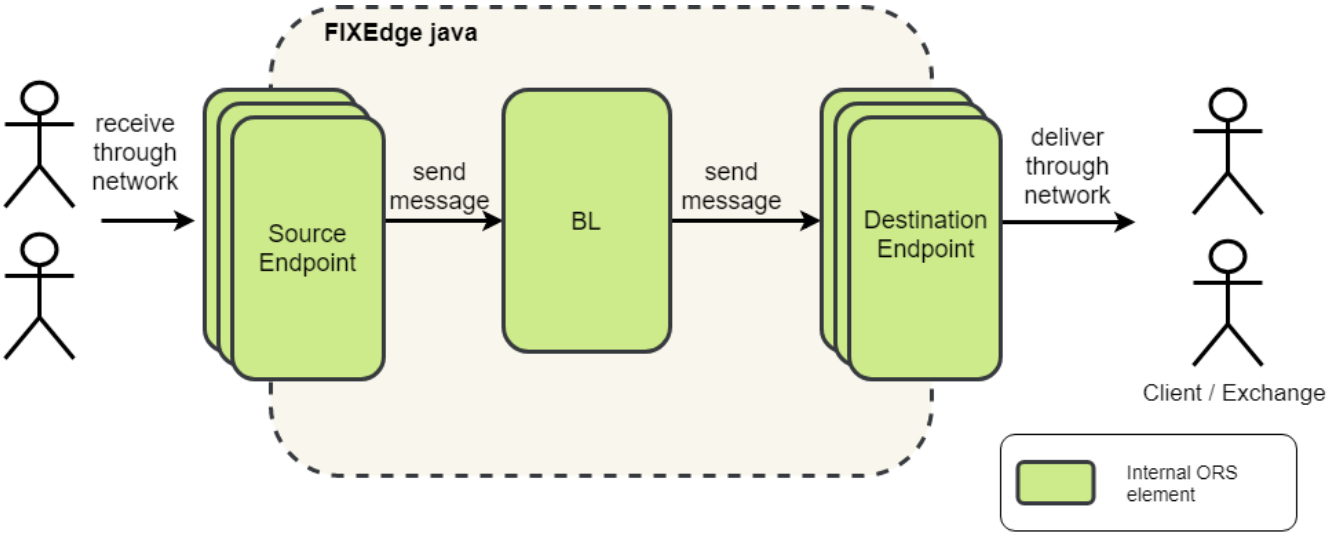
Store and Forward

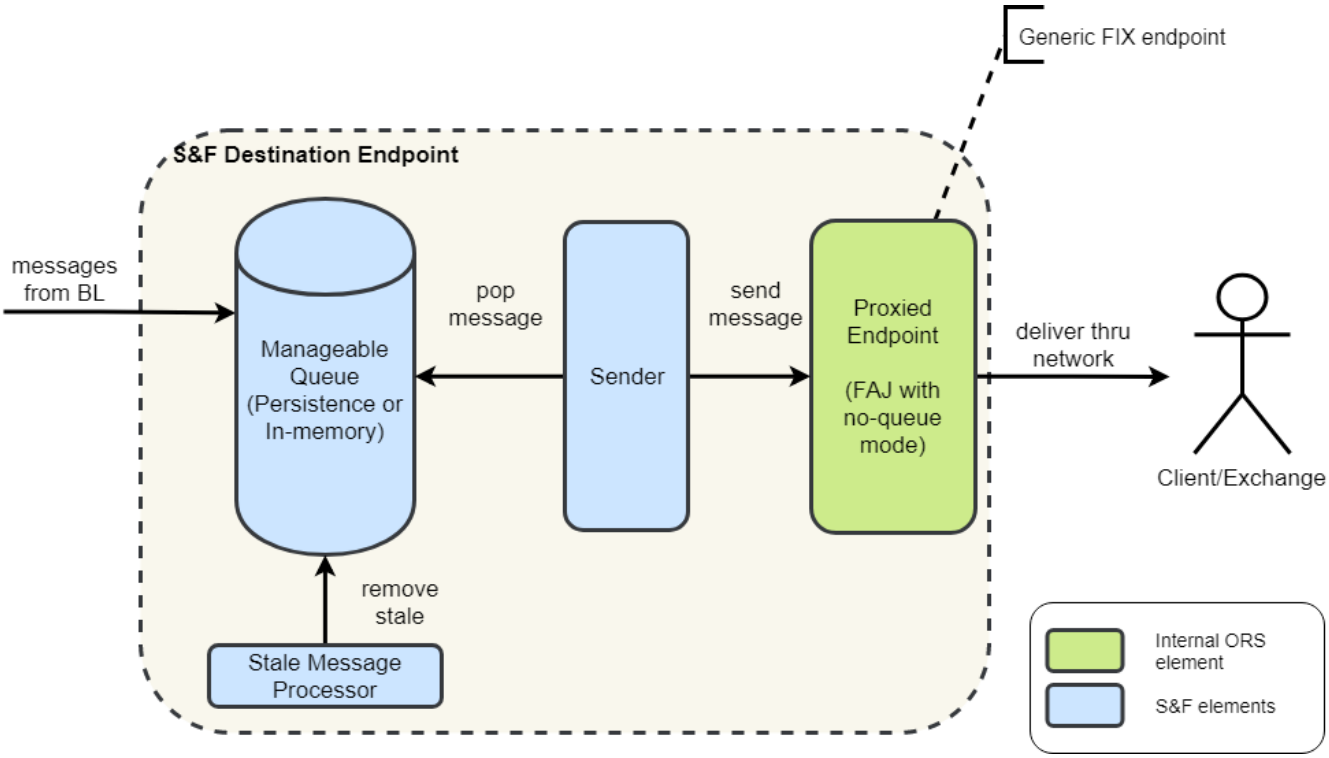
Overview

FIXEdge/J provides functionality that messages are stored and forwarded in the event when a connection with one of the counterparties is dropped and messages are not delivered. These messages will be stored by the queue and forwarded when the receiving counterparty connection is re-established. For example, if one of the counterparties is logged off and messages are being sent, FIXEdge/J can place the undelivered messages into the Store & Forward Queue and route them once the counterparty is logged on.

Store and Forward functionality:

- can be enabled/disabled for a particular session in FIXEdge/J
- can be used for specific message types (business only)
- can be used for specific messages by specifying a particular filter with tags values, for example, 59=0
- can define stale messages and remove them from the queue
- has persisting options - in the memory and on the disk





Configuration

Enable Store and Forward functionality

Enable Store and Forward queue for an endpoint

To enable the Store and Forward queue for a particular endpoint, use the property *storeAndForwardEndpoint.enabled=true/false.*

For example, Client A may want to store all business messages that it sends to the Exchange. In this case, the Store and Forward queue should be enabled on the destination, i.e. the Exchange where it sends to.

s_fix_Exchange.properties

```
sessionType=acceptor
senderCompID=FIXEdgeJCompID
targetCompID=Exchange
storeAndForwardEndpoint.enabled=true
```

NOTE: To enable/disable the Store and Forward queue for all endpoints on FIXEdge/J, use the *storeAndForwardEndpoint.enabled=true/false property* in **s_fixDefault.properties**.

Enable Store and Forward functionality for a session

To enable the Store and Forward queue for a particular session, use the property *storeAndForward.enabled=true/false.*

For example, Client A may want to store all business messages that it sends to the Exchange. At the same time, Client B may not want to store the messages that it sends to the same Exchange. In this case, the Store and Forward functionality should be disabled for Client B.

s_fix_Client_B.properties

```
sessionType=acceptor
senderCompID=FIXEdgeJCompID
targetCompID=ClientB
storeAndForward.enabled=false
```

Compatibility matrix

	storeAndForward.enabled=true	***storeAndForward.enabled=false***
storeAndForwardEndpoint.enabled=true	Client A: storeAndForward.enabled=true Client B: storeAndForward.enabled=true Exchange: storeAndForwardEndpoint.enabled=true Messages from Client A and Client B will be stored in the Exchange Store and Forward queue.	Client A: storeAndForward.enabled=true Client B: storeAndForward.enabled=false Exchange: storeAndForwardEndpoint.enabled=true Messages from Client A will be stored in the Exchange Store and Forward queue. Messages from Client B will not be stored in the Exchange Store and Forward queue.
	Client A: storeAndForwardEndpoint.enabled=true Client B: storeAndForwardEndpoint.enabled=true Exchange: storeAndForward.enabled=true Messages from the Exchange will be stored in the Client Store and Forward queue.	Client A: storeAndForwardEndpoint.enabled=true Client B: storeAndForwardEndpoint.enabled=true Exchange: storeAndForward.enabled=false Messages from the Exchange will not be stored in the Client Store and Forward queue.
storeAndForwardEndpoint.enabled=false	Client A: storeAndForward.enabled=true Client B: storeAndForward.enabled=true Exchange: storeAndForwardEndpoint.enabled=false Messages from Client A and Client B will not be stored in the Exchange Store and Forward queue.	Client A: storeAndForward.enabled=true Client B: storeAndForward.enabled=false Exchange: storeAndForwardEndpoint.enabled=false Messages from Client A and Client B will not be stored in the Exchange Store and Forward queue.
	Client A: storeAndForwardEndpoint.enabled=true Client B: storeAndForwardEndpoint.enabled=false Exchange: storeAndForward.enabled=true Messages from the Exchange will be stored in the Client A Store and Forward queue. Messages from the Exchange will not be stored in the Client B Store and Forward queue.	Client A: storeAndForwardEndpoint.enabled=true Client B: storeAndForwardEndpoint.enabled=false Exchange: storeAndForward.enabled=false Messages from the Exchange will not be stored in the Client Store and Forward queue.

Configure Store and Forward for particular message types

To enable the Store and Forward queue for particular message types, use the property *storeAndForward.msgTypesRequiredForDelivery=F,G.*

For example, Client A may want to store only Order Cancel Request (MsgType = F) messages. In this case, a message type to be stored in the Store and Forward queue should be specified for Client's session.

s_fix_Client_A.properties

```
sessionType=acceptor
senderCompID=FIXEdgeJCompID
targetCompID=ClientA
groups=Group_A
storeAndForward.enabled=true
storeAndForward.msgTypesRequiredForDelivery=F
```

Store and Forward functionality for a particular tag value

Filtering messages by particular tag values can be configured using business rules. There is a special 'requiredForDelivery' function which enables message saving in the Store and Forward queue.

If the "requiredForDelivery" parameter is set to *true* or not defined, then the message will be saved and delivered in any case.

```
...
def requiredForDelivery = false
if(msg.getTagValueAsDouble(44) > 1000 && msg.getTagValueAsString(55) == "AAPL") {
    requiredForDelivery = true
}
send(routingContext, messageContext, destination, requiredForDelivery)
...
```

Configure handling 'stale' messages

FIXEdge/J provides an ability to remove 'stale' orders from the Store and Forward queue. The period after which an order becomes outdated can be defined for each session. In addition, there is an option to exclude some message from the set of 'stale' messages.

For example, Client A may want to define all messages that are older than 30 sec at sending time as 'stale'. At the same time, Client A wants Order Cancel Request (MsgType = F) messages to be always processed to the Exchange.

There are two properties:

- *storeAndForward.stalePeriod* property defines a period (in seconds) which is used to check if the message becomes 'stale' and should not be sent. If the property is set, all queued messages will be checked when a destination becomes available. In addition, if the difference between TransactTime (Tag = 60) and sending time is more that the defined period, the messages will not be sent into the destination.
- *storeAndForward.staleProcessingExcludedTypes* property specifies for which types of messages the stale checking and removing is not be applied.

s_fix_Client_A.properties

```
sessionType=acceptor
senderCompID=FIXEdgeJCompID
targetCompID=ClientA
groups=Group_A
storeAndForward.enabled=true
storeAndForward.stalePeriod=30
storeAndForward.staleProcessingExcludedTypes=F
```

Configure Store and Forward persistence mode

By default, the Store and Forward queue is persisted on the disk in the 'logs' directory.

In addition, there is possibility to configure messages in the memory using the *storeAndForwardEndpoint.queueType* parameter.

Possible values:

- *storeAndForwardEndpoint.queueType* = IN_MEMORY - In this mode, all messages provided are saved in memory. If the system is down (gracefully or non-gracefully), all the messages will be lost.
- *storeAndForwardEndpoint.queueType* = PERSISTENT - In this mode, all messages provided are saved on disk and they will be restored and sent in case of shutdown (gracefully or non-gracefully).
- *storeAndForwardEndpoint.queueType* = REPLICATED - In this mode, all messages will be stored/replicated with the Persistent API functionality.

Configure Store and Forward processing mode

By default, FIXEdge/J processes messages in the asynchronous mode.

In the case when synchronous processing is needed, there is a parameter that allows processing a message in this mode - *storeAndForwardEndpoint.maxTimeForSyncSendingInUsec*.

Please note, that this parameter will be applied only if *storeAndForwardEndpoint.enabled* is *true*.

s_fix_Client_B.properties

```
sessionType=acceptor
senderCompID=FIXEdgeJCompID
targetCompID=ClientB
groups=Group_A
storeAndForwardEndpoint.enabled=true
storeAndForwardEndpoint.maxTimeForSyncSendingInUsec = 60000000
```

This parameter is also possible to be used with slow consumer sessions. Reducing the value of the parameter allows you to start sending in the synchronous mode. If a message wasn't processed in the defined period (in microseconds), FIXEdge/J will switch to the asynchronous mode.

s_fix_Client_B.properties

```
sessionType=acceptor
senderCompID=FIXEdgeJCompID
targetCompID=ClientB
groups=Group_A
storeAndForwardEndpoint.enabled=true
storeAndForwardEndpoint.maxTimeForSyncSendingInUsec = 100
```

Managing messages in the queue

Description

It is possible to manage the store and forward the queue to iterate through messages, remove or add some of them.

Methods

Method name	Description	Example
clear()	Removes all messages	
Iterator<MessageEvent> iterator()	Iterates through all the elements in the queue	`for (Iterator<MessageEvent> iterator = queue .iterator(); iterator.hasNext(); ) { MessageEvent nextEvent = iterator.next(); if (<some condition>) { iterator.remove(); } }`
isEmpty()	Returns *true* if the queue is empty	
MessageEvent peek()	Gets the first element without removing it	
remove(MessageEvent messageEvent)	Removes the specified message event	`MessageEvent first = queue.peek(); queue.remove(first);`

Example of a rule for removing queued orders by cancel request

```
        rule("Remove queued order by order cancel request")
                .condition({
                    ctx -> stringValue(ctx.getMessage(), 35) == "F" //filter by OrderCancelRequest msg type
                })
                .action({
                    ctx ->
                        def cancelRequest = ctx.getMessage()
                        def queue = getManageableQueueByDestinationId(routingContext, "Exchange")
                        def removedFromQueue = false
                        for (Iterator iterator = queue.iterator(); iterator.hasNext(); ) {
                            def queuedMessage = iterator.next().getMessage()
                            if (stringValue(queuedMessage, 37) == stringValue(cancelRequest, 41)) {
                                iterator.remove()
                                removedFromQueue = true
                                break
                            }
                        }
                        if (!removedFromQueue) {
                            //send cancellation to exchange
                        } else {
                            logger.info("Order with id '{}' has been removed from the queue by cancel request
with id '{}'", stringValue(cancelRequest, 41), stringValue(cancelRequest, 37))
                        }
                        ctx.exit()
                })
        .build()
```

Rule for routing order messages

```
        rule("Route order messages to exchange")
                .condition({
                    ctx -> stringValue(ctx.getMessage(), 35) == "D" //filter by Order msg type
                })
                .action({
                    ctx ->
                        send(routingContext, ctx, "Exchange") // send a message to the destination
                        ctx.exit()
                })
                .build()
```