Integration of the Managed Trading Service with AMQP

betradar
driven by facts

Version	Author	Date	Comment
1.7.3	Aleksander Michalik	2015-10-23	replyRoutingKey and correlationId features introduced: - Chapter 2.4. Message control added - Other affected passages altered.
1.7.4	Aleksander Michalik	2016-02-17	Acknowledgement specs updated - Routing keys - Acknowledgement-Exchange types and message descriptions
3.0	Andrei Byvshev	2017-03-05	- TLS connection with MTS AMQPS - correlationId naming clarified - delivery mode for message clarified - Cancelation protocol additional details

betradar
driven by facts

Table of Contents

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

1. Purpose of this document

This is a specific guide, aiming at an audience knowledgeable in the programming language of their choice and preferably experienced in the application of AMQP. Therefore AMQP basics are kept short and more emphasis is put on MTS specifications. Nevertheless some general potential problematic points and/or best practices are lined out in chapter 3 still. Additionally a compilation of links to AMQP basics, such as principles and tutorials and further content is found at the end.

2. Integration with MTS

2.1 General MTS integration principle

The general MTS integration is based on the message-answer principle - as in every sort of bi-directional communication.
The core concept is the submission of tickets and their evaluation.
A ticket represents a bet placed by the customer (the punter).
The customer submits a punter's *ticket information* for initial evaluation, which represent the main message. He then receives the MTS answer - which can be the recommendation to that submission - means either *ticket acceptance* or *refusal*. Additionally the customer has the option to respond, informing MTS about his decision regarding the MTS recommendation. It means he has the option to submit the *Info Ticket acceptance.*
Other messages to and replies from MTS beyond the mentioned are also thinkable and their realization partly realized or in progress.
The *ticket cancellation* falls here under.
Their status will be updated with the evolution of MTS itself.

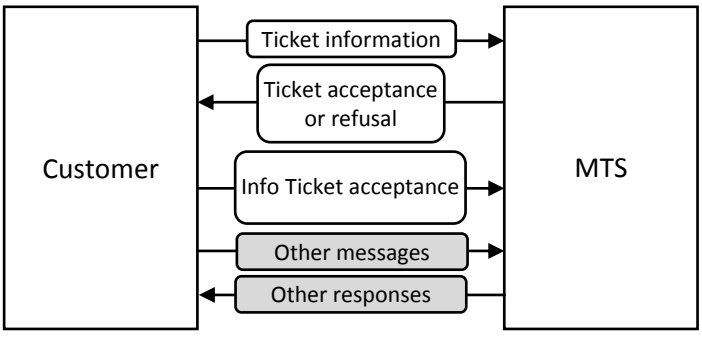


Fig. 3.1.: General principle of the MTS integration

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

2.2. Access information

2.2.1. Connection points

- **Client integration environment**

Address	**integration-mts.betradar.com [91.201.213.134]**
Port	**5672**
Secure connection port	**5671**

- **Production environment**

Address	**tradinggate.betradar.com [91.201.212.86]**
Port	**5672**
Secure connection port	**5671**

2.2.2. Virtual host, user, password

Please contact your Betradar MTS-integration manager for your *username* and *password*.
As a convention, this *username* is used in the virtual host's name as well as in your designated Exchanges and Queues.
The virtual host's name is then simply ***/username*** (a slash followed by the username).

Secure connection

When establishing a TLS connection with MTS AMQPS service you will get a *.betradar.com wildcard x.509 certificate which was signed by a trustworthy CA - Entrust. That means you should connect to the hostname tradinggate.betradar.com on port 5671. Machines usually come with that CA included in the trust store so verification of the chain should succeed without any intervention on client's side. Of course it is still in your domain to do the proper action (e.g., disconnect) when validation fails. We don't employ client certificates for authentication but there is a IP whitelist in place to ensure only authorized clients are able to connect.

You can expect such metadata in the cert:

subject=/C=CH/L=St.Gallen/O=Sportradar AG/CN=*.betradar.com issuer=/C=US/O=Entrust, Inc./OU=See www.entrust.net/legal-terms/OU=(c) 2012 Entrust, Inc.

You can also use certificate pinning, but please be advised the validity period of the wildcard certificate is relatively short (1 year).

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

2.2.3. Exchanges

Exchange name	Purpose	Type	Access/ Options	Description
username-Submit	submit ticket information	fanout	write	sending the ticket
username-Control	submit control messages	topic	write	f.eg. cancel ticket
username-Confirm	receive ticket acceptance or refusal	topic	read	accept/refuse recommendation
username-Reply	receive control messages reply	topic	read	f.eg. accept/refuse cancellation request
username-Ack	submit - info ticket acceptance and - info ticket cancellation acceptance	topic	Write	- customer feedback to MTS' ticket acceptance/refusal recommendation and - customer feedback to MTS' cancellation acceptance/refusal

Exchanges are durable and are provided and maintained by MTS. You only have to know to which one you have to publish a respective message and to which one you will have to bind your Queue before consuming messages (see following chapters).

2.2.4. Queues

Queue name	Type	Purpose	Description
username-(Submit|Confirm|Ack|Control|Reply)*	any Queue	consume any message	see subchapter 3.3.

You are obliged to declare, bind, consume from and to maintain the Queues of your custom needs (see chapter 3.3.).

2.2.4.1. Queue naming convention

Therefor you have the right to instantiate any AMQP-entity within your vhost that follows the following naming-pattern:

"username-(Submit|Confirm|Ack|Control|Reply)" …your username followed by a dash followed by one of the ' "|" (or) – separated strings ' within the brackets followed by any valid*

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

string. According to the RabbitMQ documentation a valid string here can be empty, or a sequence of these characters: letters, digits, hyphen, underscore, period, or colon.

For convenience reasons this is the same permission range as is granted to you regarding the entity-names from which you are allowed to read. However, currently, if it comes to creating named entities within your vhost, you only need to regard Queues. These Queues that you have to create you are required then to either bind to your "Confirm" or "Reply"-Exchange (see chapter 2.2.3. or 2.3.).

2.2.4.2. Queue naming recommendation

Therefore we recommend you to name your Queues respective of the Exchange you want to bind them to:

- username-Confirm-node*X*, …if you intend to bind this Queue to your Confirm-Exchange
- username-Reply-node*X*, …if you intend to bind this Queue to your Reply-Exchange

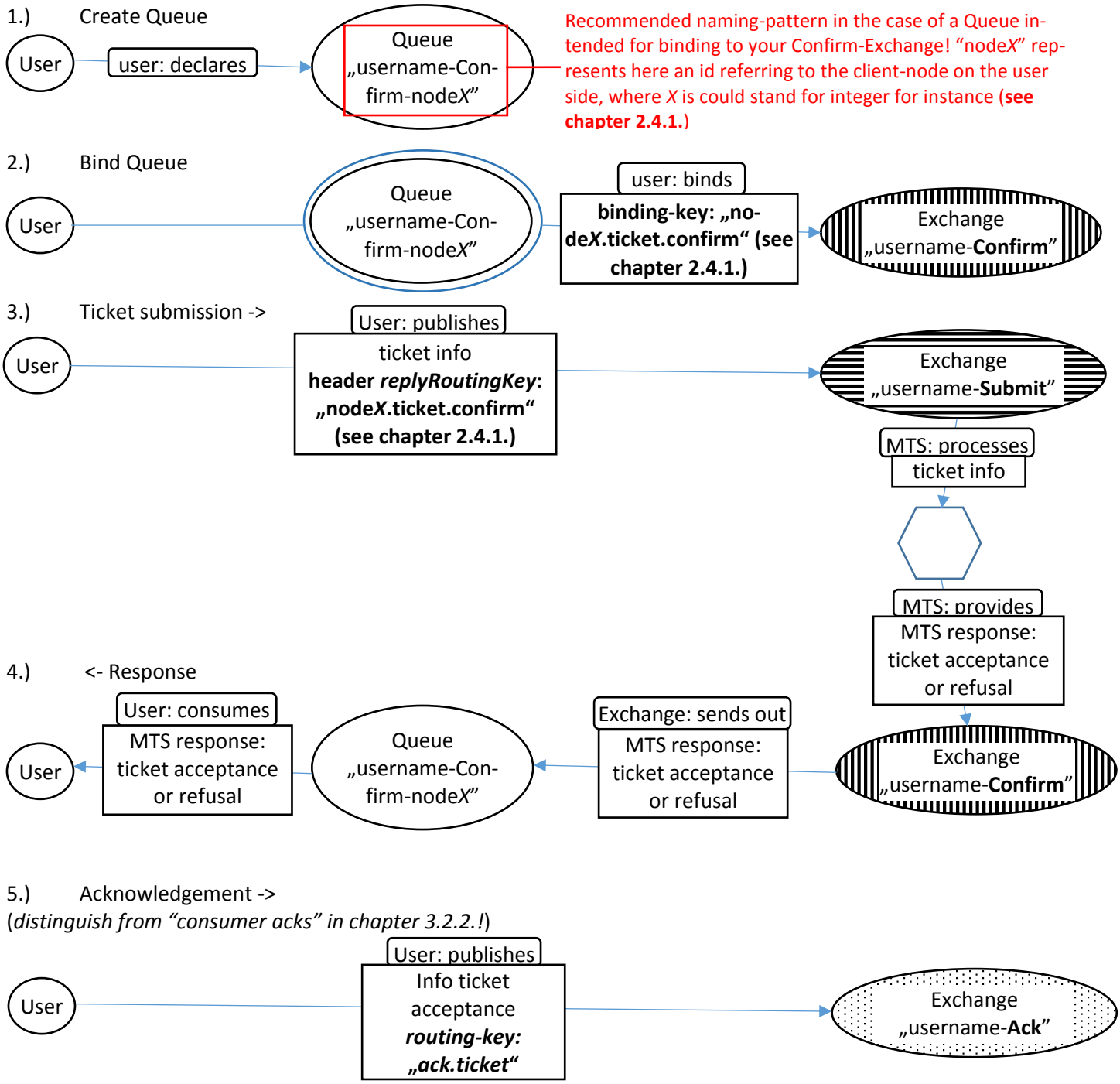
Here "username-Reply-" and "username-Control-" are strings whereas "–node*X*" represents the id you dedicate internally on your side to the client-node within your cluster that you wish to receive the MTS response on (typically the one you send the related message to MTS in the first place with), of which the *X* stands for an integer. (**see chapter 2.4.1.)**

In the end you are of course, free to disregard this recommendation and name your Queue the way you like as long as you follow the earlier mentioned naming-convention.

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

2.3. Currently supported message flows

2.3.1. Ticket submission + Response + Acknowledgment

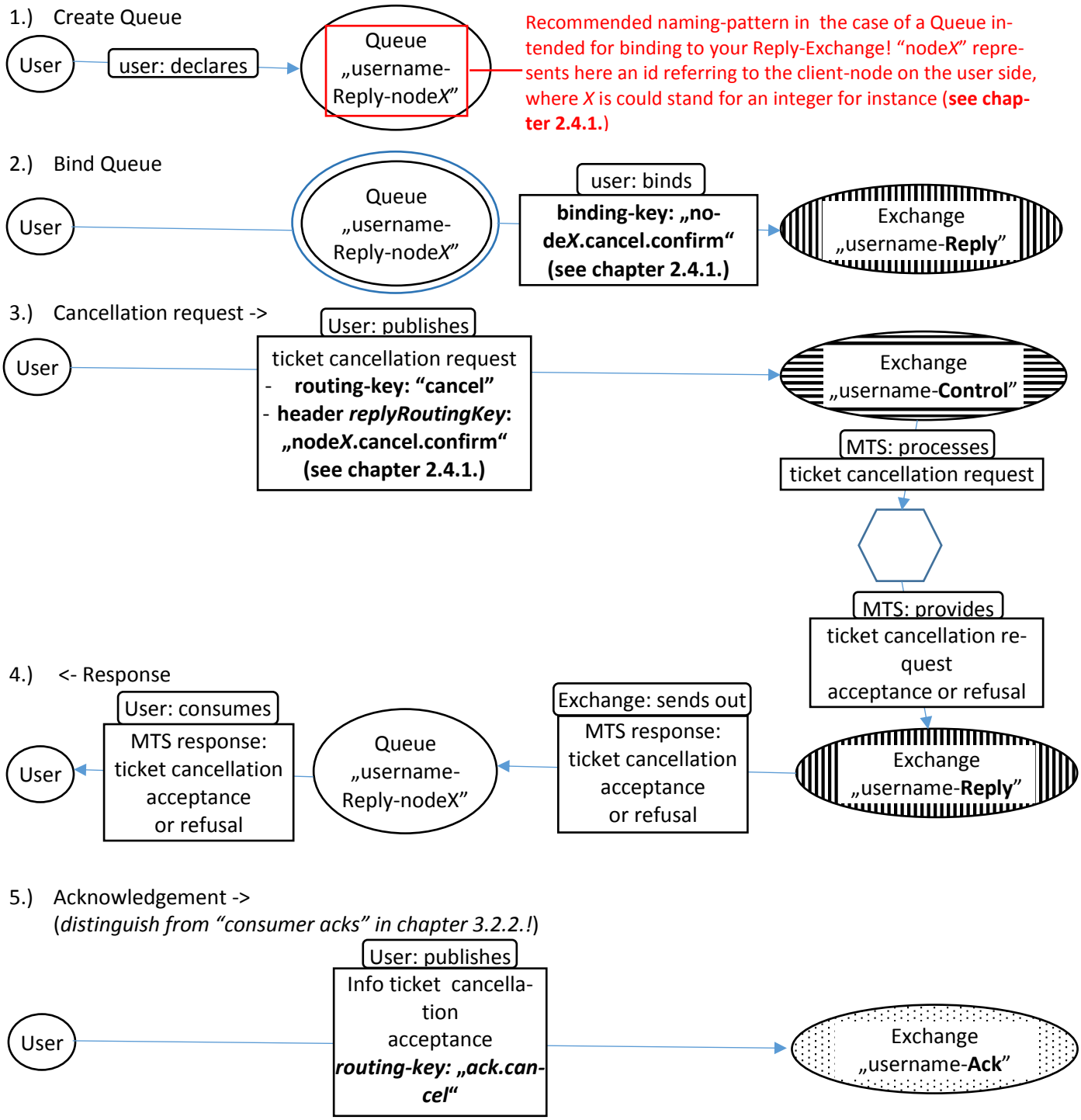


Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

1.) The user declares a Queue named “username-Confirm-node*X*” … his username followed by a dash followed by the string “Confirm” followed by a string (in place of “-node*X*”).
Note: Queue names have to be unique within one vhost! Thus we recommend that “node*X”* represents an id referring to the client-node on the customer side he wishes to receive the respective MTS response on. If the recommendation of chapter 2.2.4.2. is followed, the Queue names are going to be strings like “*myMTSUserName-Confirm-node1*”, “*myMTSUserName-Confirm-node2*”, “*myMTSUserName-Confirm-node3*” …, or deviating strings, if the recommendation is disregarded deliberately, but strings that are within the range that is lined out by the Queue naming-convention of chapter 2.4.2.1. **Also see chapter chapter 2.4.1.**
2.) The user binds his Queue to the Exchange named “username-Confirm”… his username followed by a dash followed by the string “Confirm”. Here he applies a binding-key that equals the message header *replyRoutingKey* described in the *Note of point 3* below.
3.) Then he can publish his ticket (ticket info) to the Exchange named “username-Submit”…his username followed by a dash followed by the string “Submit”.
NB!: Submitted tickets (ticket info or JSON object) require a message header named *replyRoutingKey* whose value, a non-empty valid string, will be used to route the respective MTS response (ticket acceptance or refusal) to the Queue that was bound with the same binding-key value. **Also see chapter chapter 2.4.1.**
4.) As soon as the MTS response (ticket acceptance or refusal) arrives in the Queue the user can consume it.
5.) The user informs MTS if the recommendation contained in the MTS response (to accept or refuse the ticket) was followed on his side or not. Therefore he publishes his response-acknowledgement (info ticket acceptance) to the Exchange called “username-Ack”… his username followed by a dash followed by the string “Ack”. Info ticket acceptance messages have to be sent with the routing-key “ack.ticket”.
Note: Don’t confuse these acknowledgements with the “consumer acks” as described in chapter 3.2.2.!

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

2.3.2. Ticket cancellation + Response



Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

Preface: Messages regarding a ticket cancellation flow in both directions via Exchanges of the type “topic”. Therefore
a.) A meaningful routing-key and binding-key (for response queues) are required.
Since the topic is the cancellation of tickets the simple string “cancel” is used as routing key for messages towards MTS concerning this topic.
This means MTS binds its internal Queues with a binding-key to listen for cancellation request messages incoming with a routing-key “cancel”.
b.) However cancellation requests require a message header (exactly as ticket submissions do, see chapter 2.3.1.) which equals the binding-key of the targeted Queue of response - ***in addition to a routing-key***. **Also see chapter chapter 2.4.1.**

1.) The user declares a Queue named “username-Reply-node*X*” … his username followed by a dash followed by the string “Reply” followed by a dash followed by a string (in place of “-node*X”*).
Note: Queue names have to be unique within one vhost! Thus we recommend that “node*X”* represents an id referring to the client-node on the customer side he wishes to receive the respective MTS response on. If the recommendation of chapter 2.2.4.2. is followed, the Queue names are going to be strings like “*myMTSUserName-Reply-node1*”, “*myMTSUserName-Reply-node2*”, “*myMTSUserName-Reply-node3*” …, or deviating strings, if the recommendation is disregarded deliberately, but strings that are within the range that is lined out by the Queue naming-convention of chapter 2.4.2.1. **Also see chapter chapter 2.4.1.**
2.) The user binds his Queue to the Exchange named “username-Reply”… his username followed by a dash followed by the string “Reply”. While binding he applies a binding-key that equals the value of the message header *replyRoutingKey* as described in the *Note of point 3* below.
3.) Then he can publish his ticket cancellation request to the Exchange named “username-Control”…his username followed by a dash followed by the string “Submit”. He publishes these requests with the routing-key “cancel”.
Note: Ticket cancellation requests (ticket infos) additionally require a message header named *replyRoutingKey* whose value, a non-empty valid string, will be used to route the respective MTS response (ticket cancellation acceptance or refusal) to the Queue that was bound with the same binding-key value. **Also see chapter chapter 2.4.1.**
Note2: Cancelation messages are processed in asynchronous way. MTS will generally process cancellations at the time of their arrival but there might be cases where response will be delayed for a couple of seconds or even minutes. MTS still makes guarantee that a cancellation request will eventually be processed, but you are allowed to re-send a cancellation only once after a timeout period of 1 minute. This operation is idempotent.
4.) As soon as the MTS response (ticket acceptance or refusal) arrives in the Queue the user can consume it.
5.) The user informs MTS if the recommendation within the MTS response (to cancel the ticket or not) was followed on his side or not. Therefore he publishes his response-

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

acknowledgement (info ticket cancellation acceptance) to the Exchange called “username-Ack”… his username followed by a dash followed by the string “Ack”.Info ticket cancellation acceptance messages have to be sent with the routing-key “ack.cancel”.
Note: Don’t confuse these acknowledgements with the “consumer acks” as described in chapter 3.2.2.

2.4. Message control

2.4.1. Determining the response reception Queue

The user is required to communicate to MTS to which Queue he wants each of the MTS responses to each of his messages, be it ticket submissions or cancellation requests, routed to. This information comes in handy for sorting the responses in single node client environments on the customer side. However it is a necessary requirement for clustered client environments, where multiple nodes work independently of each other and the MTS customer has no possibility to match the outgoing messages with the incoming MTS responses on all nodes. In other words, without you predetermining the route for the response from MTS in your message towards MTS, we can’t guarantee that the response will be relayed to a Queue on the particular node that the message was initially sent on.

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

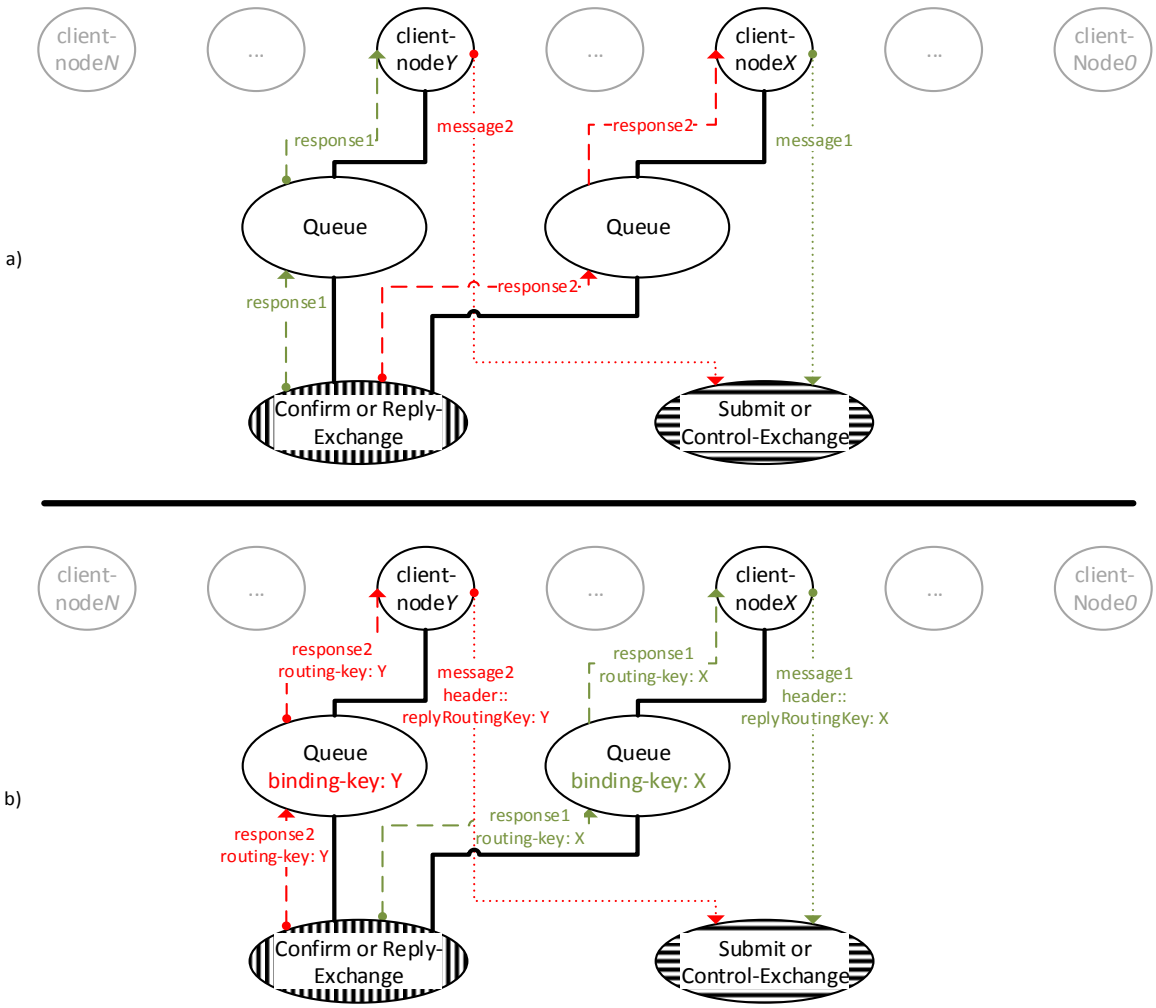


Fig 2.4.: a) Without predetermining the reply target in advance the MTS response might be sent to a different node its corresponding message was originally sent from.
b) Specifying the reply target in advance works via the field *replyRoutingKey*, which you set in the header of your message towards MTS. The corresponding MTS response will be sent with a routing key equaling your replyRoutingKey value to your Exchange of response. If you happen to have a Queue bound with the same binding-key value, the response will be relayed there successfully.

2.4.1.1. replyRoutingKey convention

The *value* of this header field must contain any valid non-empty string as a prerequisite for a successful response relay. However, it must match at least one binding-key of at least one Queue you bound to the respective Exchange of reply in order to be eventually relayed to you, since this value will be used as the routing-key for the MTS responses.
Note: If you don't set a valid replyRoutingKey for tickets, it will be set by MTS automatically to the string "*not_set*".

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

If you don't set a valid replyRoutingKey for cancellation requests it will be set by MTS automatically to the string "*cancel"*.

2.4.1.2. Recommendation

We recommend the following values if it comes to the replyRoutingKeys and the matching binding-keys for your Queues (for Queue naming recommendations see chapter 2.2.4.2):

message-type	**replyRoutingKey & Queue binding-key**
ticket (info)	node*X*.ticket.confirm
cancellation request	node*X*.cancel.confirm

For simplicity reasons we recommend to use "node" as a literal string and a cluster-unique character representing an integer (optimally starting with 1 and incrementing by 1 within your cluster: node1, node2, node3) in place of "*X*".

2.4.2. correlationId

For identifying which MTS response corresponds to which message sent initially by you towards MTS we offer a more convenient option, apart from a tedious parsing of the message payload in order to then compare the ticket-id.
A more convenient and efficient way is having you set the message property field correlation_id. Its value will be found then on the corresponding MTS responses in their header field of the same name.
Note: There is sometimes confusion in regards to naming of this field correlation_id vs correlationId: correlationId is something you will find among used AMQP library methods during message publishing. correlation_id is actual field name that is sent.

3. Best practices & known problems

3.1. Handling of connections and channels

Connections are TCP based and require authentication.
You normally establish it by the means of your AMQP API to MTS RabbitMQ cluster, that contains your targeted vhost while logging into it with your credentials (username/password – see chapter 2.2.2.).

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

3.1.1. Channels

Channels and their creation are lightweight. Thus many channels / connection shouldn't be problematic. So feel free to segregate the message flow by using separate channels.
It is even recommended to use separate channels for publishing and consuming in any case!
When not used anymore channels should be closed, though fewer long-lasting channels seem preferable instead of a constant opening and closing of channels.

3.1.1.1. Thread safety

Neither the Java-client nor the .NET-client channels are considered "ultimately" thread safe. Thus it is recommended to use one channel / thread.
http://www.rabbitmq.com/api-guide.html#channel-threads
http://www.rabbitmq.com/dotnet-api-guide.html#model-sharing

These and other best practices regarding channels can be found here:
https://dzone.com/articles/rabbitmq-amqp-channel-best

3.1.2. Connections

Generally goes – the fewer connections you establish – the better. If your system topology or internal processes don't require multiple or alternating connections, just open one connection and keep it up as long a particular messaging flow is ongoing.
As soon as one can regard a particular message flow done for a reasonable time though you should close that particular connection. In fact, if for any reason you should decide to abandon a connection you have to close it respectively! However, one long-lasting connection at a time appears to be desirable.

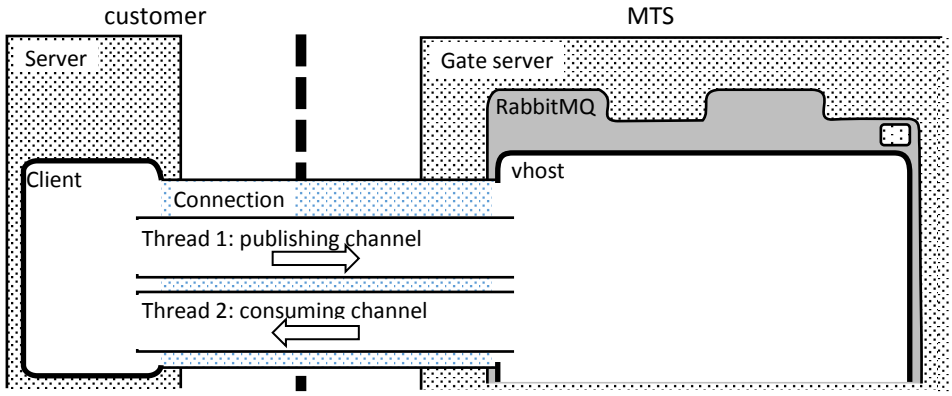


Fig 3.1.: Your needs might deviate in number of connections and especially channels from this line-out. However this line-up represents the desired minimum configuration of connections and channels at a time.

3.2. Message handling

3.2.1. Durability and memory threat

Durable messages pose a potential memory threat to the broker. Therefore in high message load systems, such as MTS, none of the messages may be durable within RabbitMQ.
Since RabbitMQ is not a DB your messages to MTS will be safely stored somewhere else anyhow. Thus we strongly *discourage* you to set the durability parameter of your messages to

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

"true". In any case, MTS has mechanisms in place in order to counteract the durability of customer messages. This above means **delivery mode** for message itself must be set to **1.**

If it comes to responses from MTS - we strongly encourage you to consume all your responses from your Queue on time. Not consumed responses pose, again, a potential memory threat.

3.2.2. Delivery guarantees

There is of course no ultimate delivery guarantee. However, in order to improve the existing mechanisms it is recommended to confirm / acknowledge sent messages.

After having consumed MTS responses from a Queue you should confirm that you have received those messages via basic acknowledgements. We recommend to send these acknowledgements in any scenario.

In the other direction, if it comes to get confirmations for your published messages the application of publisher confirms is thinkable (see chapter 3.2.2.2.).

3.2.2.1. Consumer Acks

We recommend you to send us acknowledgements after you have intentionally consumed them, as opposed to sending automatic acknowledgements or not sending any acknowledgements at all (don't confuse these acks with the acknowledgements as described in chapter 2.3.!). This you can achieve by a call of the "basic.ack"-method of the channel. However, as it is generally a good practice to minimize the frequency of network-calls, it is recommended to send them in reasonable intervals, after having consumed a batch of MTS responses.
Note that in order to make sense of intentional acknowledgements you should set the "autoAck"-parameter in your "basicConsume"-method call to false.
Note further that the channel you dedicate to consuming messages should stay transactional, since Acknowledgements have to be sent via the same channel as the respective message they were consumed from and since the transactional mode is the one Acknowledgements operate in.

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

3.2.2.2. Publisher Confirms (advanced publishing safety scenario)

In an advanced scenario, when a heightened publishing safety is required, it is advised to make use of publisher confirms: https://www.rabbitmq.com/confirms.html.
It is a good practice to either:

1) send messages in batches and then to wait for respective publisher confirms, or
2) handle confirms asynchronously.

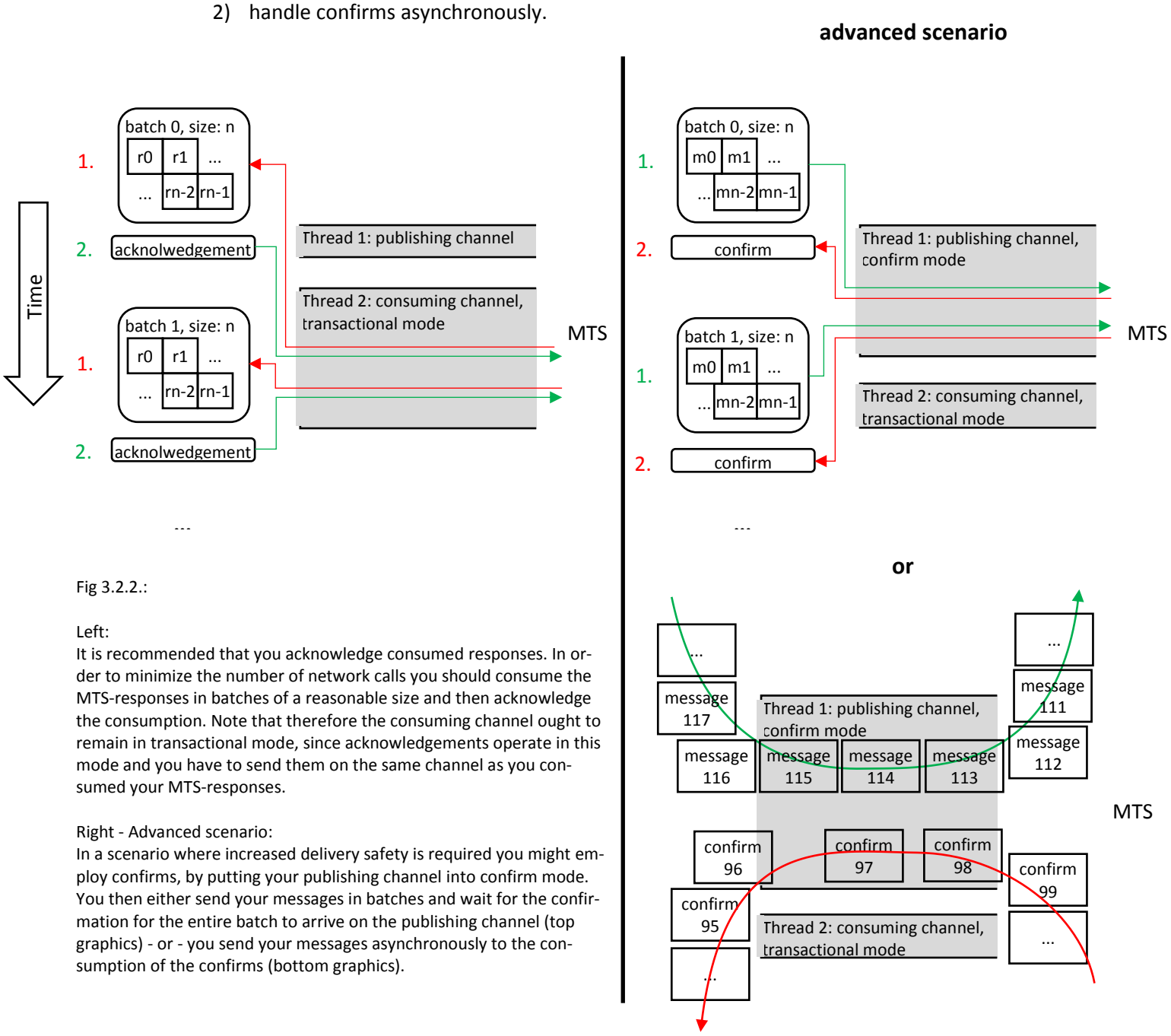


Fig 3.2.2.:

Left:
It is recommended that you acknowledge consumed responses. In order to minimize the number of network calls you should consume the MTS-responses in batches of a reasonable size and then acknowledge the consumption. Note that therefore the consuming channel ought to remain in transactional mode, since acknowledgements operate in this mode and you have to send them on the same channel as you consumed your MTS-responses.

Right - Advanced scenario:
In a scenario where increased delivery safety is required you might employ confirms, by putting your publishing channel into confirm mode. You then either send your messages in batches and wait for the confirmation for the entire batch to arrive on the publishing channel (top graphics) - or - you send your messages asynchronously to the consumption of the confirms (bottom graphics).

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

3.3. Queue handling

Since the Queues shall reflect your very custom system topology of servers, clients, connections and channels down to the Queue-number and -distribution of your need, it is up to you to declare and bind the Queues you intend to consume messages from.
Therefore you should get familiar with Queue properties and their meaning.

3.3.2. Queue properties

There is a comprehensive list of Queue properties, as well as domains in general found in the "General AMQP reference" on the RabbitMQ-site -> section "Queue" -> method-section "declare". There the relevant Queue property information can be found.

3.3.2.1. Recommendation

We recommend you the following settings for the Queue-parameters:

parameter	**recommended value**	**comment**
durable	set (true)	If you do not want to re-declare the Queues in case of an unexpected connection loss (e.g. a broker restart on our side) you might want to persist them to the disk, which would let them survive a broker restart. This is achieved by declaring them durable.
auto-delete	unset (false)	'Auto-delete = true' would mean that the Queue gets deleted when the last consumer unsubscribes. That would contradict the purpose of having durable Queues. Thus auto-delete = false.
exclusive	unset (false)	Setting this parameter would make a Queue exclusive to the connection it was declared within. Since connections are not durable you can't have durable Queues exclusive to a connection. Thus exclusive = false.
passive	unset (false)	A passive declaration serves only the purpose to invoke a "Declare-OK" response, if a Queue of that name already exists within the vhost, or a channel exception if not. Usually used only to verify the existence of a specific Queue since this declaration doesn't create a Queue on the server. Thus passive = false.
no-wait	unset (false)	You might want to wait for a server response to make sure your Queue was created (or found). Therefore no-wait = false.
arguments	unset (null)	Not expected.

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

4. References

The collection of the references here makes for a basic guide. It is in no way exhaustive, especially if it comes to links concerning particular programming languages.
You are hereby encouraged to do your own research!

4.1. RabbitMQ Concepts

https://www.rabbitmq.com/tutorials/amqp-concepts.html

4.2. AMQP client libraries (as offered on the RabbitMQ-site)

General download site:
https://www.rabbitmq.com/download.html
(Note that for the MTS integration with RabbitMQ no server setup on the your side is required)

- **Java:**
  https://www.rabbitmq.com/java-client.html

- **.NET:**
  https://www.rabbitmq.com/dotnet.html

- **PHP:**
  https://github.com/videlalvaro/php-amqplib

4.3. RabbitMQ Documentation

General site:
https://www.rabbitmq.com/documentation.html

- **Java:**
  https://www.rabbitmq.com/documentation.html

- **.NET:**
  https://www.rabbitmq.com/releases/rabbitmq-dotnet-client/v3.4.3/rabbitmq-dotnet-client-3.4.3-client-htmldoc/html/index.html

- **PHP:**
  http://de.slideshare.net/old_sound/integrating-php-withrabbitmqzendcon
  (Introduction to RabbitMQ with PHP's Zend framework)

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

4.4. Tutorials

General site:
https://www.rabbitmq.com/getstarted.html

General AMQP reference:
https://www.rabbitmq.com/amqp-0-9-1-reference.html

Quick reference Guide:
https://www.rabbitmq.com/amqp-0-9-1-quickref.html

- **Java:**

A comprehensive guide starting with lesson one:
https://www.rabbitmq.com/tutorials/tutorial-one-java.html

Master tree:
https://github.com/rabbitmq/rabbitmq-tutorials/tree/master/java

- **.NET:**

A comprehensive guide starting with lesson one:
https://www.rabbitmq.com/tutorials/tutorial-one-dotnet.html

Master tree:
https://github.com/rabbitmq/rabbitmq-tutorials/tree/master/dotnet

- **PHP:**

A comprehensive guide starting with lesson one:
https://www.rabbitmq.com/tutorials/tutorial-one-php.html

4.5. Additional Notes

If you want to use the PHP libraries offered here without in-depth knowledge of their abstraction structure - a PHP dependency manager (autoloader) is recommended, due to the abstraction level found in the respective libraries.
Here a link to the common manager used also in the official tutorials:
https://getcomposer.org/
Nevertheless, feel free to use a PHP dependency managing tool (and generally the "AMQP PHP client library") of your choice.

You can find support for languages not covered in this document and in general all sorts of links and resources here: https://www.rabbitmq.com/devtools.html.

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404