IFS Aviation Maintenance

PubSub User Guide

ICAM 26.2



Document Revision: 26.2-1

Publication Date: November 02, 2026

Contents

1 About PubSub 4

2 Subscribers 5

Registering subscribers 6

Find subscribers through REST 7

3 Subscriptions 8

Defining server properties 9

HTTP subscriptions 10

JMS subscriptions 11

4 PubSub property files and subscriptions 14

Variables 15

Lookups 16

Property Encryption 16

CyberArk 18

5 Publishers 19

6 Data model 21

7 The message queue 23

Clustered environments 23

8 Alerts 25

9 Guaranteed delivery and handling failures 27

Handling failures 28

JMS failures 29

HTTP failures 29

How to retry failed transactions 30

Available dispatchers 30

10 Dispatchers....31
Available dispatchers....32

11 PubSub configuration....33

12 Enabling and disabling PubSub....37

13 Troubleshooting PubSub....38

1

About PubSub

PubSub is an integration framework that allows for multiple systems to communicate without knowing about one another.

This communication is achieved by following a "publish - subscribe" model. In this model, one system, called the publisher, may publish a message to a specific channel while another system, the subscriber, may subscribe to that channel. The two systems do not know about one another; they only know about the PubSub framework, which sits in the middle, operating as a message broker.

2

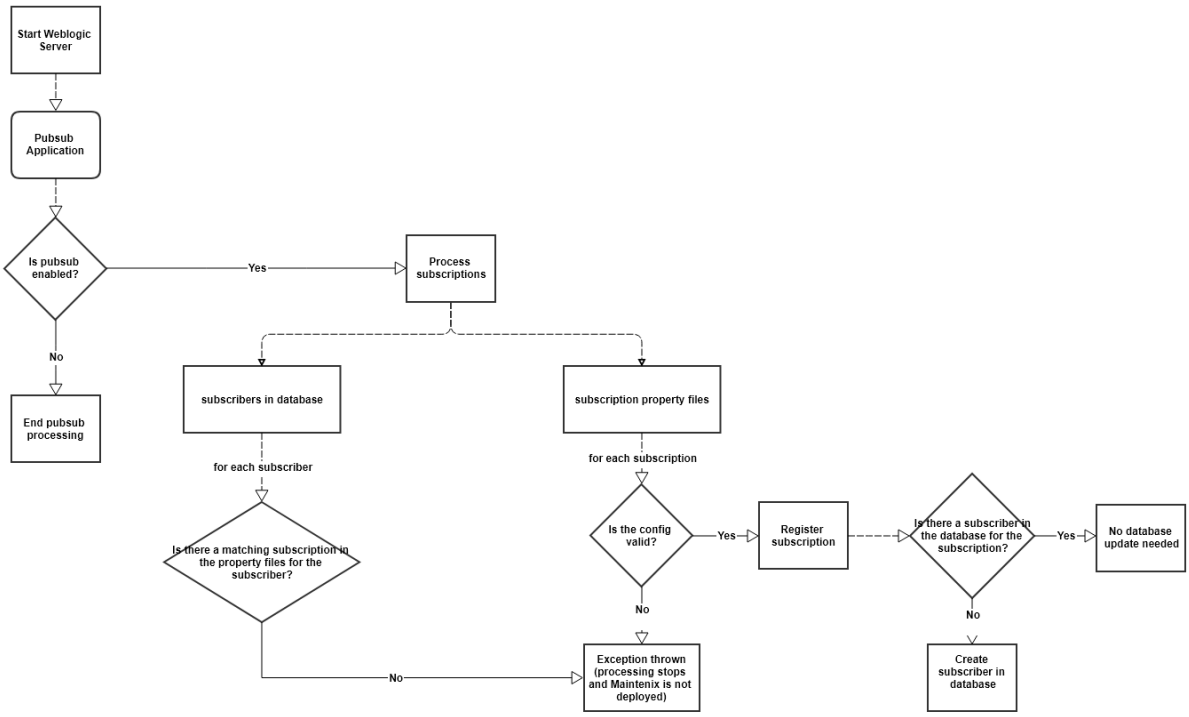
Subscribers

The subscriber is the PubSub component that receives messages on a specified subscription.

To be a subscriber in the PubSub framework, a system must provide the following information:

- The channel it wants to subscribe to.
- An endpoint for the PubSub framework to send messages to.
- Any necessary connection configuration information.

An application registering a subscriber must also have an ID for itself. A subscriber is uniquely identified by its channel and application ID.



Subscriber flow diagram

Registering subscribers

The PubSub framework exposes a few property files. Systems add entries to the property files to register themselves as subscribers and provide connection information. These entries are called subscriptions.

Example

On a server (server1), there is an application (app1) that wants to subscribe to a channel with the name TEST. The complete URL to the application is `http://server1/app1/endpoint`, where `http://server1` is the server URL.

To register the subscriber, the following property files must be updated:

- the `pubsub-subscriptions.properties` file:

```
subscription.TEST.app1.serverId=server1
subscription.TEST.app1.path=/app1/endpoint
```

- the `servers.properties` file:

```
server1.url=http://server1
```

When the server starts up, the subscriber is registered.

Registration exceptions

Certain exceptions during subscriber registration will cause the PubSub application to fail and hence the Maintenix deployment to fail. An error message is shown to indicate which subscriptions have problems.

These exceptions include:

- A subscriber is in the database but is not in the subscription property files.
- A subscription fails validation because of a missing mandatory property.
- A subscription fails validation because of a server property:
  - The server definition cannot be found (invalid server ID).
  - The server definition is missing a mandatory property.
  - The server definition has an invalid or unsupported URL protocol.

Registration warnings

Other scenarios during subscriber registration can cause a warning. The following warning does not prevent the subscriber from being registered or the PubSub application from starting:

- The subscriber has unrecognized properties

Find subscribers through REST

You can retrieve a list of existing subscribers or the details of a specific subscriber using the GET command.

Steps

1. To get the full list of subscribers, enter:

   ```
   GET http://<server>/pubsub/subscribe
   ```

   Response: 200 OK with the list of subscribers.
2. To get the details of a specific subscriber, enter:

   ```
   GET http://<server>/pubsub/subscribe/<id>
   ```

   Response: 200 OK with the subscriber (or 404 Not Found if the subscriber could not be found).

3

Subscriptions

Subscribers register for channels via subscriptions which follow specific format and are added to the `pubsub-subscriptions.properties` file. PubSub accepts customer subscriptions for HTTP and JMS endpoints.

Subscription format

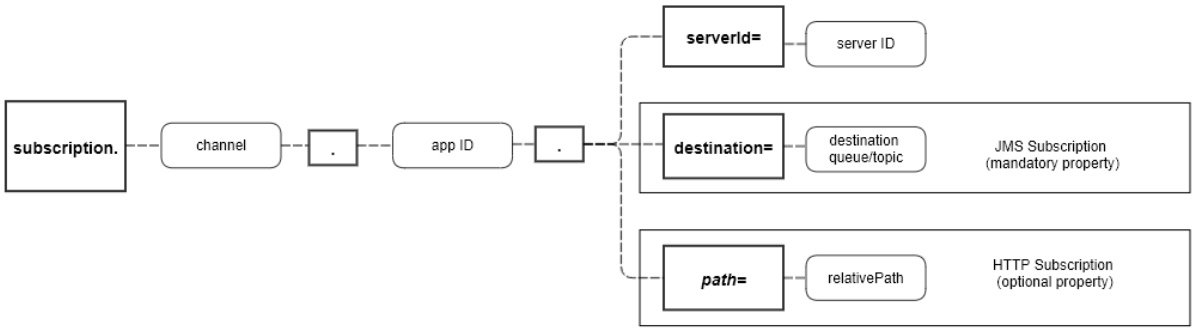
All subscriptions must define a server ID which references a server defined in one of the server property files:

```
subscription.<channel>.<appId>.serverId=<serverId>
```

Subscriptions may also have an optional or mandatory configuration:

```
subscription.<channel>.<appId>.<property>=<value>
```

- The prefix `subscription` defines the start of a subscription property key.
- The field `<channel>` is replaced with the name of the channel to subscribe to.
- The field `<appId>` is replaced with the application identifier (together the channel and the app id make up the unique key for a subscription).
- The field `<property>` is replaced with a configuration property; (The configuration properties of the subscription are defined in the sections that follow.)



Subscription format

Defining server properties

All PubSub subscriptions must have a reference to a server through the server ID. This server can be defined in either `servers.properties` or `mx-servers.properties` files as follows.

- All servers must define a URL:

```
<serverId>.url=<url>
```

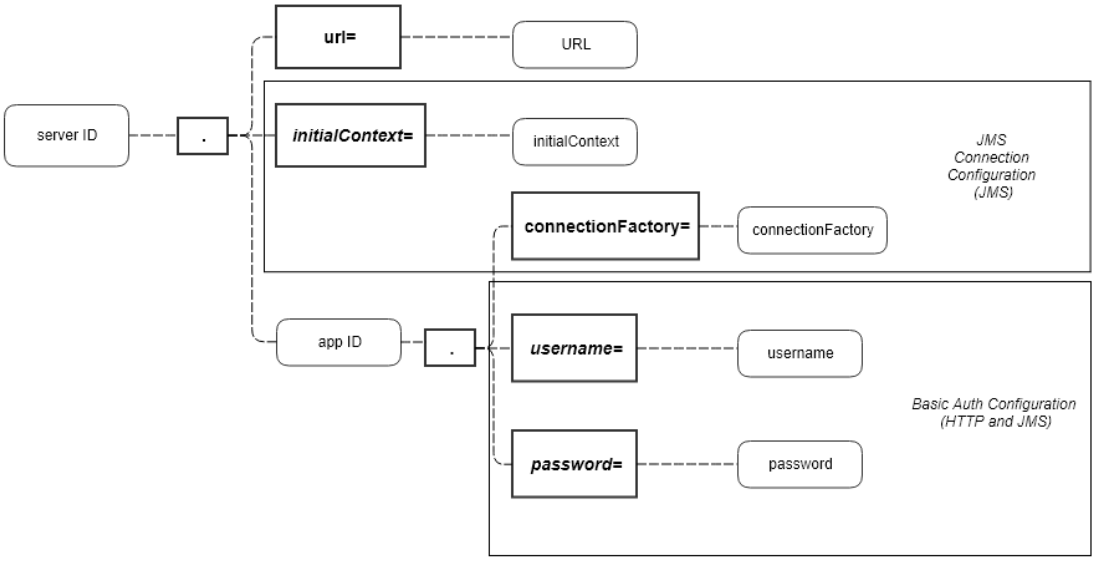
- The server may have its own properties:

```
<serverId>.<property>=<value>
```

- There may be many applications on a server with their own properties:

```
<serverId>.<appId>.<property>=<value>
```

- The prefix `<serverId>` is replaced with the name of the server ID.
- The field `<appId>` is replaced with the application identifier.



Server properties

HTTP subscriptions

Production use of htttp subscriptions consist of valid protocols, configurations, and structure.

Valid protocols for an HTTP subscription:

- http
- https

Allowable configuration:

- path (optional - specifies a relative path to append to the server ID's URL)

HTTP subscriptions refer to a server ID for the server defined in `mx-servers.properties` or `servers.properties` files.

Recommended structure for production use:

- In the `servers.properties` file, define the server information:
  - URL - The URL is the server URL (mandatory)
  - Credentials - The credentials are for a specific application of the server (optional)
- In the `pubsub-subscriptions.properties` file, point the subscription to the server:
  - The subscription gets its connection information from the server properties specified by the serverId (mandatory)

- The subscription can also have a specific path to append to the server URL (optional)
- If credentials are required, ensure the application ID of the subscription matches the application ID used when specifying server credentials

Example subscription 1

in the `servers.properties` file:

```
server1.url=http://server1:8080
server1.app1.username=myUser
server1.app1.password=${encrypted:myPassword}
```

in the `pubsub-subscriptions.properties` file:

```
subscription.TEST.app1.serverId=server1
subscription.TEST.app1.path=/relative/path
```

Example subscription 2

There are some pre-defined servers and authentication URLs in the `mx-servers.properties` file. More information about these is provided in the integration guides that use these properties. PubSub has internal knowledge of which pre-defined servers use which authentication scheme.

JMS subscriptions

This topic covers valid protocols, configuration, and structure for production use of JMS subscriptions. It also lists a few examples of JMS subscriptions.

Valid protocols for a JMS subscription:

- t3
- ssl
- tcp

Allowable configuration:

- destination (mandatory - specifies the queue or topic to append to the JMS server URL)

JMS subscriptions refer to a server ID for the server defined in the `mx-servers.properties` or the `servers.properties` files. Maintenix Weblogic has default server properties already defined in the `mx-servers.properties` file to facilitate adding subscriptions to Weblogic queues and topics.

Recommended structure for production use:

- In the `servers.properties` file, define the following information:

- URL - the URL is the server URL (mandatory).
- Connection factory - the connection factory is the Java class to use as a connection factory and is specific to an application on the server (mandatory).
- Initial context - the initial context holds a list of properties defining the initial context for JNDI which is used to look up the connection factory (optional). Note that the list is comma-separated.
- Credentials - The credentials are for a specific application of the server (optional).

- In the `pubsub-subscriptions.properties` file, point the subscription to the server:
  - The subscription will get its connection information from the server properties specified by the serverId (mandatory).
  - The subscription also must have a destination queue or topic to append to the server URL (mandatory).
  - If credentials are required, ensure the application ID of the subscription matches the application ID used when specifying server credentials.

Multiple URLs format:

If you have multiple URLs, separate them by commas:

```
server1.url=<url1, url2>
```

where `<url1, url2>` is a list of URLs.

Example 1 - Minimal queue subscription with Maintenix Weblogic

The Maintenix Weblogic server properties are defined in the `mx-servers.properties` file with a default application:

```
maintenix.weblogic.url=t3://localhost:7001
maintenix.weblogic.initialContext=java.naming.factory.initial=weblogic.
 jndi.WL
InitialContextFactory
maintenix.weblogic.default.connectionFactory=com.mxi.am.
 jms.ConnectionFactory
```

Add a subscription to the `mx-pubsub-subscriptions.properties` file:

```
subscription.TEST.default.serverId=maintenix.weblogic
subscription.TEST.default.destination=myQueue
```

Example 2 - Queue subscription with basic Auth and Weblogic context

In the `servers.properties` file:

```
server1.url=t3://localhost:7001/
server1.initialContext=java.naming.factory.initial=weblogic.jndi.
 WLInitialCont
extFactory
server1.app1.connectionFactory=com.mxi.am.jms.ConnectionFactory
server1.app1.username=myUser
server1.app1.password=${encrypted:myPassword}
```

In the `pubsub-subscriptions.properties` file:

```
subscription.TEST.app1.serverId=server1
subscription.TEST.app1.destination=myQueue
```

Example 3 - Topic subscription with multiple URLs and Tibco SSL context

In the `servers.properties` file:

```
server1.url=ssl://localhost:7001,ssl://localhost:7002/
server1.initialContext=java.naming.factory.initial=com.tibco.
 tibjms.naming.Tib
jmsInitialContextFactory,com.tibco.tibjms.naming.
 security_protocol=ssl,com.tib
co.tibjms.naming.ssl_enable_verify_host=false
server1.app1.connectionFactory=com.mxi.am.jms.ConnectionFactory
server1.app1.username=myUser
server1.app1.password=${encrypted:myPassword}
```

In the `pubsub-subscriptions.properties` file:

```
subscription.TEST.app1.serverId=server1
subscription.TEST.app1.destination=myTopic
```

4

PubSub property files and subscriptions

There are four property files that PubSub scans for subscription information: `pubsub-subscriptions.properties, servers.properties, mx-pubsub-subscriptions.properties, mx-servers.properties`. They contain either Maintenix-specific information or customer-specific information and variables for subscriptions.

The PubSub property files fall in two categories: customer files (`pubsub-subscriptions.properties` and `servers.properties`) and Maintenix files (`mx-pubsub-subscriptions.properties` and `mx-servers.properties`).

Customer files

The content of the customer-owned files is preserved between upgrades.

pubsub-subscriptions.properties

Contains customer-owned subscriptions and variables for subscriptions.

This file is the primary mechanism for adding subscribers.

servers.properties

Contains variables for customer server configuration

Maintenix files

The content of the Maintenix files is owned by IFS and is subject to change between upgrades.

mx-pubsub-subscriptions.properties

Contains Maintenix-specific subscriptions

mx-servers.properties

Contains variables for Maintenix server configuration

Editing the properties files

- Any additions by customers should be done in the customer-owned files, over which the customer has full control.
- Customers may modify the values of the `mx-servers.properties` file, but they should not add or remove keys.
- No modifications should be made to the `mx-pubsub-subscriptions.properties` file.

Important It is recommended to keep the server-configuration variables in the `servers.properties` file to facilitate the move from test to production environment.

The Maintenix `mx-servers.properties` and `servers.properties` files store dynamic information that will change between environments (e.g., host and port information).

The content of the `mx-pubsub-subscriptions.properties` and `pubsub-subscription.properties` files should store static information. If these standards are followed, then the only files that will need to be updated in an environment upgrade will be the `servers.properties` and `mx-servers.properties` files.

Important Having duplicated variables across the property files or within one property file is not recommended; it will cause errors on startup or when dispatching a message to a subscriber.

Variables

You can create variables that can later be used in defining subscriptions. Variables can be defined in either of the customer-owned property files: `pubsub-subscriptions.properties` or `servers.properties`.

Variable interpolation

Variables are very useful for containing content that may change between environments. Variables can also be reused between many subscriptions (for example, a queue name).

For example:

- In the `servers.properties` file, you can define the following variable:

```
server1.myqueue=myQueue
```

- Then in `pubsub-subscriptions.properties` file, you can refer to that variable as:

```
subscription.TEST.app1.destination=${server1.myqueue}
```

Best practices for variable naming

Variable names must be unique, or you will get exceptions when starting up of the server. You are recommended to use a distinctive name for each variable.

Lookups

PubSub supports two types of lookups that allow specialized processing of a value: a custom "encryption" lookup to encrypt properties, and the CyberArk lookup.

The lookup format is similar to that of variable interpolation but has a keyword followed by a colon (:).

For example:

```
app1.key=${lookup_keyword:lookup_value}
```

Important Nested variable interpolations inside lookups are not supported. In the preceding example, replacing `lookup_value` with a variable like `${lookup_variable}` would be invalid.

Property Encryption

There are two methods of encrypting values in the property file: using an encrypted lookup or using the .password suffix on a property key.

- Example of using an encrypted lookup:
  - How to use it: `app1.key=${encrypted:valueToEncryptGoesHere}`
  - After encryption:

`app1.key=${encrypted:[AES]ulWDBAJ7lARKEnLCkzDGIA==}`

- Example of using the `.password` suffix on a property key.
- The property key must end with ".password" and the value must be plain-text.
  - How to use it: `app1.key.password=valueToEncryptGoesHere`
  - After encryption:

`app1.key.password=${encrypted:[AES]ulWDBAJ7lARKEnLCkzDGIA==}`

When the server starts up for the first time, the property files will be scanned and any values that need encryption (as indicated by one of the two methods above) will be encrypted. The value in the file will be rewritten to the encrypted value that only the server can decrypt.

Important You cannot use variable interpolation with either encryption method.

Note When setting a password in the text editor, the password will be encrypted when the server is restarted. To have the password automatically encrypted as you enter it, use the Password Encryption Utility. See the *Maintenix Installation and Configuration Guide* for more details.

Examples of invalid formats

Example 1:

```
someVariable=valueToEncryptGoesHere
```

In this example, nothing will be encrypted

Example 2

```
app1.key=${encrypted:${someVariable}}
```

In this example, `"${someVariable}"` will be treated as the value and encrypted

Example 3

```
someVariable=valueToEncryptGoesHere
app1.key.password=${someVariable}
```

This example is not technically invalid, but it illustrates the risk of a password remaining unencrypted. Here `${someVariable}` will be replaced by its value `"valueToEncryptGoesHere"` which is never encrypted.

CyberArk

By implementing a CyberArk lookup, PubSub supports CyberArk for those customers that have CyberArk support installed. CyberArk is a password vault system.

The following string is an example of the CyberArk lookup format:

```
${cyberark:<safe>:<folder>:<object>:<reason>:<app_id>}
```

An example below displays a subscription using CyberArk in the `servers.properties` file:

```
server1.url=http://localhost:7001
server1.app1.username=myUser
server1.app1.password=${cyberark:safe1:folder1:object1:reason1:app1}
```

An example in the `pubsub-subscriptions.properties` file:

```
subscription.TEST.app1.serverId=server1
subscription.TEST.app1.path=/endpoint
```

This example illustrates the use of CyberArk lookup. Note that there is a ':' separated ordered list of information that must be communicated with CyberArk to allow it to provide PubSub with the password when dispatching a message.

The list order is as follows:

```
safe, folder name, object, reason code, app id
```

5

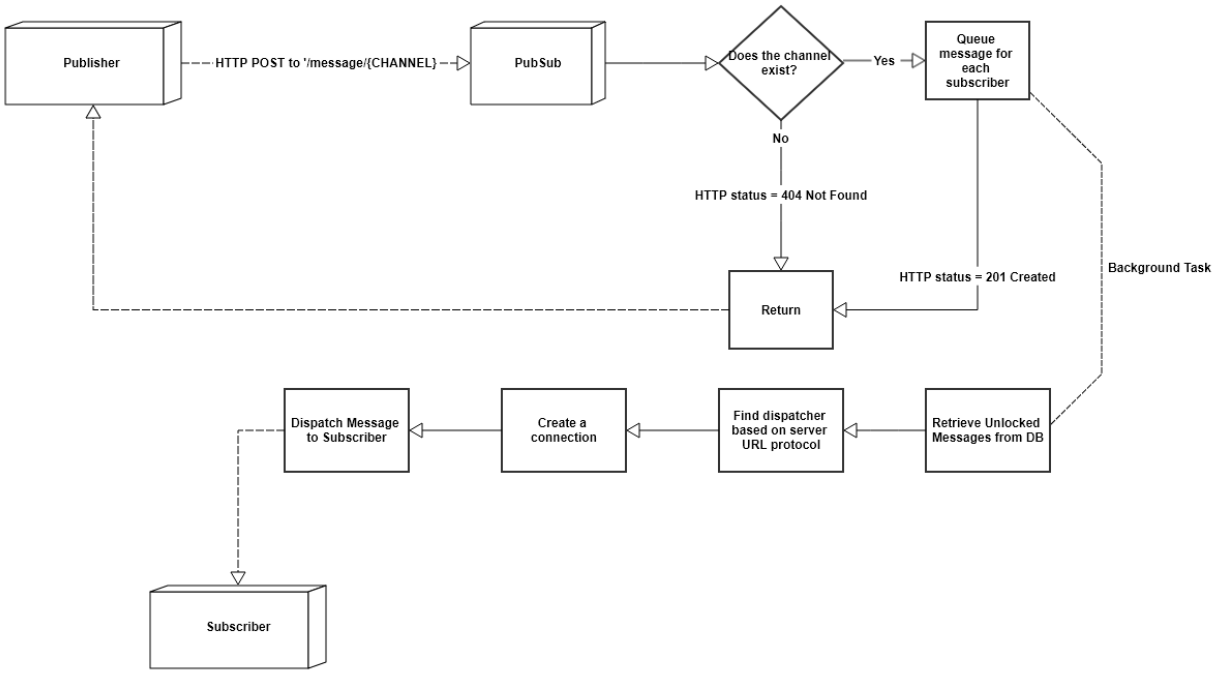
Publishers

Unlike subscribers, publishers are not registered using the property file. Publishing systems simply send a message to the PubSub framework and nothing more. The PubSub framework makes no attempts to transform the message sent by the publisher; it simply dispatches the message to all subscribers of the channel on which the message was sent.

The following endpoint exists to publish a message to PubSub:

- HTTP endpoint `<server>/pubsub/message/<channel>`

Note If the message was sent to a channel that doesn't exist, then the message is dropped and a "404 Not Found" response is returned.



Publisher flow diagram

Properties/Headers

When the message is dispatched (via HTTP or JMS), any property/header keys that are prefixed with an "x-" or "X-" are passed along to the subscriber. The HTTP dispatcher will also pass along the Content-Type header.

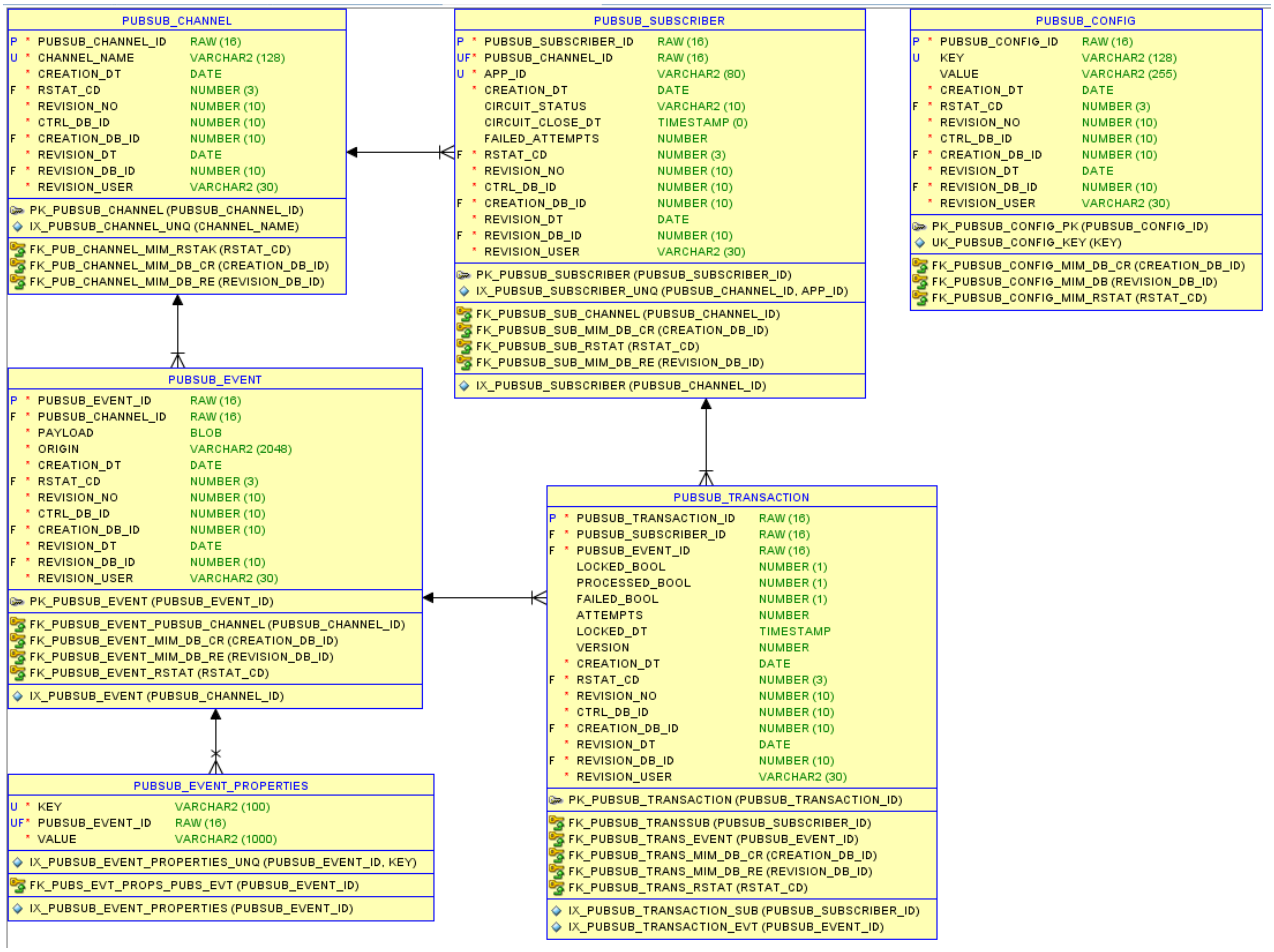
Message body

Whatever the publisher sends as the message body is what the subscriber will receive. This content may be plain text, JSON, XML, YAML, etc. The PubSub framework DOES NOT perform any message transformation.

6

Data model

The following figure shows the database model for the PubSub framework.



The purpose of the PUBSUB_EVENT table is to store information about a message, such as its content and which channel it belongs to; however, it does not directly join to the subscribers.

This connection occurs through the PUBSUB_TRANSACTION table, which stores information about the publishing of a message to a specific subscriber. This table tracks whether the message has been successfully processed, and how many attempts have been made to deliver the message.

7

The message queue

The message queue is what drives the PubSub framework under the hood. It is responsible for loading messages from the database and processing them.

This process occurs every five seconds and is executed on multiple threads to improve execution time.

Clustered environments

Due to the threaded nature of the message queue and the potential for the PubSub framework to be deployed in a clustered environment, it is important to understand how messages are processed.

The PUBSUB_TRANSACTION table has a column named LOCKED_BOOL. This Boolean value is used to track whether or not the PubSub transaction is currently being processed. When the message queue runs, it gathers all transactions that have a value of false for LOCKED_BOOL and PROCESSED_BOOL. These unlocked transactions are then locked to prevent any other PubSub servers from picking them up, which in turn prevents messages from being delivered multiple times.

If a message is successfully delivered, then its PROCESSED_BOOL value is set to TRUE and is eligible to be deleted. It is important to note that when a transaction is locked, its LOCKED_DT value is updated. If a transaction has not been successfully processed and has a LOCKED_DT that is more than 30 minutes in the past, it will count as having a LOCKED_BOOL value of FALSE. This process is in place to handle the scenario in which one of the servers in the cluster has gone down while processing a message.

Known Limitation: In a clustered environment, the configuration will be applied only to the node handling the REST request. All other nodes will not have their configuration updated until their application servers have been restarted.

Note If a transaction is successfully processed, it will be automatically deleted by the message queue from the database. If all transactions for a message are deleted, then that event will also be deleted from the database.

8

Alerts

PubSub raises alerts to Maintenix to indicate to users when there is a communication error or another scenario that needs to indicate to users when there is a communication error or another scenario that needs attention.

There are two instances when PubSub raises alerts:

- A message fails to publish and is not retried.
- A subscribers circuit is closed due to too many retry attempts.

By default, the alerts are assigned to the INTEGRA role. PubSub alerts are managed in Maintenix on the Alert Setup page. For more information on alerts, see the Maintenix Administration Guide.

PubSub alert values

Alert Name	Role	Description	Message
PubSub subscriber circuit closed	INTEGRA	No messages will be sent to the subscriber while its circuit is closed.	`The circuit was shutdown for the subscriber with app ID [appId] and channel [channel] due to connection issues on the server with ID [serverId]. No messages will be sent to this subscriber until the circuit reopens. See the server logs for more information.`
PubSub transaction failed	INTEGRA	Failed to publish message to PubSub channel.	`Failed to publish message to channel [channel] with subscriber with app ID [appId] for the transaction with ID [transactionID]. Error: [error_message]`

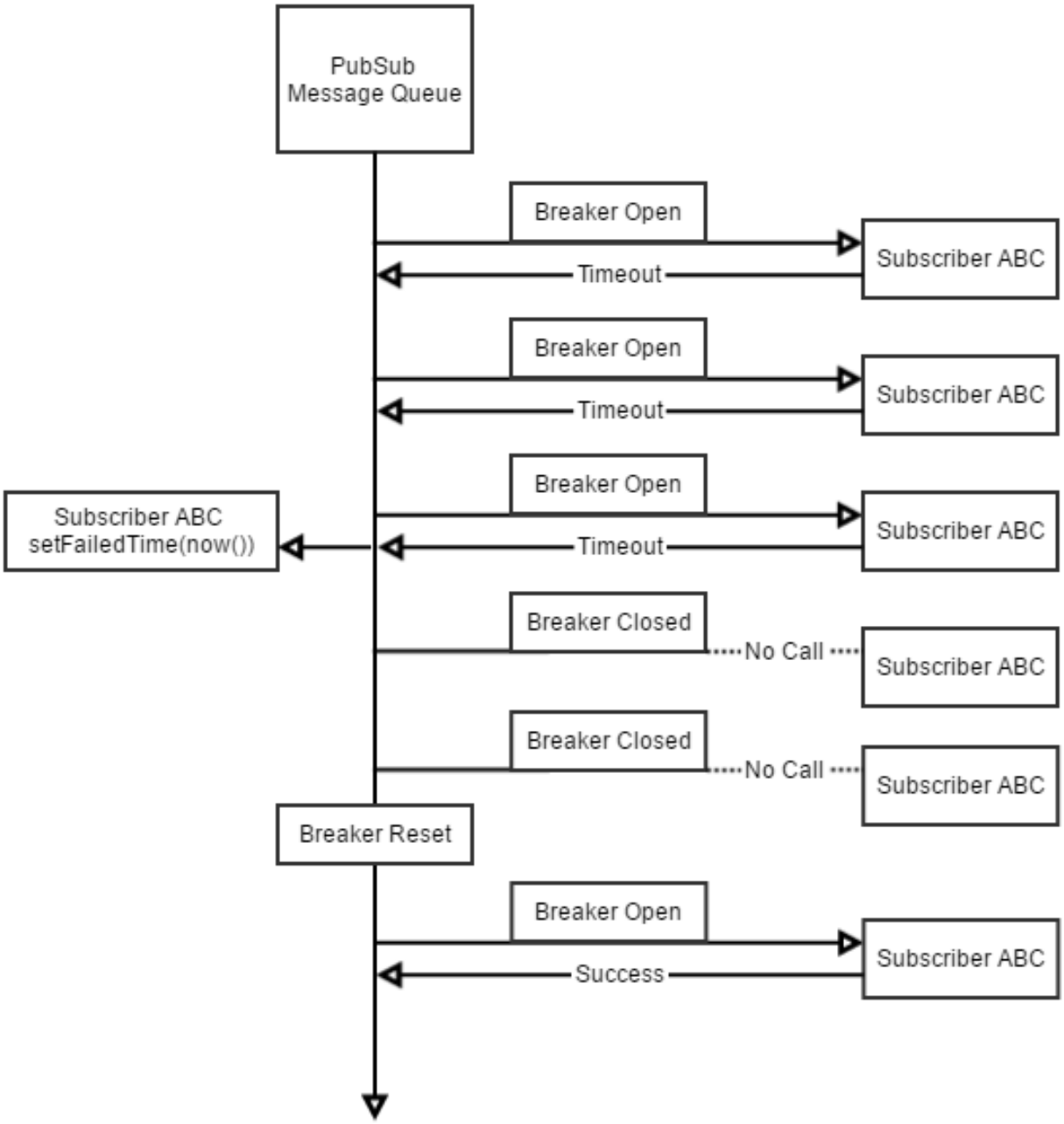
9

Guaranteed delivery and handling failures

The PubSub system guarantees message delivery by implementing the circuit breaker pattern.

When the system receives a message, it will create a transaction for each subscriber and attempt to dispatch that transaction to the subscriber.

The following diagram outlines the basic idea of the circuit breaker pattern in the context of the event queue.



Circuit breaker diagram

Handling failures

PubSub has measures for handling message failure.

If the system fails to dispatch the message, it will do one of two things:

1. If the failure is deemed to be recoverable, the fail counter will be incremented on the subscriber, and the transaction will be tried again.

2. If the failure is deemed to be non-recoverable, the transaction will be marked as failed which prevents any further attempts on the transaction.

If a subscriber's fail count passes the configured threshold, then the subscriber's circuit will be shut down for the configured amount of time. While the circuit is shut down, no attempts will be made to dispatch messages to the subscriber. This process helps to prevent the PubSub system from becoming overloaded by failing messages. After the configured shutdown duration has passed, the circuit will reopen and more delivery attempts to the subscriber will be made.

Note While PubSub guarantees message delivery, it is possible in certain circumstances that a message is delivered multiple times to the same subscriber. Please be aware of this and take appropriate precautions even though it is an infrequent occurrence.

JMS failures

If a JMS `ConnectException` is thrown, it is considered recoverable and the transaction will be tried again. Any other exception that occurs during dispatching a message via JMS is considered non-recoverable.

Non-recoverable exceptions include:

- The given connection factory cannot be found.
- The specified queue or topic cannot be found.

HTTP failures

If an HTTP connection cannot be established due to a connection exception, this event is treated as a recoverable exception. If any other type of exception occurs before dispatching, it is considered non-recoverable.

If a connection is successfully established, then the classification of recoverable or non-recoverable depends on the HTTP status code returned in the HTTP response.

Recoverable HTTP Status Codes

The following HTTP response status codes are considered recoverable by PubSub:

- 408 (Request Timeout)
- 429 (Too Many Requests)
- 503 (Service Unavailable)
- 504 (Gateway Timeout)
- 507 (Insufficient Storage)

- 509 (Bandwidth Limit Exceeded)
- 598 (Network read timeout error)
- 599 (Network connect timeout error)

Non-recoverable HTTP Status Codes

HTTP response status codes above 299 (300 and higher) are considered non-recoverable, so they do not appear on the recoverable list. For example, 500 (Internal Server Error).

How to retry failed transactions

To retry a failed transaction, you can manually update the pubsub_transaction table in the Maintenix database. Set both the failed_bool and locked_bool to 0 to reset the transaction.

For example, use the following SQL to reset all transactions:

```
UPDATE
 pubsub_transaction
 SET
 failed_bool = 0,
 locked_bool = 0
 WHERE
 failed_bool = 1;
```

Available dispatchers

The HttpDispatcher and JmsDispatcher are available for sending messages to different channels

The HttpDispatcher sends a message to a subscriber through an HTTP/HTTPS POST call. When provided with credentials, it will send an Authorization header using the context value. The HttpDispatcher forwards the Content-Type header from the original request as well as any headers prepended with "X-"

- Protocols: HTTP, HTTPS
- Supported Configuration: Basic Auth

The JmsDispatcher sends a message to a JMS queue or topic. It requires a JMS context which contains the connection factory class name, as well as the JMS queue or topic name. If it's provided with credentials, it will use those credentials when connecting to the JMS queue. Message properties prepended with X- will also be passed along.

- Protocols: TCP, T3, SSL
- Supported Configuration: Basic Auth

10

Dispatchers

Before a message can be delivered to a subscriber, it must first be dispatched. This is achieved through the use of dispatchers. Each dispatcher is responsible for delivering a message to a subscriber based on the subscriber's URL.

Each dispatcher may support one or more protocols. These protocols are used to determine which dispatcher should dispatch the message to the subscriber.

For example, the HTTP dispatcher supports the HTTP and HTTPS protocols. Given a subscription with the URL endpoint set as `http://system1/endpoint`, the protocol will be identified as HTTP and the PubSub framework will dispatch the message using the HTTP dispatcher. If you choose to use the JMS dispatcher, you will need some additional configuration. Here is an example of a JMS subscription:

- in the `servers.properties` file:

```
jmsServer1.url=t3://localhost:7001/
jmsServer1.jmsServer1App.username=system
jmsServer1.jmsServer1App.password=${encrypted:password1}
jmsServer1.jmsServer1App.connectionFactory=com.mxi.am.jms.
 ConnectionFactory
jmsServer1.initialContext=java.naming.factory.initial=weblogic.
 jndi.WLInitial
ContextFactory
```

- in the `pubsub-subscriptions.properties` file:

```
subscription.TEST2.jmsServer1App.serverId=jmsServer1
subscription.TEST2.jmsServer1App.destination=someQueue
```

The dispatcher requires a connection factory class with the fully qualified Java class name. Also, there are JNDI initial context parameters specified as a comma-separated list of keys and values (key=value). The JNDI initial context is used to look up the connection factory.

Available dispatchers

The HttpDispatcher and JmsDispatcher are available for sending messages to different channels

The HttpDispatcher sends a message to a subscriber through an HTTP/HTTPS POST call. When provided with credentials, it will send an Authorization header using the context value. The HttpDispatcher forwards the Content-Type header from the original request as well as any headers prepended with "X-"

- Protocols: HTTP, HTTPS
- Supported Configuration: Basic Auth

The JmsDispatcher sends a message to a JMS queue or topic. It requires a JMS context which contains the connection factory class name, as well as the JMS queue or topic name. If it's provided with credentials, it will use those credentials when connecting to the JMS queue. Message properties prepended with X- will also be passed along.

- Protocols: TCP, T3, SSL
- Supported Configuration: Basic Auth

11

PubSub configuration

The PubSub system can be configured through a REST endpoint located at `sys/config` and also through the WebLogic system management console.

The REST resource exposes a number of variables which affect the behavior of the PubSub system. Some variables determine their value based on the current state of the system, while others are simply values (all of which have a default). When a client sets a config value, that value is stored in the database. The next time the PubSub system starts up, it will load the values from the database instead of using the defaults.

The following table lists the configuration values that are currently exposed.

Key	Default	Description
QUEUE_STATUS	STARTED	This value displays and controls the status of the PubSub system's internal message queue. It may be set to a value of STARTED or STOPPED, however it may also return STARTING and STOPPING as values to better reflect the state of the system. Note that other configuration values may restart the queue, so that their new values can take effect.

Key	Default	Description
QUEUE_CYCLE_TIME	5000	This value (time provided in milliseconds) represents how often the message queue runs. Changing this value will trigger a message queue restart.
DELETE_PROCESS_STATUS	STARTED	This value displays and controls the status of the PubSub system's internal delete queue which cleans up processed transactions or events. It may be set to a value of STARTED or STOPPED, however it may also return STARTING and STOPPING as values to better reflect the state of the system. Note that other configuration values may restart the queue so that their new values can take effect.
DELETE_CYCLE_TIME	60000	This value (time provided in milliseconds) represents how often the delete queue runs. Changing this value will trigger a delete queue restart.
LOCK_EXPIRE_TIME	1800000	This value (time provided in milliseconds) represents the threshold for counting a locked transaction as unlocked.
SHUTDOWN_MODE	GRACEFUL	This value controls how thread pools are shut down. It may have a value of GRACEFUL or FORCE. GRACEFUL will wait for all threads in the pool to finish execution, regardless of how long it takes while FORCE will shut down the thread pool immediately.

Key	Default	Description
CIRCUIT_RESET_TIME	30000	This value (time provided in milliseconds) controls for how long a subscriber's circuit is in shutdown. When a subscriber's circuit is shutdown, no messages will be delivered to the subscriber. Once the specified time has elapsed, then the circuit will be reopened and messages will begin to flow.
DISPATCH_ATTEMPTS	5	This value controls how many connection errors must occur before a subscriber's circuit is shutdown.
BATCH_SIZE	100	This value controls how many unlocked transactions (or transactions with expired lock) are fetched to be dispatched to their subscribers at one time.
SEQUENCE_MAX	199	This value is the maximum value of the PUBSUB_BATCH_SEQ cyclical database sequence. The sequence ensures that multiple PubSub nodes do not dispatch the same messages. If the sequence maximum is updated in the database then this configuration parameter must also be updated to match it, and vice versa, in order for the sequence to be reset when no messages are found at later sequence values.

Certain values can be adjusted using the WebLogic system management console (`http://servername:7001/console`). The dispatcher queue has a thread pool with minimum and

maximum values depending on the load. These values can be adjusted in the console in the following location:

`Domain Structure > Deployments > assetmanagement (expand with the +) > pubsub > Configuration > Workload > wm/PubSub-DispatcherWorkManager`

When you navigate to this page, you will see two links at the bottom of the page for `PubSub-DispatchMaxThreadConstraint and PubSub-DispatchMinThreadConstraint`. You can navigate to each place and adjust the values. A server restart is required for the changes to take effect. The default is for a minimum of 5 threads and a maximum of 25 threads.

12

Enabling and disabling PubSub

PubSub is disabled by default. In a disabled state, PubSub will not publish messages to the subscribers.

You can still access and use the PubSub APIs, but a transaction (sending a message to a subscriber) will never be completed as the background processes for dispatching and deleting messages are turned off. Any configuration information in the database will be disregarded. To enable PubSub, find the `pubsub.properties` property file and set the value of `com.mxi.pubsub.config.enabled` to TRUE. Save the file, and restart the application server.

13
Troubleshooting PubSub

In most cases, the logs contain a message when an error occurs with PubSub. If there are no errors, it is likely because PubSub is turned off or the endpoint is improperly configured.

Scenario: Messages are not published to PubSub

Symptoms: There are no new entries in the pubsub_transaction and pubsub_event table right after an action occurs which should publish a message.

Possible Causes:

- PubSub is not turned on (`pubsub.properties -> com.mxi.pubsub.config.enabled=true`).
- The message is published by a trigger and that trigger is not enabled in utl_trigger.

Scenario: Messages are published to PubSub and are being sent out, but the endpoint is not receiving them

Symptoms: There are new entries in the pubsub_transaction and pubsub_event table right after an action occurs which publishes a message. The transactions are then processed and purged, but the endpoint does not receive a message.

Possible Causes:

- The endpoint is sending back a 200 for rejection/error scenarios. PubSub assumes the message has been accepted if it receives a response in the 200s. (If the message has a wrapper, ensure that the wrapper designated for the endpoint is used.)
- The endpoint account is improperly configured.

Scenario: Messages are published to PubSub, and PubSub is receiving error responses when it sends the message out

Symptoms: PubSub is not purging messages or transactions from tables. Transactions are marked as failed or have been attempted multiple times. There should be errors in the logs.

Possible Causes:

- Malformed subscriptions.
  - Wrong endpoint (check the subscription and server property files to see which server is being used and what the server's URL is).
  - Invalid credentials or authentication (username or password; check the server property files for credentials).
- Invalid schema is followed.
- Invalid message wrapper is used for the endpoint.
- Endpoint provided is not configured or active.

ABOUT IFS

IFS develops and delivers enterprise software for customers around the world who manufacture and distribute goods, maintain assets, and manage service-focused operations. The industry expertise of our people and solutions, together with commitment to our customers, has made us a recognized leader and the most recommended supplier in our sector. Our team of 3,500 employees supports more than 10,000 customers world-wide from a network of local offices and through our growing ecosystem of partners.

#forthechallengers

ifs.com

WHERE WE ARE

AMERICAS

+1 888 437 4968

ASIA PACIFIC

+65 63 33 33 00

EUROPE EAST

+48 22 577 45 00

EUROPE CENTRAL

+49 9131 77 340

UK & IRELAND

+44 1494 428 900

FRANCE, BENELUX AND IBERICA

+33 3 89 50 72 72

MIDDLE EAST AND AFRICA

+971 4390 0888

NORDICS

+46 13 460 4000

COPYRIGHT © 2026 INDUSTRIAL AND FINANCIAL SYSTEMS, IFS AB. IFS AND ALL IFS PRODUCTS AND SERVICES NAMES ARE TRADEMARKS OF IFS. ALL RIGHTS RESERVED. THIS DOCUMENT MAY CONTAIN STATEMENTS OF POSSIBLE FUTURE FUNCTIONALITY FOR IFS'S PRODUCTS AND TECHNOLOGY. SUCH STATEMENTS ARE FOR INFORMATION PURPOSES ONLY AND SHOULD NOT BE INTERPRETED AS ANY COMMITMENT OR REPRESENTATION. THE NAMES OF ACTUAL COMPANIES AND PRODUCTS MENTIONED HEREIN MAY BE THE TRADEMARKS OF THEIR RESPECTIVE OWNERS.