receives the connection from the RCP client, the Hub will present its TLS certificate to the RCP client.
3. The RCP client will verify the presented Hub's certificate with the CA Root certificate stored in its trusted certificate store.
4. If both digital certificates match, the TLS handshake is made, and the RCP client can send messages to Hub using the encrypted TLS connection.

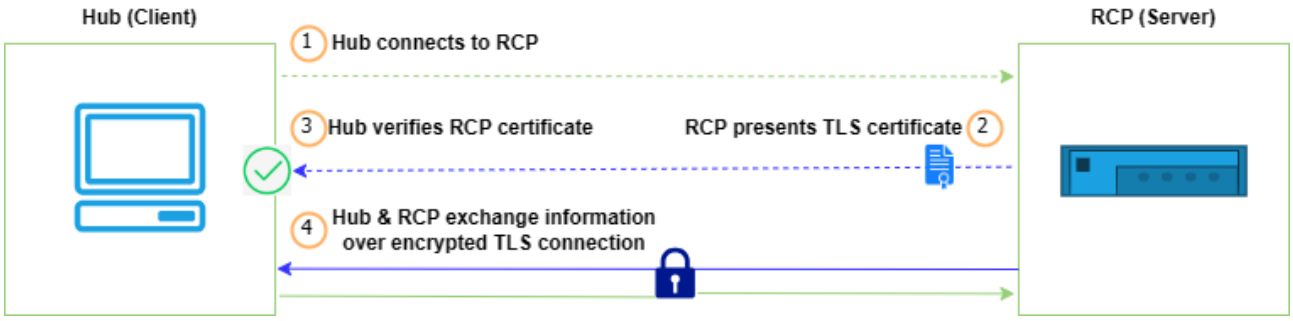
Connection from Hub to the RCP

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 RCP has implemented a digital certificate issued by a Public Certificate authority (CA) mentioned in Appendix B, then the RCP will not need to share their digital Certificate with TOTSCo. It is each RCP'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 RCP has implemented a digital certificate issued by a Private CA or is using a Self-Signed certificate, then the RCP 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 RCPs individual client record. TOTSCo will send an email notification regarding renewal of the RCP certificate before the expiry date.

Message Process



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

3.1.2 Mutual TLS

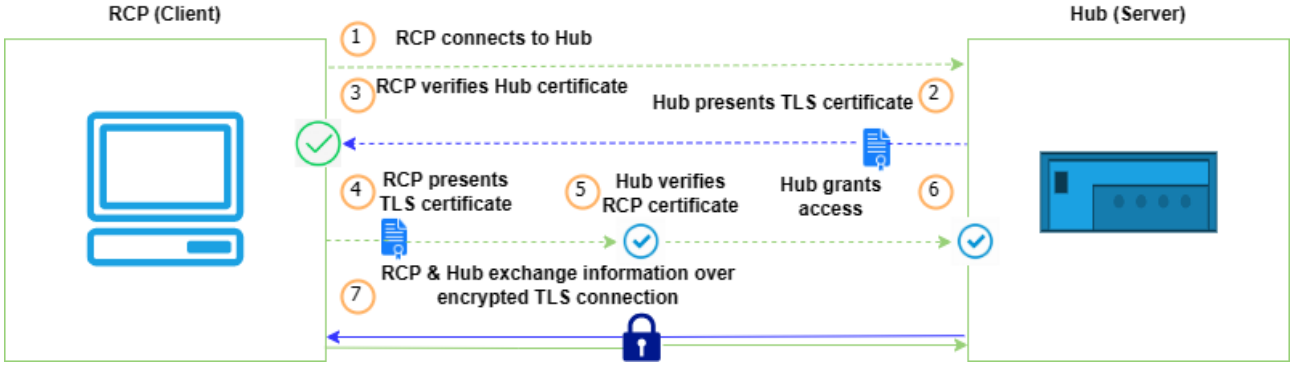
Mutual TLS or mTLS is a method for mutual authentication used in a Zero Trust security framework. Both the client and server have a certificate, which is used by both parties to authenticate each other. In Mutual TLS, the digital certificates of the Hub and the RCP endpoint server will be exchanged during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

RCP's 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 RCP endpoint..

Connections from RCP to the Hub
Connections from an RCP to the Hub

1. The TOTSCo TechOps Team will share the Hub's digital certificate and expiry date with each RCP through an email. This will be shared with the RCP when configuring the source network and if the RCP has opted for the mutual TLS security option. The digital certificate will be shared for each source network that the RCP needs configuring in the Hub. These can be the same or unique certificates as the RCP would prefer to implement.
2. The RCP will import the Hub's digital certificate into its Trusted Certificate Store. The RCP 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 digital certificate.
3. The RCP must share their digital certificate and the expiry date for each of the RCP 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 RCP digital certificate(s) into the Hub Servers.

Message Process



1. When a connection is being made to the Hub, the RCP will connect to the Hub by calling the Hub letter box API. When the Hub receives the connection from the RCP client, it will present the Hub's digital certificate back to the connecting client.
2. The RCP client will verify the certificate presented by the Hub against the one stored in the RCPs trusted certificate store.
3. If the Hub certificate is valid, the RCP client will present its digital certificate to the Hub.
4. 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 RCP client.
5. The RCP client will now be able to send messages through the Hub API within the encrypted TLS connection.

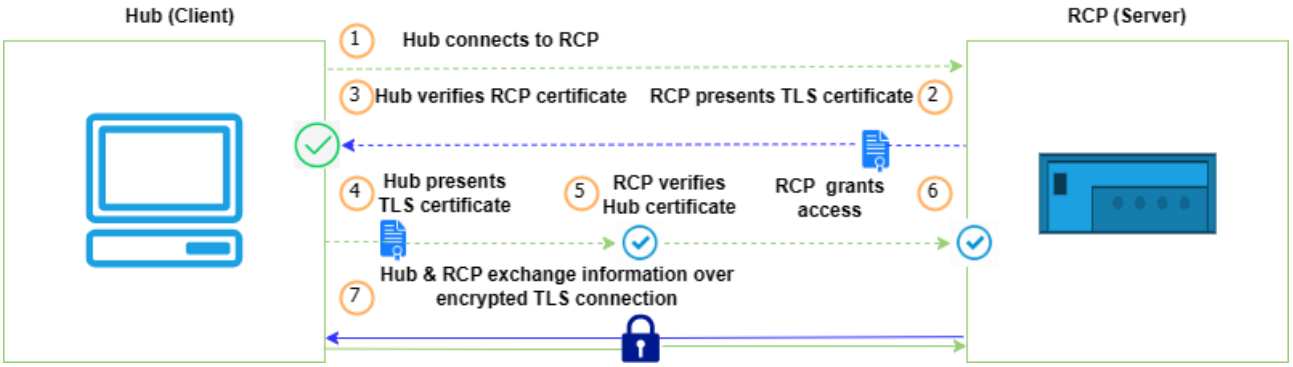
Maintaining validity of the digital certificate.

- The Hub TechOps team will provide the Hub's new digital certificate to each registered RCP, 30 days prior to expiry. The new certificate can be imported into the RCP 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 RCP should remove any expired certificates once the new one is being used.
- All registered RCP's should send any new digital 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 RCP

1. The Hub TechOps Team will share the Hub's digital certificate and expiry date with RCP via email. This will be done when the RCP configures the destination endpoint for receiving the messages from the Hub and triggers if the RCP has opted for mutual TLS security option.
2. The RCP must import the Hub's digital certificate into its Trusted Store. The RCP 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 RCP will need to share their digital certificate and expiry date with the Hub TechOps team when setting up the RCP destination endpoints. The certificates will need to be shared for each destination endpoint that RCP will be configuring in the Hub. The certificate should have a minimum 1-year validity.
4. The TOTSCo TechOps team will import the RCP digital certificate(s) into the Hub's Trusted Certificate Store. The TechOps team will give the digital 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 RCP endpoint, the Hub will connect to the endpoint API. The API URL will have been shared by RCP at the time of onboarding and setting up the endpoint in the Hub.
2. When the RCP endpoint server receives the connection from the Hub, it will present the RCP certificate stored, to the Hub.

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

Maintaining validity of the certificate.

1. The Hub will auto trigger reminder email communication to RCP a month prior to RCP destination endpoint or Hub's certificate expiry.
2. The Hub TechOps team will provide the Hub's new digital public certificate to the RCP 30 days prior to expiry.
3. An RCP must send new digital certificates through to the Hub TechOps team ideally 30 days prior to expiry in order to ensure there is no disruption to the CPs 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 RCP 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 an RCP/MAP and the TOTSCo Hub.

Inbound Communication from RCP to the Hub:

Registered RCPs 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 an RCP OAuth 2.0 supporting client / application as an intermediary between the RCP API systems and the TOTSCo OAuth 2.0 server.

Each RCP will be provided the following information:

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

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

The RCP 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 3.3.1 for the process to generate access token.

Once the authentication process has successfully completed, the RCP 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 RCP:

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

To support the OAuth2 inbound authorisation mechanism, each registered RCP 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 RCP specified endpoints that are configured to receive messages from the TOTSCo Hub.)

If an RCP is choosing to use the services of a MAP (Third party integrators) for their connection to the Hub, the RCP will need to configure their endpoints to be the MAP connection points, and they will need to inform TOTSCo which MAP they have selected, so that this can be configured within the Hub and messages successfully relayed through their chosen MAP.

The Hub will securely store each registered RCP clientid and secret key, token key generation webservice URL in the Hub platform. When a message needs to be pushed to a receiving RCP 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 RCP endpoint. The RCP 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.	String	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 RCP. 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 RCP to Hub:

For inbound communication between an RCP source network and the TOTSCo Hub, an API Key will be provided to the RCP by Hub TechOps team. The RCP will need to store the API Key securely. The RCP will be provided with an API Key for each of the source locations specified by the RCP 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 RCP 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 RCP 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 RCP 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 RCP can change the keys over. The RCP 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 RCP 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 RCP:

Registered RCPs 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 RCP 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 RCP in the Hub. Where an RCP has taken the service of MAP for their message journey, the RCP can configure their end points to the MAP connection points and provide the API Key details for the MAP endpoints. The Hub will store the API Key in its secure database.

Before making a connection to a RCP'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 RCP endpoint.

Sample Format:

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

where <API_key_value> is the RCP provided API key for the destination endpoint.

The API key provided by RCP should have a validity of 6 months and the RCP will have to ensure that the API key tokens are not expired. The RCP 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 Routing ID Summary

The TOTSCo Hub will be capable of routing messages of any kind from many different industry processes. Below is a sample list of the current known and proposed routing IDs.

The TOTSCo Hub will allow configuration of adding new routing IDs as and when required.

Gaining Provider	**Losing Provider**	**Post Office**
residentialSwitchMatchRequest	*residentialSwitchMatchConfirmation*	*messageDeliveryFailure*
	residentialSwitchMatchFailure	
residentialSwitchOrderRequest	*residentialSwitchOrderConfirmation*	
	residentialSwitchOrderFailure	
residentialSwitchOrderUpdateRequest	*residentialSwitchOrderUpdateConfirmation*	
	residentialSwitchOrderUpdateFailure	
residentialSwitchOrderTriggerRequest	*residentialSwitchOrderTriggerConfirmation*	
	residentialSwitchOrderTriggerFailure	
residentialSwitchOrderCancellationRequest	*residentialSwitchOrderCancellationConfirmation*	
	residentialSwitchOrderCancellationFailure	

5 Appendix A – Messaging version control

All messaging specifications that will be used by industry over the TOTSCo Hub will have the ability to define their own requirements for version control. However, some basic principles for versioning are outlined below for guidance in the event a message format is changed, to help define a standard process for versioning.

Normal practice for APIs is to use a version number in the URI, and indeed the letterbox API's do have such a version number so that if that API was to functionally change then a new version can both be created and supported.

With messaging you can go one of two ways, either include a version number within a message or rename the message itself.

In a distributed environment, it is important to know what the party you are sending messages to supports. So, the key decision factor in determining the versioning approach is how to inform the sender what format they can use to communicate with the recipient, as in all cases the recipient is the gating factor.

To version control a message format without changing its name would require another level of information within the Hub to relate the version of the message and then logic on both the sender and receiver side to interpret the message in specific ways depending on that version and its contents.

Taking the approach of simply introducing a new message format name makes this much simpler, no changes needed to the Hub, and a much clearer communication to the builder and processor of that message what to do with it right up front. This is therefore the recommendation for how new versions of messages should be managed within processes using the Hub.

The next question relates to what drives a message format version change. The consideration here is where a change can be considered minor, adding optional elements to a message for example, or major, renaming an element.

Any system parsing JSON messages should discard elements they are not expecting, most if not all JSON parsers support that capability and was one of the main reasons for choosing JSON as the messaging standard.

Consider a switch match request requires adding a new element called clientContactNumber, that can be considered a minor, non-destructive change and the element made optional. Consumers would update their systems when they are ready to make use of that element, but it does not break the process or messaging if you have not yet added support for it and simply ignore the element.

Now consider we rename the services element in the matching request (a bad thing to do, but as an example), and we change it to serviceList. That is clearly a major, destructive change. In this case the message format name should be changed from residentialSwitchMatchRequest to residentialSwitchMatchRequestv2 as the structural format is different and therefore the processing of existing consumers of that message would break.

As the directory holds the supported message formats, it will be incumbent on the message sender to ensure that they use the appropriate format for the intended recipient. If provider A supports v1 and v2 of the match request but provider B only supports v1, then when provider A is creating a match request to BT it would know it had to create the message in the v1 format as v2 isn't yet supported by provider B.

Through industry consultation, process owners will agree on the message format changes, the most appropriate way of updating them, and how to control the versions.

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