**AMQP** 0-9-1 (Advanced Message Queuing Protocol) is a messaging protocol that enables conforming client applications to communicate with conforming messaging middleware brokers.

AMQP entities:

**Queues**, **exchanges** and **bindings** are collectively referred to as AMQP entities.

**Brokers**

Messaging brokers receive messages from publishers/producers and route them to consumers (applications that process them).

Since it is a network protocol, the publishers, consumers and the broker can all reside on different machines.

**AMQP 0-9-1 Model in Brief**

* Messages are published to exchanges (post offices or mailboxes).
* Exchanges then distribute message copies to queues using rules called bindings.
* Then AMQP brokers either deliver messages
  + To consumers subscribed to queues, or
  + Consumers fetch/pull messages from queues on demand.

When a message is delivered to a consumer the consumer notifies the broker, either automatically or as soon as the application developer chooses to do so. When message acknowledgements are in use, a broker will only completely remove a message from a queue when it receives a notification for that message (or group of messages).

AMQP 0-9-1 is a programmable protocol in the sense that AMQP entities and routing schemes are defined by applications themselves, not a broker administrator.

Accordingly, provision is made for protocol operations that declare queues and exchanges, define bindings between them, subscribe to queues and so on.

This gives application developers a lot of freedom but also requires them to be aware of potential definition conflicts.

In practice, definition conflicts are rare and often indicate a misconfiguration.

Applications declare the AMQP entities that they need, define necessary routing schemes and may choose to delete AMQP entities when they are no longer used.

**Exchanges**

Exchanges are AMQP entities where messages are sent.

* Exchanges take a message and route it into zero or more queues.
  + The routing algorithm used depends on the exchange type and rules called bindings.
* Exchanges can be durable or transient.
  + Durable exchanges survive broker restart whereas transient exchanges do not

AMQP 0-9-1 brokers provide four exchange types:

Default Exchange

The default exchange is a direct exchange with no name (empty string) pre-declared by the broker.

It has one special property that makes it very useful for simple applications:

Every queue that is created is automatically bound to it with a routing key which is the same as the queue name.

The default exchange makes it seem like it is possible to deliver messages directly to queues, even though that is not technically what is happening.

Direct Exchange

A direct exchange delivers messages to queues based on the message routing key.

Here is how it works:

A queue binds to the exchange with a routing key K

When a new message with routing key R arrives at the direct exchange, the exchange routes it to the queue if K = R

Fanout Exchange

A fanout exchange routes messages to all of the queues that are bound to it and the routing key is ignored.

Fanout exchanges are ideal for the broadcast routing of messages.

Topic Exchange

Topic exchanges route messages to one or many queues based on matching between a message routing key and the pattern that was used to bind a queue to an exchange.

The topic exchange type is often used to implement various publish/subscribe pattern variations.

Topic exchanges are commonly used for the multicast routing of messages.

Topic exchanges have a very broad set of use cases.

Whenever a problem involves multiple consumers/applications that selectively choose which type of messages they want to receive, the use of topic exchanges should be considered.

Headers Exchange

A headers exchange is designed for routing on multiple attributes that are more easily expressed as message headers than a routing key.

Headers exchanges ignore the routing key attribute. Instead, the attributes used for routing are taken from the headers attribute.

A message is considered matching if the value of the header equals the value specified upon binding.

It is possible to bind a queue to a headers exchange using more than one header for matching.

In this case, the broker needs one more piece of information from the application developer, namely, should it consider messages with any of the headers matching

, or all of them? This is what the "x-match" binding argument is for. When the "x-match" argument is set to "any", just one matching header value is sufficient.

Alternatively, setting "x-match" to "all" mandates that all the values must match.

Headers exchanges can be looked upon as "direct exchanges on steroids".

Because they route based on header values, they can be used as direct exchanges where the routing key does not have to be a string;

it could be an integer or a hash (dictionary) for example.

**Queues**

Queues store messages that are consumed by applications.

* Before a queue can be used it has to be declared.
* Declaring a queue will cause it to be created if it does not already exist.
* The declaration will have no effect
  + If the queue does already exist and
  + Its attributes are the same as those in the declaration.
* When the existing queue attributes are not the same as those in the declaration a channel-level exception with code 406 (PRECONDITION\_FAILED) will be raised.

Queues properties:

* Name: Applications may pick queue names or ask the broker to generate a name for them.
  + Queue names starting with "amq." are reserved for internal use by the broker.
  + Attempts to declare a queue with a name that violates this rule will result in a channel-level exception with reply code 403 (ACCESS\_REFUSED).
* Durable: Durable queues are persisted to disk and thus survive **broker** restarts.
  + If broker is taken down and then brought back up, durable queue will be re-declared during broker startup.
  + Durability of a queue does not make messages that are routed to that queue durable.
  + only persistent messages will be recovered
  + Queues that are not durable are called transient.
* Exclusive (used by only one connection and the queue will be deleted when that connection closes)
* Auto-delete (queue is deleted when last consumer unsubscribe)
* Arguments (some brokers use it to implement additional features like message TTL)

**Bindings**

Bindings are rules that exchanges use (among other things) to route messages to queues. To instruct an exchange E to route messages to a queue Q, Q has to be bound to E. Bindings may have an optional routing key attribute used by some exchange types. The purpose of the routing key is to select certain messages published to an exchange to be routed to the bound queue. In other words, the routing key acts like a filter.

This layer of indirection enables routing scenarios publishing directly to queues and also eliminates certain amount of duplicated work application developers have to do.

If AMQP message cannot be routed to any queue it is either dropped or returned to the publisher, depending on message attributes the publisher has set.

Consumers : There are two types of Consumers:

* Have messages delivered to them ("push API")
  + With the "push API", applications register a consumer or subscribe to a queue.
* Fetch messages as needed ("pull API")

AMQP broker remove messages from queues in one of below scenarios:

* Automatic acknowledgement model - After broker sends a message to an application
* Explicit acknowledgement model. -After the application sends back an acknowledgement

When will application send an acknowledgement?

* It can be right after receiving a message, or
* After persisting it to a data store before processing, or
* After fully processing the message

If a consumer dies without sending an acknowledgement the AMQP broker will redeliver it to another consumer or, if none are available at the time, the broker will wait until at least one consumer is registered for the same queue before attempting redelivery.

An application can indicate to the broker that message processing has failed by rejecting a message.

When rejecting a message, an application can ask the broker to discard or requeue it.