QUEUES AND THE WHITESPACE PLATFORM

v1.2.2

Executive Summary

The queues system offers a completely seamless, fully-automatable interface between Whitespace's servers and clients' in-house systems.

When a user performs an action on the Whitespace platform, a text-based message describing that action is added to the client's queue. When the client system retrieves that message, it has everything it needs to fetch all the updated database documents and precisely understand the user's action. This allows the client system to process the action in any way necessary, effectively integrating the Whitespace platform into their native IT environment.

Whitespace Queues in Practice

Queues are an efficient method of sending very precise, sophisticated information in a simple, concise form – one that can be easily tailored to a client's exact needs. The process is straight-forward.

When any user action changes or updates data on our servers, this triggers creation of a new activity document, updating our database without obscuring any prior changes. This activity document is free of sensitive information, but provides all the references required for a secured process to retrieve all the information. This document goes onto the client's queue, and also creates emails, generates notifications, and populates the contract history timeline.

The change might be simple – receiving a notification that a broker has not taken up a quote, for example – or it might be as detailed as creating an entirely new contract with multiple sections. Each new activity document becomes a new message.

Once the message has been generated, it goes onto the end of the queue and remains there, until it reaches the head of the queue and is retrieved. Client queues are completely individual, but a client is not restricted to one single queue – they can be set up for specific teams or groups of teams as required.

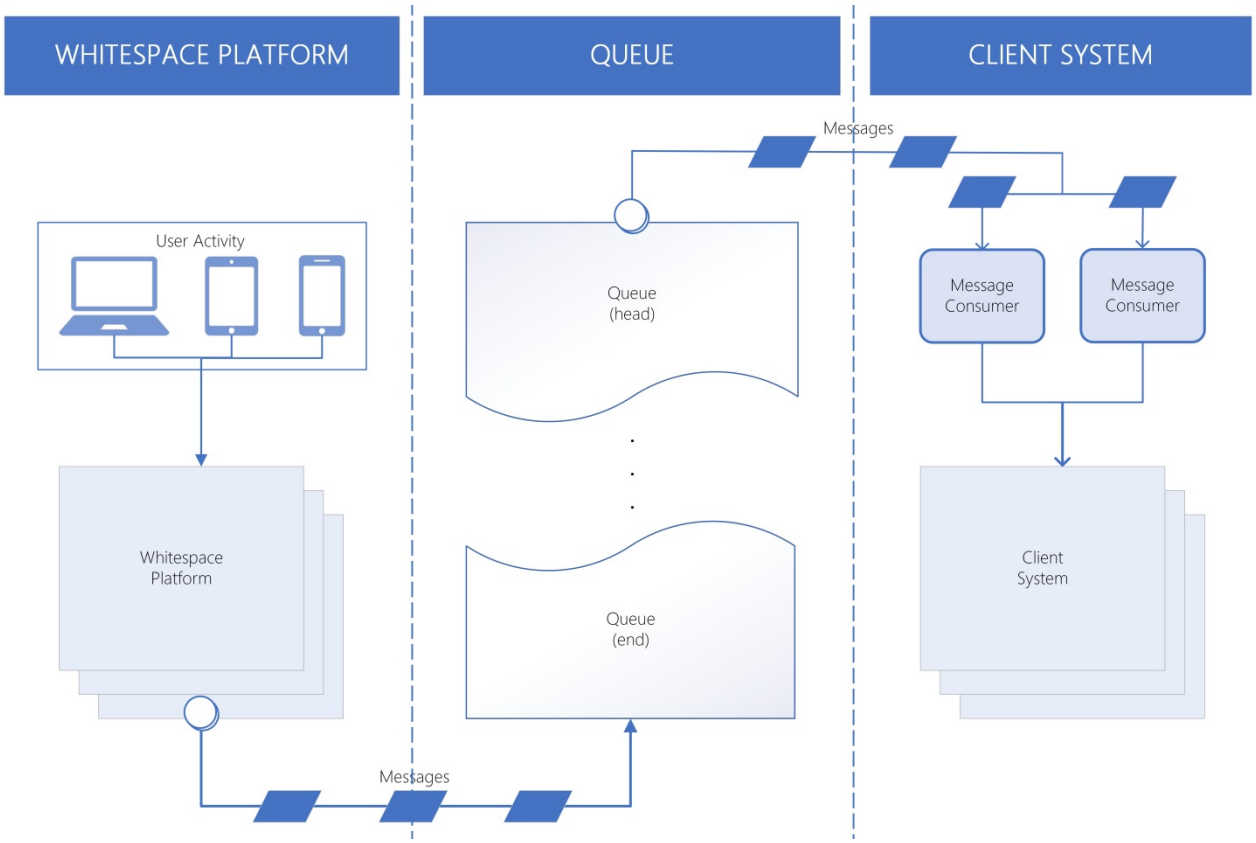

Figure 1: Whitespace Queues system diagram

Messages are time-stamped, and they are served in strict chronological order. When a client system asks the queue for a message, the oldest message available is transmitted. This message is then locked, indicating to the queue that it is being processed.

Each message includes the time-stamp, the exact name of the database document it reflects, an activity code that describes the exact type of operation that created it, and a unique message id. The client system uses the activity code to decide what actions to take – such as informing the responsible underwriter that their written line has been signed, or calling the Whitespace API to download a PDF copy of the new contract and store it in a contract library. Between the activity code, the document reference, the other information encoded in the message, and the API, it is possible to perfectly duplicate every change to the clients' data made on our servers.

Once the message has been processed, the client system lets the queue know that the message processed correctly. The queue then deletes the message to make sure it is not double-processed. If something goes wrong, and the completion isn't sent, the message is unlocked and placed back at the head of the queue.

Common Activity Codes

Some of our more commonly-generated activity codes are detailed below.

As you can see, they are straight-forward and simple to configure. Every message containing an activity code also includes a specific reference to the risk that it is associated with or, where applicable, to the exact instance of the contract.

Broker-facing:

- `Quoted` – An underwriter has sent a quote.
- `Quoted With Quote Details` – An underwriter has sent a quote that includes quote details.
- `Line Written` – An underwriter has added and confirmed a written line.
- `Risk Signed` – The broker has signed the risks on a contract.
- `Showed Endorsement` – An underwriter has examined an endorsement.

Underwriter-facing:

- `Quote Requested` – A broker has requested a quote.
- `Quote Not Taken Up` – A broker has declined a quote.
- `Requested A Line` – A broker has placed a firm order for a quoted line.
- `Signed Lines` – A broker has signed the underwriter's line.
- `Showed Endorsement` – The underwriter has examined an endorsement.

Sample Messages

The activity shown in figure 2 below is a log of Whitespace message activity codes received by a single queue, and indicates a broker using the Whitespace platform to create and place a simple risk.

As the queue receives the messages from the system, certain elements of core data are excerpted to this log and stamped with a precise timecode. The messages themselves are added to the end of the queue. Note that the first of the referenced messages to be served will come from the top of this image, rather than the bottom.

The natural language activity codes in each message show a clear step-by-step path through the process.

Figure 2: A log of message identifiers generated by the Whitespace platform in the process of creating and placing a simple risk

The messages generated by the Whitespace platform are in the JSON text format and precisely describe the activity taking place without exposing any sensitive information. The message payload always includes an activity code and the unique ID of the associated database document. This ID is a long string of values that always begins with IC and

a 36-character code identifying the root risk. Since figure 2 reflects activity on a single risk, the root risk code is the same in each message. The risk code is then followed by various other values appropriate to the action, separated with double colons. The client system can pass this full unique ID and other material contained in the message to the Whitespace API to retrieve complete details of the activity that generated the message, including the protected sensitive data.

By processing the messages according to the activity code in the message, a client system is able to exactly mirror the user's activity on the Whitespace platform.

Messages and Queues in Detail

Queues are a feature of Microsoft Azure's cloud-based Service Bus middleware system. They hold plain-text JSON messages which can be both placed and retrieved using a wide variety of common development platforms – .Net, Java, Python, and many more. These messages are generated by the providing system, the 'originator', and are pushed asynchronously onto the client's unique queue, where they stay until retrieved by the recipient system, the 'consumer'.

Messages use agreed activity codes and reference tokens which allow the consumer system to obtain full details of activity on the originator system. The queue becomes a seamless integration layer between the two systems, one which is completely platform-agnostic.

Queues are extremely flexible and completely robust. Messages are served on a strict first-in first-out basis, delivering them uniquely when a consumer system requests one. For pull-based delivery, the queue can handle up to 100 simultaneous connections. There are also APIs that provide push functionality on an unlimited basis. So messages will be served to as many consumer systems as the client cares to set up. Scaling becomes extremely straight-forward.

There is no need to process messages as they are generated, however. They will wait on the queue indefinitely until a consumer system is available – there is no expiry timer. Each queue can hold more than quarter of a million Whitespace platform messages, so having consumers process messages during user downtime is perfectly feasible. Because of this asynchrony, the client consumers only have to balance their consumer systems around average message load, not peak load.

Settlement systems built into the queue ensure that messages are processed correctly. When a consumer process requests a message, the message at the head of the queue is sent to the consumer and then locked from the queue. Once the consumer has processed the message, it acknowledges this fact to the queue, which permanently deletes the locked message. If the consumer crashes, fails to process, or times out, the message is unlocked and returned to the head of the queue. This is known as at-least once processing.

The alternative to at-least once processing is at-most once processing, where the message is delivered and then instantly deleted with no check for whether the consumer processed the message correctly. This can be implemented if required, but it does come with the risk of lost data if the consumer fails during processing. Each message contains a unique identifier, so having the consumer system generate a log of successfully processed messages and then screen incoming messages against recent log entries is the strongest way to achieve guaranteed exactly-once processing.

Settlement is not the only error-trapping available. The queue has functionality to detect and reject duplicate messages from the originator, helping to ensure no message will be enqueued more than once. Peek functionality allows a consumer system to examine messages without claiming them, so that different consumers can be set up to deal with different activity codes or with different teams' activity. The queue is also able to flag undeliverable, non-processable or poison-pill messages, and directs them to a dead-letter drop which can be examined later as required.

The versatility inherent to queues makes for a very flexible interface between systems. Queues are a loose integration tier, so they are completely independent of any upgrades or system modifications on either side. They span multiple networks, data delivery options, protocols, and trust domains. Consumption patterns are entirely down to the client's preferences. Processes of arbitrary complexity can be assigned to a simple activity code, and matching APIs then provide the client system with perfect access to the exact data.

By floating on an Azure Service Bus queues layer, the Whitespace platform becomes a sophisticated but seamlessly robust front end to any existing client system.