TOTSCo

TOTSCo Hub API Specification

A guide for developers

Version 1.0
21 July 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 (CPs) who are responsible for the technical implementation of the communication between that CP and the TOTSCo Hub.
- Representatives of Tech Mahindra, the vendor chosen by TOTSCo to implement the Hub.

1.1 Communications providers and Managed Access Providers

The TOTSCo Hub will initially provide services for retail CPs 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 CPs who may not be the retailers. So, this document used the generic term of CP.

Note that retail CPs may choose to engage an agent, such as a managed access provider (MAP). Any reference in this document to a requirement against a CP 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

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 CPs 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 Communication Provider This term was first defined in the OTS Industry Process to define those CPs 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 CPs but are not themselves CPs.
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.

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 CPs 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 CPs) 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 CP will push a message to the TOTSCo Hub, and the Hub will onwards push the message to the destination CP.
- Message exchange between CPs 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 a CP (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 a CP (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 CP (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. The TOTSCo Hub FQDN values are listed in the next table. Each CP 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 Hub. If a new version is introduced, the previous versions will remain in service for compatibility with existing processes.	n.n Initially "1.0"

The FQDN values used by the TOTSCo Hub will be:

Environment	FQDN
SIT (Simulator)	sit.otshub.totsco.co.uk
Pre-production	preprod.otshub.totsco.co.uk
Production	prod.otshub.totsco.co.uk

Note the use of the different domain of totsco.co.uk – this is intentionally different to the totsco.org.uk domain used for TOTSCo's public facing website and email addresses.

Each CP/MAP will need to provide their FQDN values and are encouraged to use similar naming conventions for SIT, pre-production and production 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 CPs / 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": "RCBA",
         "correlationID": "ABC123"
      },
      "routingID": " residentialSwitchMatchRequest",
      "auditData": [
         {
            "name": "auditFieldName",
            "value": "auditFieldValue"
         },
         {
            "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

JSON element	Description	Format	Notes
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
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	String	Required

JSON element	Description	Format	Notes
	being sent. Please refer to §4 for the messages supported for the process described in this document.		

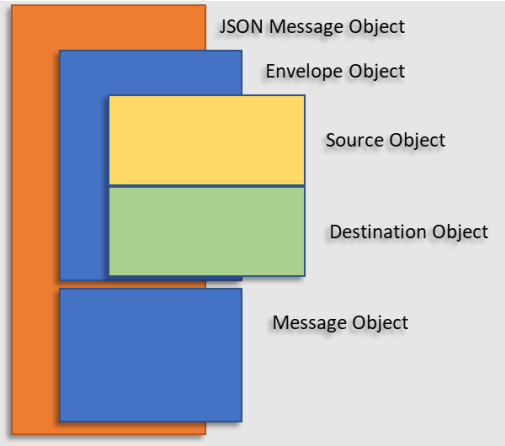


Figure 1 – Post Office JSON Message Envelope Structure

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 fault message sent through the Hub.

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

         {
            "name": "faultCode",
```

```
            "value": "9001"
         }
      ]
   },
   "residentialSwitchMatchFailure": {
      "faultCode": "101",
      "faultText": "failure to match with the supplied information".
   }
}
```


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

- Validated the OAuth credentials.
- 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 CP and destination CP 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.

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

HTTP Status Code	Exception Name / Code	Error Description
		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
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	OTS Hub exceeded the quota. You can access API after YYYY-MMM-DD xx: xx:xx+xxxx UTC. Note: 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
503	Service Unavailable	The server cannot handle the request for a service due to temporary maintenance.

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.

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

```
{
    "errorCode": "9002",
    "errorText": "Unknown or invalid source type. "
}
```

The content of this message is as follows.

JSON element	Description	Format
errorCode	A numeric description of the specific error	Integer
errorText	A description that represents the nature of the error and can be used by the message originator to determine remedial action.	String

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	Validate JSON Message Object structure	400	NA
3	Validate Message Envelope attributes	400	NA
4	Validate if Source Type is valid	400	9002
5	Validate if Source RCPD Id is valid	400	9003
6	Validate if Source RCPD ID account status is valid	403	9003
7	Validate if Destination Type is valid	400	9000

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

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

Code	Message	Description	Sample response

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

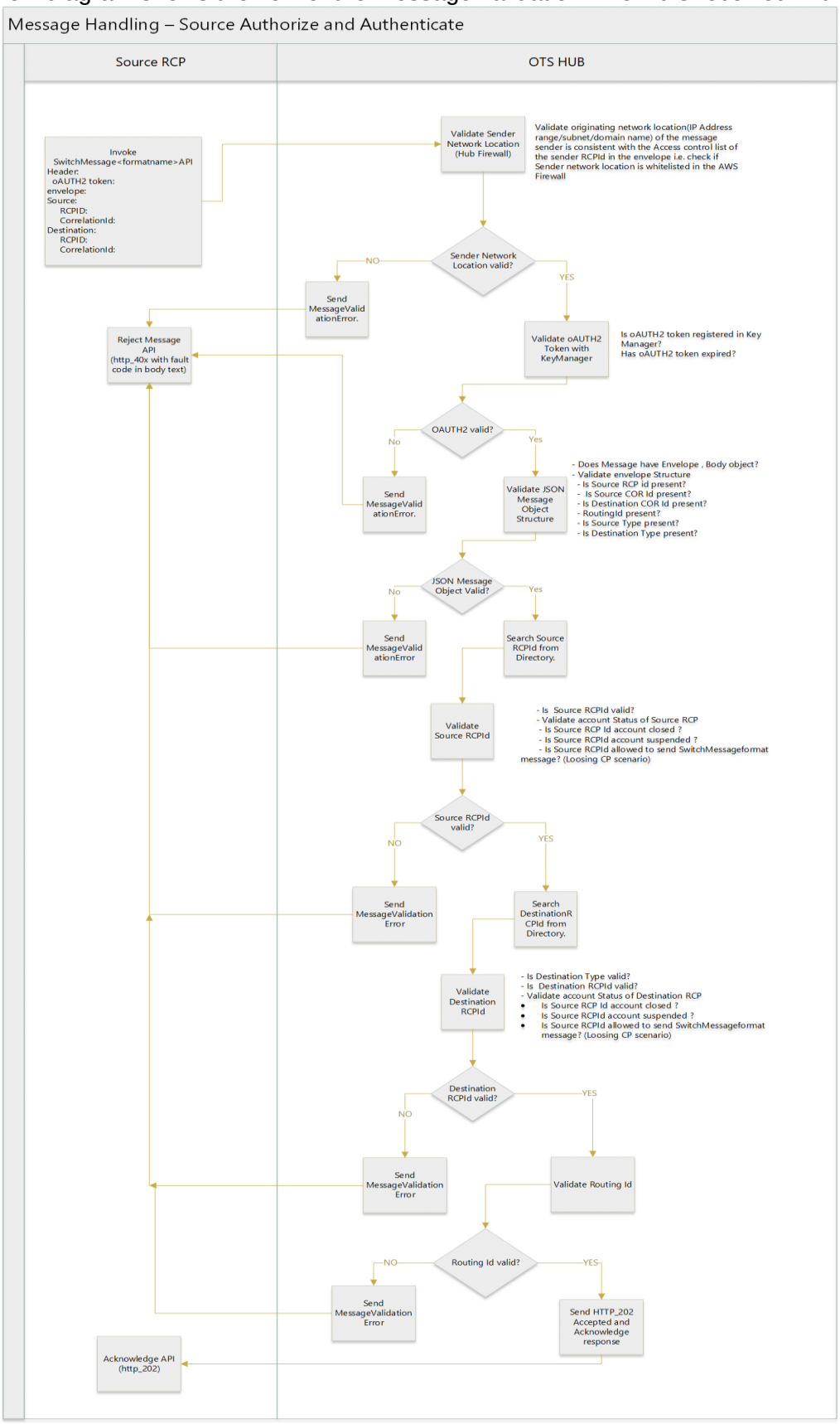
401	Unauthorized	Access failure for API: /directory/v1.0, version: v1.0 status: (900901) - Invalid Credentials. Make sure you have provided the correct security credentials.
		Note: code 900901 is an internal process code and not an error code.
			<pre>{
			  "code": "900901",
			  "message": "Invalid 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."
			}</pre>

400		Unknown or missing destination Type	<pre>{
			  "errorCode": "9000",
			  "errorText ": "Unknown or missing destination Type."
			}</pre>

400		Unknown or invalid source Type.	<pre>{
			  "errorCode": "9002",
			  "errorText ": "Unknown or invalid source Type."
			}</pre>

400		Unknown or invalid routing ID.	<pre>{
			  "errorCode": "9012",
			  "errorText ": "Unknown or invalid routing ID."
			}</pre>

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



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"
            }
        ]
    },
    "postOfficeMessage": {
        "code": "9008",
        "text": " Unable to deliver the message to the destination, timed out.",
        "severity": "failure"
    }
}
```

The post Office Message 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 postOfficeMessage 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.

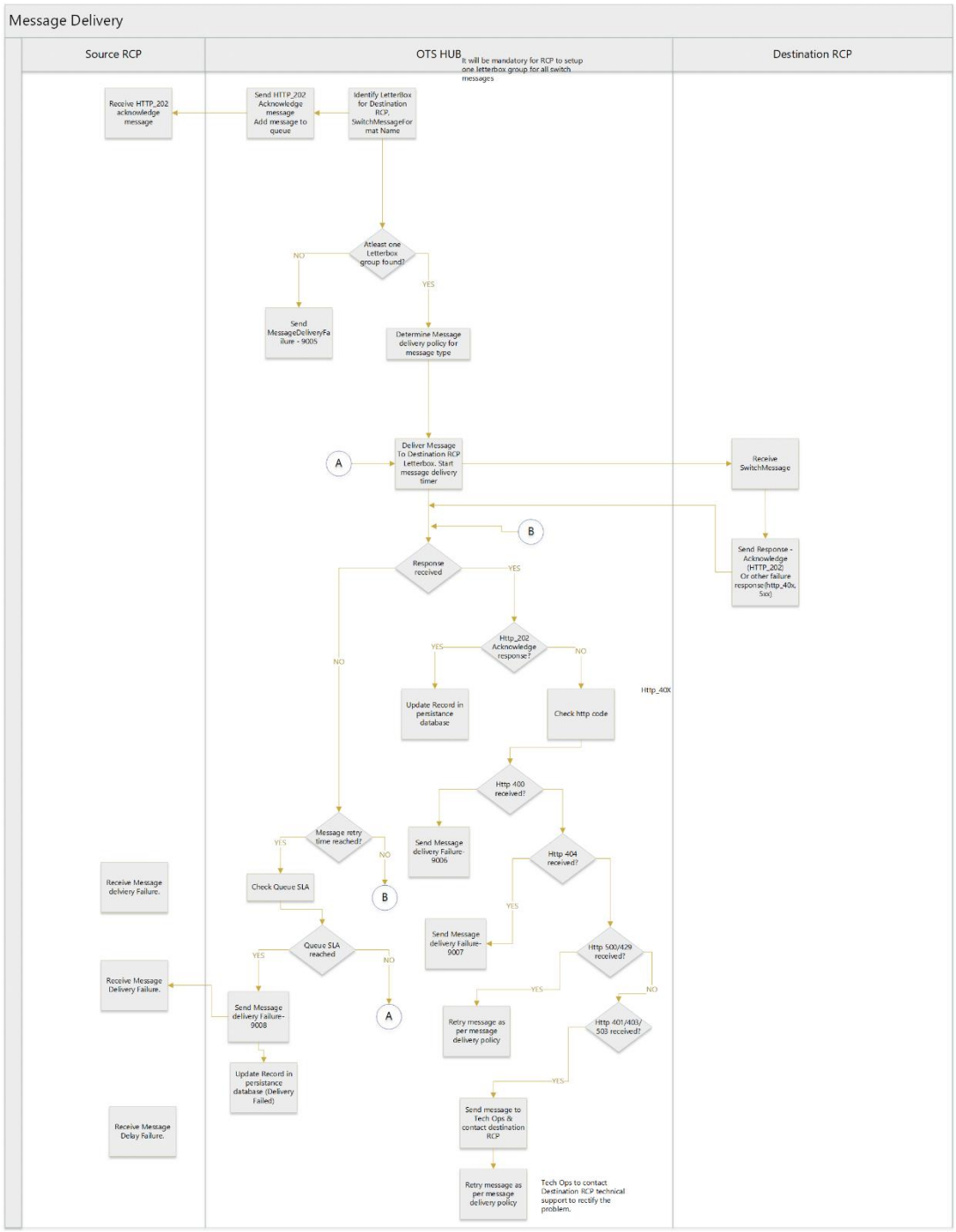
JSON element	Description	Format
postOfficeMessage	Container for messages from the post office	Object
Code	A number that represents the nature of the fault and can be used by the message originator to determine remedial action.	Integer
Text	A description of the associated response code	String

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

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



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	/letterbox
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	POST
Resources	/directory
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 CP on the hub
2. For all CPs on the hub, (a list type for all parties of that specific list type, or all parties documented on the Hub.)

2.2.2 For a specified RCP on the hub

To get the details of a specified RCP, 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}/letterbox/{version}/directory?list={listID}&identity={identityID}`

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	n.n (e.g., 1.0)

	by the TOTSCo hub. If a new version is introduced, the previous versions will remain in service for compatibility with existing processes.	
listID	This mandatory value specifies the entity types to be included in the results. For example, RCPID.	Text (e.g., RCPID)
identityID	This mandatory value specifies a specific identity to obtain the details for. If this value is specified, the listID MUST also be specified.	Text (e.g., RGXD)

Note: Please refer to the section 2.2.4 for the directory response structure

Response Code:

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

Code	Message	Description	Sample 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\"]), " }
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 For all RCPs on the hub

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

`https://{fqdn}/letterbox/{version}/directory?list={listID}&identity=`**`all`**

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)
listID	This mandatory value specifies the entity types to be included in the results. For example, RCPID.	Text (e.g., RCPID)
identityID	This value should always be set as "**all"**	Text

A JSON payload will be returned in the response as a part of API call.

Note: Please refer to the section 2.2.4 for the directory response structure

Response Code:

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

Code	Message	Description	Sample 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\"]), "
			}

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."
			}

2.2.4 Directory API Response Structure

a) For a specified RCP


```
{
  "directory": [
      {
          "listType": "RCPID",
          "identityList": [
              {
                  "id": " RGXD ",
                  "tradingName": "Xenon Telecoms",
                  "status": "live",
                  "processSupport": [
                      {
                          "process": "OTS",
                          "customerassistURL": "https://xyztelco.com/OTS",
                          "salesassistURL": "https://xyztelco.com/OTS"
                      }
                  ]
              }
          ]
      }
  ]
}
```


b) For all RCPs on the Hub


```
{
  "directory": [
      {
          "listType": "RCPID",
          "identityList": [
              {
                 "id": " RGXD ",
                 "tradingName": "Xenon Telecoms",
                 "status": "Live",
                 "processSupport": [
                     {
                        "process": "OTS",
                        "customerassistURL": "https://xyztelco.com/OTScustomer",
                        "salesassistURL": "https://xyztelco.com/OTS"
                     },

                 ]
              },
              {
                 "id": " RGXE ",
                 "tradingName": "XYZ Telecoms",
                 "status": "Suspend",
                 "processSupport": [
                     {
                        "process": "OTS",
                        "customerassistURL": "https://XYZtelco.com/OTScustomer",
                        "salesassistURL": "https://XYZtelco.com/OTS"
                     }
```

```
                ]
            },
            {
                "id": " RDNB ",
                "tradingName": "VPN Telecoms",
                "status": "Live",
                "processSupport": [
                    {
                        "process": "OTS",
                        "customerassistURL":
"https://VPNtelecom.com/OTScustomer",
                        }
                ]
            }
        ]
    }
]
}
```


Response elements of the JSON document will be as follows.

JSON element	Description	Format	Notes
directory	An array of the lists for the directory information	Object array	Required
directory/listType	The list type associated to the contained identities.	String	Required
directory/identityList	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/tradingName	Value to identify the trading name of the RCP organization	String	Required
identity/status	A value indicating the production status of this process. Current supported values are "live" and "suspend". A full list of the values and their uses will be described in a future update.	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
processSupport/customerassistURL	Website for CP to provide information on the OTS process.	String	Optional
processSupport/salesassistURL	website for CP agents to assist on the sales journey during the switching process.	String	Optional

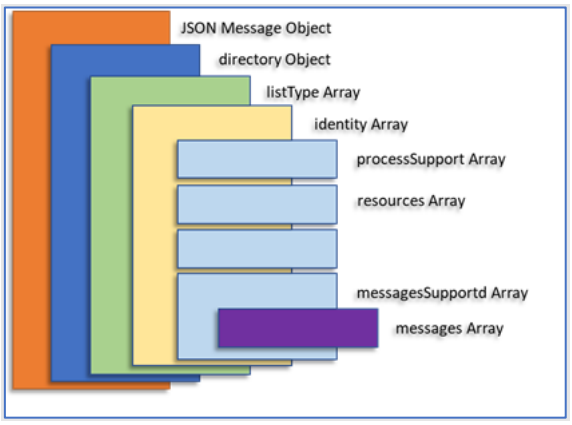


Figure 2 – Directory JSON Structure

The container structure of the directory is displayed above. How many identities and lists you receive in your result depends on your selection criteria.

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

3.1 Transport Layer Security (TLS)

Messages will be pushed from the sender to the recipient (e.g., CP/MAP to Hub, or Hub to CP/MAP using HTTPS. The TOTSCo Hub will expose its API using port 443. It is recommended that CPs / 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 TOTSCo CP 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 compatibility to work with version 1.2. The TLS will be a one-way TLS. For inbound communication (message coming into the hub), all registering CP's will be provided the TOTSCo hub public key. This will need to be installed into the CP certificate key store. The public key will be shared during onboarding or if the certificate key is changed / updated.

For outbound communication (messages being pushed to a CP), the TOTSCo hub will require the registering CP to supply its own TLS certificate, which TOTSCo will then install into the hub's certificate store. Registered CPs will need to provide the public key to TOTSCo at the time of onboarding or subsequently whenever the keys is changed, in order to maintain a successful and secure connection stream.

The CP server will require the implementation of a digital certificate issued by a certificate authority (CA). Ideally the certificate needs to be a trusted by a root CA so the TOTSCo hub can validate its authenticity.

The HUB will support permissible ciphers to the following values, for both server and client usage:

- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

The certificate created validates the CP, and its own service, so the FQDN should match the certificate specified address path.

There are some responsibilities to ensure this security is meaningful:

- CPs / MAPs must ensure that the FQDN they have configured for the TOTSCo Hub is based on a valid source, such as documentation retrieved directly from the TOTSCo website.
- CPs / MAPs must ensure that the FQDN they configure for their endpoint(s) via the TOTSCo CP portal are valid for their organisation.
- The private keys associated with digital certificates must be held securely by the owning party.
- TOTSCo will ensure their implementation respects the values configured via the TOTSCo CP portal, and that requests for changes are not accepted from unverified parties.

The TLS process ensures that the communication session between the TOTSCo hub and the CP is encrypted and any transmissions are secure between the two parties.

3.2 API Authentication and Authorisation

The API authentication process ensures that individual transactions within the communication stream are validated as genuine and identified to the specific sending CP and TOTSCo hub. The OAuth 2.0 protocol will be used to support this security process.

OAuth will provide token-based validation and authorisation for all communication between CP/MAP and TOTSCo Hub.

Inbound Communication from CP to TOTSCo Hub:

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

Each CP will be provided the following information:

- Client ID
- Secret key
- OAuth2 token generation webservice URL for each of the source locations that will be configured by the CP for sending messages to the HUB.

The Token generation URL, Client Id and Client Secret must be stored by the CP within the trusted database store.

The CP 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 CP system can post messages to the TOTSCo hub webservice. 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 ensure that messages are successfully delivered to the hub.

Outbound Communication from TOTSCo Hub to CP:

Once this mechanism is live, effectively the process above will work in reverse. The TOTSCo Hub OAuth 2.0 client will request an access token from the receiving CP's OAuth 2.0 server for validation. Upon successfully completing this, outbound messages will then be pushed to the CP platform.

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

- A deployment of an OAuth2.0 Server
- The creation of a client id for TOTSCo
- A generated secret key for TOTSCo
- A valid token key generation webservice URL. (This will need to encompass each of the CP endpoints that will be configured to receive messages from the TOTSCo hub.)

If a CP is choosing to use the services of a MAP (Third party integrators) for their connection to the TOTSCo hub, the CP 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 TOTSCo hub and messages successfully relayed through their chosen MAP.

The TOTSCo hub will securely store each registered CP clientid and secret key, token key generation webservice URL in the HUB secure database. When a message needs to be pushed to a receiving CP endpoint,

the TOTSCo hub will create a connection to the registered token key generation webservice URL, supplying the credentials of the client Id and secret key in order to generate a valid OAuth2 access token. This token will then be parsed within the https header of the message pushed, to the receiving CP endpoint, which will validate the credentials to ensure the identity and transaction is genuinely from the TOTSCo hub.

3.2.1 Process to request and generate OAuth2 Access Token from TOTSCo Hub

3.2.1.1 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_key:client_secret>	This will convert the passed **Client_Id** and **Client_Secret** into Base64-encoded.
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

3.2.1.2 Access Token Response

In a successful authorization grant type, the response will contain 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

3.2.1.3 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

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.

The Hub directory is where all message formats are listed that a recipient accepts – although not explicitly defined as the formats, they are routingID's, they serve the same function.

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.

End of Document