TOTSCO

The guidance in this bulletin has been superseded by guidance in bulletin 67

TOTSCo Bulletin No 66
Date: 22 July 2024
Subject: Guidance on Receiving HTTP Messages from the HUB

This bulletin has been written by the One Touch Switch Industry Process Group to provide additional guidance on handling receipt of messages from the TOTSCO hub, following issues experienced from testing.

Disclaimer: *This bulletin has been produced in good faith by industry participants, facilitated by TOTSCo, to assist other industry participants. It is not legal or regulatory advice nor approved and/or endorsed by Ofcom, ICO and/or TOTSCo. Communications providers may not rely on its contents and are responsible for their own regulatory compliance.*

TOTSCo
July 2024

Handling receipt of HTTP messages

As part of the implementation specifications for the TOTSCo hub, the letterbox API specification lays out a set of expectations for processing and acceptance of a message when delivered to a CP by the TOTSCo hub. This is a mirror specification as the TOTSCo hub is also required to follow the same standards.

Alongside the API specification, the One Touch Switch Message Delivery Policies v1.0 document specifies the SLAs for processing messages, and this is the area that is causing issues. The delivery policies details the response timeout period that the Hub will wait for a CP to respond before retrying delivery, with retry policies applicable depending on the type of message sent.

The API specification also describes how the post process is independent of the message content, and only the envelope is relevant to the onward deliver of that message.

Some background here is relevant to the design process. The JSON envelope message architecture is intended to be an open message standard for inter CP communications regardless of the industry process or message content. The hub design, and indeed to the letterbox API specifications were intended to be lightweight data processors allowing messages to be moved from one CP to another rapidly without over processing. This is further emphasized with the asynchronous message protocols where message response can be performed out of band and subject to SLAs appropriate to the individual process, and not encumber the API itself with the data processing in real time.

In practice, some CPs have gone beyond a basic level of validation on receipt of a message from the hub to the extent that the whole message content is being validated, and even processing the transaction contained within the message.

This overprocessing of the message on receipt has resulted in situations where the message is not responded to within response timeout SLA time required in the specification. The result of this is that the TOTSCo hub is resending the message to the letterbox and the CP is not properly dealing with the duplicate delivery of the message either.

This document will provide guidance on the best practices for the letterbox listener, and the expectations for its use and suggestions to CPs how they should handle messages that time out and are resent by the hub.

Guidance on letterbox API implementation
For any message sent to a CP using the letterbox API interface, there are a list of suggested do's and don'ts that can be used to optimise processing time and ensure the solutions is fit for further message applications in future.

Do's

1. Schema validation: It is reasonable that the schema of the JSON document can be validated on receipt of a message on the letterbox API. TOTSCo will only validate the schema up to the envelope, and not beyond, although they will validate the message as a whole is a validly constructed JSON document. CPs may include extended schema validation, aligned with the swagger definitions published in support of the OTS process. This may be extended in the future with support for other processes as well.
2. Envelope validation: The intent of the letterbox API was that only the envelope should be validated on receipt of the message to ensure it can be responded to. TOTSCo performs this on behalf of any messages it receives and rejects the message at the letterbox API if it knows the message cannot be onward delivered, or a response to the message could not be relayed to the originator.
3. Target a processing time of under 200ms for a response. This is well below the response timeout SLA but provides contingencies for lags or delays in internal processes, such as writing onward to queues or databases where real processing is performed.

Don'ts

1. Whole message validation: If the envelope is well formed, and the content valid, then any errors resulting from the body of the message can be dealt with either in response messages if a request message contained invalid content, or by discarding the message if the message send did not expect a reply, for example an acknowledgement contained an unrecognised SOR.

Although it is recognised that Asynchronous messaging does have weaknesses in some situations, for example being unable to respond to a message where no reply is expected, it is expected that extensive testing between CPs will identify faults and that in life errors requiring such real time validations should not be necessary.

It is acceptable if there are some key issues that do arise, such as unknown SOR that could benefit from a real time validation, but a change would also be required for CPs sending messages to be aware of rejections of this nature and to be able to handle those responses. There is currently no

requirement for this, and therefore these real time responses from the letterbox API are currently not a required function. Instead, industry has defined reporting processes for issues such as this that may result in changes to the specifications in future.

Handling Duplicate Message

Even with a lightweight Letterbox API, there can be situations beyond your control where you may not be able to process the message sent to you within the response timeout SLA and respond to the hub before it closes its connection.

For this situation, the following suggestion may help with your system architecture to ensure only a single message is processed, and the that end-to-end processes remain intact.

Firstly, it would be necessary to maintain a short-term cache of received message IDs, containing the sending RCPID and their correlation ID. Remember that different CPs could use the same correlation IDs so don't rely on them alone.

When a message is received, immediately check the cache for duplicates and, if it is not there, add it to the cache and begin processing the message. Once the message has been processed (onward delivered to your main system for processing) reply to the message with the appropriate 202 or other HTTP error code, and cache that with the message ID. If you were unable to deliver that response because the socket was already closed, you now have a copy of the information to respond if it is sent again.

If you receive a message and find that the RCPID and correlation ID are in your cache, simply reply with the response code and information you have cached, no further processing of the message would be required as you would already be processing the original message in your core systems.

If you find that the RCPID and correlation ID are in the cache but no response is present, then wait until it is, and then attempt to send the reply. You could put a timeout on this to avoid leaving a process ongoing beyond the SLA time knowing that the socket connection would have been lost anyway.

Using mechanisms like the above would mean that you only process a message once, and a reply is sent back to the sending system only once, regardless of how many times TOTSCo attempts to deliver the message.

This is not intended to be a how to guide, but to demonstrate the intent of the asynchronous message design and the mechanisms required to handle situations like socket closures when dealing with messages like this.

Recommendations

Consider your existing letterbox API designs and try to ensure they are as lightweight as possible. Long response times have knock on impacts to the TOTSCo hub as well, so it is in every body's interests to keep the end-to-end delivery mechanisms as light as possible.

Testing

Testing is simple to achieve by adding in test code to your letterbox listener to add random delays when processing messages, you should easily be able to create situations where messages exceed timeouts and test that you are handling the resending of duplicate messages adequately. Just remember not to put that random delay into production.

Clarification
CPs should not generate and send new messages to the hub with the same correlation ID for at least 12 days to avoid duplicate transaction detections resulting in messages being incorrectly responded to. The ideal would be that all correlation IDs are uniquely generated by a CP wherever possible.

Future Considerations
Consideration will be given to out of band response mechanisms for incorrect message content for messages that do not expect a response. Currently the requirement is that CPs report instances of non-compliance and they will aggregate these and assist with CPs not following the specifications. It may be helpful to be able to generate messages, notifications, in future for these situations. That will be discussed in future design groups after OTS has gone live.

System Startup Processes
Each CP should understand their own systems and ensure they are started up and available for processing transactions as soon as the socket listener interfaces is opened and is listening for incoming messages. For example, ensuring all database connections are established before receiving messages. This will help to ensure messages are processed within the response timeout SLA and avoid unnecessary duplicate transaction processing.

Implications of Slow Response Rates
As the HUB operates a FIFO (First-In First-Out) queue to ensure that messages are delivered in order to avoid race conditions, the HUB does not thread message delivery and operates strictly by sending a single message and then processing the next message on successful delivery receipt. For each CP there is one queue for residentialSwitchMatchRequests and one queue for all other message types.

The consequence of a slower response is that the longer you take to process a message receipt could result in queue building up at peak times as the Hub will only send one message at a time. As a residentialSwitchMatchRequest must be responded to in 30 seconds, any delay in processing your message via validation, or additional processing will result in a MessageDeliveryFailure response from the Hub with a fault code of 9008, without a delivery attempt being made. As illustrated below the total number of residentialSwitchMatchRequest messages types on your queue can be greater with a lower response time therefore avoiding a MessageDeliveryFailure being sent from the HUB.

Total Messages on Queue	Average Response Time (Ms)	Queued Message that will Timeout for residential SwitchMatchRequest
150	200	151+
60	500	61+
30	1000	31+
15	2000	16+
10	3000	11+