

---

## AMQP



Simple Elixir wrapper for the Erlang RabbitMQ client.

The API is based on Langohr, a Clojure client for RabbitMQ.

### Upgrading guides

To upgrade from the old version, please read our upgrade guides:

- [0.x to 1.x](#)
- [1.x to 2.x](#)
- [2.x to 3.x](#)

### Usage

Add AMQP as a dependency in your `mix.exs` file.

```
1 def deps do
2   [
3     {:amqp, "~> 3.3"}
4   ]
5 end
```

Elixir will start `amqp` automatically with this if you use Elixir 1.6+.

If that's not the case (use `Application.started_applications/0` to check), try adding `:amqp` to `applications` or `extra_applications` in your `mix.exs`. Or call `Application.ensure_started(:amqp)` at the start.

After you are done, run `mix deps.get` in your shell to fetch and compile AMQP. Start an interactive Elixir shell with `iex -S mix`.

```
1 iex> {:ok, conn} = AMQP.Connection.open()
2 # {:ok, %AMQP.Connection{pid: #PID<0.165.0>}}
3
4 iex> {:ok, chan} = AMQP.Channel.open(conn)
5 # {:ok, %AMQP.Channel{conn: %AMQP.Connection{pid: #PID<0.165.0>}, pid:
6   #PID<0.177.0>}}
7
8 iex> AMQP.Queue.declare(chan, "test_queue")
9 # {:ok, %{consumer_count: 0, message_count: 0, queue: "test_queue"}}
```

---

```
10 iex> AMQP.Exchange.declare(chan, "test_exchange")
11 # :ok
12
13 iex> AMQP.Queue.bind(chan, "test_queue", "test_exchange")
14 # :ok
15
16 iex> AMQP.Basic.publish(chan, "test_exchange", "", "Hello, World!")
17 # :ok
18
19 iex> {:ok, payload, meta} = AMQP.Basic.get(chan, "test_queue")
20 iex> payload
21 # "Hello, World!"
22
23 iex> AMQP.Queue.subscribe(chan, "test_queue", fn payload, _meta -> IO.
    puts("Received: #{payload}") end)
24 # {:ok, "amq.ctag-5L8U-n0HU5doEsNTQpaXWg"}
25
26 iex> AMQP.Basic.publish(chan, "test_exchange", "", "Hello, World!")
27 # :ok
28 # Received: Hello, World!
```

## Setup a consumer GenServer

```
1 defmodule Consumer do
2   use GenServer
3   use AMQP
4
5   def start_link do
6     GenServer.start_link(__MODULE__, [], [])
7   end
8
9   @exchange "gen_server_test_exchange"
10  @queue "gen_server_test_queue"
11  @queue_error "#{@queue}_error"
12
13  def init(_opts) do
14    {:ok, conn} = Connection.open("amqp://guest:guest@localhost")
15    {:ok, chan} = Channel.open(conn)
16    setup_queue(chan)
17
18    # Limit unacknowledged messages to 10
19    :ok = Basic.qos(chan, prefetch_count: 10)
20    # Register the GenServer process as a consumer
21    {:ok, _consumer_tag} = Basic.consume(chan, @queue)
22    {:ok, chan}
23  end
24
25  # Confirmation sent by the broker after registering this process as a
    consumer
```

---

```

26  def handle_info({:basic_consume_ok, %{consumer_tag: consumer_tag}},
27      chan) do
28      {:noreply, chan}
29  end
30  # Sent by the broker when the consumer is unexpectedly cancelled (
31      such as after a queue deletion)
32  def handle_info({:basic_cancel, %{consumer_tag: consumer_tag}}, chan)
33      do
34      {:stop, :normal, chan}
35  end
36  # Confirmation sent by the broker to the consumer process after a
37      Basic.cancel
38  def handle_info({:basic_cancel_ok, %{consumer_tag: consumer_tag}},
39      chan) do
40      {:noreply, chan}
41  end
42  def handle_info({:basic_deliver, payload, %{delivery_tag: tag,
43      redelivered: redelivered}}, chan) do
44      # You might want to run payload consumption in separate Tasks in
45      production
46      consume(chan, tag, redelivered, payload)
47      {:noreply, chan}
48  end
49  defp setup_queue(chan) do
50      {ok, _} = Queue.declare(chan, @queue_error, durable: true)
51      # Messages that cannot be delivered to any consumer in the main
52      queue will be routed to the error queue
53      {ok, _} = Queue.declare(chan, @queue,
54          durable: true,
55          arguments: [
56              {"x-dead-letter-exchange", :longstr, ""},
57              {"x-dead-letter-routing-key", :longstr,
58                  @queue_error}
59          ]
60      )
61      :ok = Exchange.fanout(chan, @exchange, durable: true)
62      :ok = Queue.bind(chan, @queue, @exchange)
63  end
64  defp consume(channel, tag, redelivered, payload) do
65      number = String.to_integer(payload)
66      if number <= 10 do
67          :ok = Basic.ack channel, tag
68          IO.puts "Consumed a #{number}."
69      else
70          :ok = Basic.reject channel, tag, requeue: false

```

---

```

67     IO.puts "#{number} is too big and was rejected."
68   end
69
70   rescue
71     # Requeue unless it's a redelivered message.
72     # This means we will retry consuming a message once in case of
73     # exception
74     # before we give up and have it moved to the error queue
75     #
76     # You might also want to catch :exit signal in production code.
77     # Make sure you call ack, nack or reject otherwise consumer will
78     # stop
79     # receiving messages.
80     exception ->
81       :ok = Basic.reject channel, tag, requeue: not redelivered
82       IO.puts "Error converting #{payload} to integer"
83   end
84 end

```

```

1 iex> Consumer.start_link
2 {:ok, #PID<0.261.0>}
3 iex> {:ok, conn} = AMQP.Connection.open
4 {:ok, %AMQP.Connection{pid: #PID<0.165.0>}}
5 iex> {:ok, chan} = AMQP.Channel.open(conn)
6 {:ok, %AMQP.Channel{conn: %AMQP.Connection{pid: #PID<0.165.0>}, pid: #
  PID<0.177.0>}}
7 iex> AMQP.Basic.publish chan, "gen_server_test_exchange", "", "5"
8 :ok
9 Consumed a 5.
10 iex> AMQP.Basic.publish chan, "gen_server_test_exchange", "", "42"
11 :ok
12 42 is too big and was rejected.
13 iex> AMQP.Basic.publish chan, "gen_server_test_exchange", "", "Hello,
    World!"
14 :ok
15 Error converting Hello, World! to integer
16 Error converting Hello, World! to integer

```

## Configuration

**Connections and channels** You can define a connection and channel in your config and AMQP will automatically...

- Open the connection and channel at the start of the application
- Automatically try to reconnect if they are disconnected

```

1 config :amqp,
2   connections: [

```

---

```
3     myconn: [url: "amqp://guest:guest@myhost:12345"],
4   ],
5   channels: [
6     mychan: [connection: :myconn]
7   ]
```

You can access the connection/channel via `AMQP.Application`.

```
1 iex> {:ok, chan} = AMQP.Application.get_channel(:mychan)
2 iex> :ok = AMQP.Basic.publish(chan, "", "", "Hello")
```

When a channel is down and reconnected, you have to make sure your consumer subscribes to a channel again.

See the documentation for `AMQP.Application.get_connection/1` and `AMQP.Application.get_channel/1` for more details.

## Types of arguments and headers

The parameter `arguments` in `Queue.declare`, `Exchange.declare`, `Basic.consume` and the parameter `headers` in `Basic.publish` are a list of tuples in the form `{name, type, value}`, where `name` is a binary containing the argument/header name, `type` is an atom describing the AMQP field type and `value` a term compatible with the AMQP field type.

The valid AMQP field types are:

```
:longstr | :signedint | :decimal | :timestamp | :table | :byte | :double | :float |
:long | :short | :bool | :binary | :void | :array
```

Valid argument names in `Queue.declare` include:

- “x-expires”
- “x-message-ttl”
- “x-dead-letter-routing-key”
- “x-dead-letter-exchange”
- “x-max-length”
- “x-max-length-bytes”

Valid argument names in `Basic.consume` include:

- “x-priority”
- “x-cancel-on-ha-failover”

Valid argument names in `Exchange.declare` include:

- “alternate-exchange”

---

## Troubleshooting / FAQ

**Consumer stops receiving messages** It usually happens when your code doesn't send acknowledgement(ack, nack or reject) after receiving a message.

If you use GenServer for your consumer, try storing the number of messages the server is currently processing to the GenServer state.

If the number equals `prefetch_count`, those messages were left without acknowledgements and that's why the consumer has stopped receiving more messages.

Also review the following points:

- when an exception was raised how it would be handled
- when `:exit` signal was thrown how it would be handled
- when a message processing took long time what could happen

Also make sure that the consumer monitors the channel pid. When the channel is gone, you have to reopen it and subscribe to a new channel again.

**The version compatibility** Check out this article to find out the compatibility with Elixir, OTP and RabbitMQ.

**Heartbeats** In case the connection is dropped automatically, consider enabling heartbeats.

You can set `heartbeat` option when you open a connection.

For more details, read this article

**Does the library support AMQP 1.0?** Currently the library doesn't support AMQP 1.0 and there is no plan to do so at the moment. Our main aim here (at least for now) is to provide a thin wrapper around `amqp_client` for Elixir programmers.

## Copyright and License

Copyright (c) 2014 Paulo Almeida

This library is MIT licensed. See the LICENSE for details.