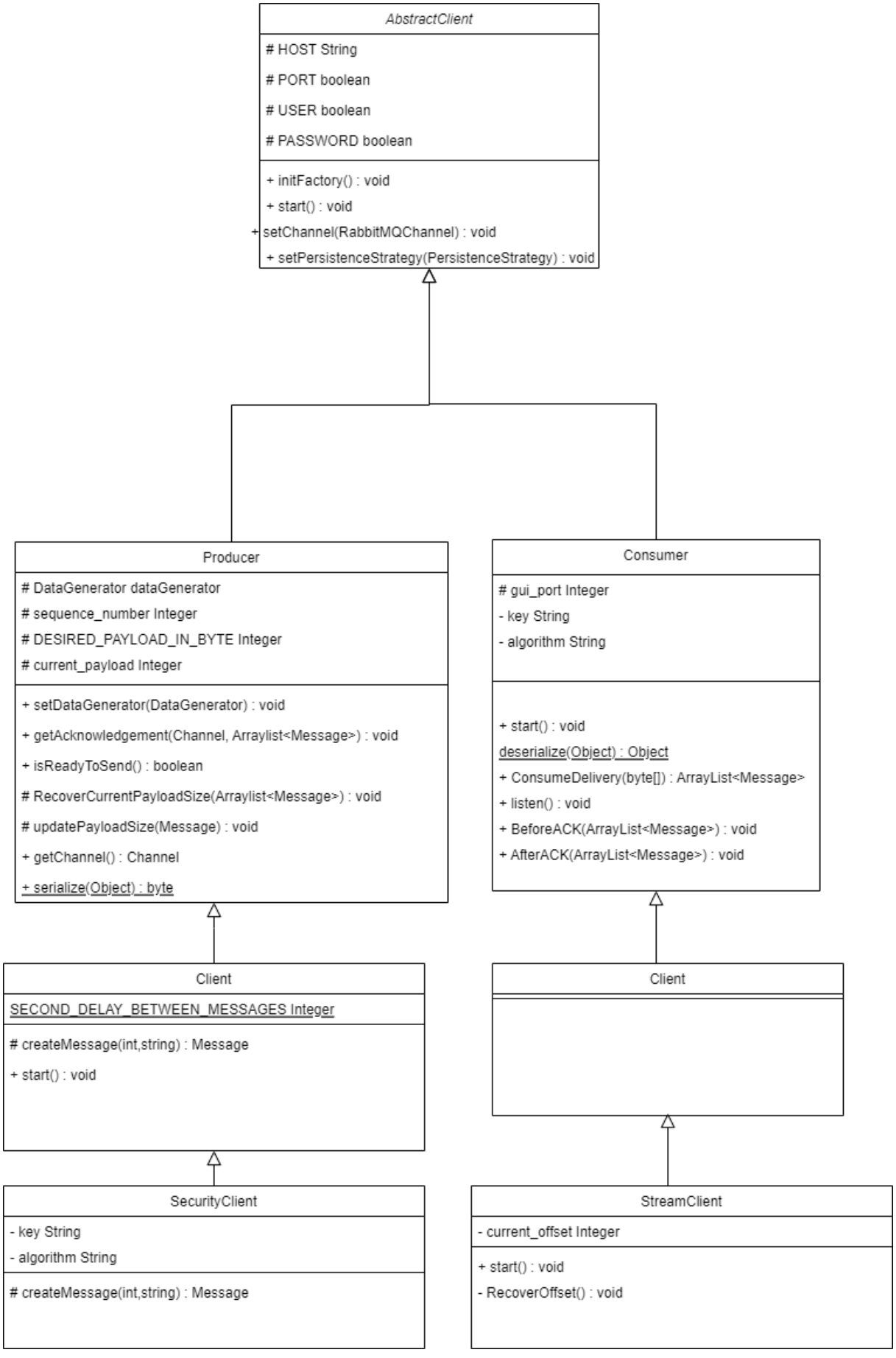
Producer Consumer Documentation

Introduction

The Producer and Consumer are one of the core Features of our Project.

Both are so designed that it easy to write your own Producer and Consumer.

Producer Consumer Overview



Producer

The Producer is a class which implements the basic functionality every Producer needs

- Sending the data to the RabbitMQ Queue via ACK to confirm, that the Message was received
- Recover current State in case of an unexpected shutdown
- Serialize Messages
- Set the desired Payload e.g. if you want a minimum size of Payload. If you set a desired Payload it is recommended to use the AggregateMessagePersistance, because the FilePesistence only stores the newest Message into a File while the first stores every Message into a File.

Client

The Client itself is a subclass of the Producer and utilizes every Method of it in order to actually communicate with the RabbitMQ. With the help of the DataGenerator the client reads these Events and assigns a unique Number to it. A Client is just sending normal Messages.

Security Client

A SecurityClient is an extended Client which enables an user to create HMAC Messages. These Messages have a Message Authentication Code which enables to detect if a Message was tempered with.

Consumer

Just like in the Producer the Consumer implements basic functionality every Consumer will need.

Apart from the normal functionality a Consumer also enables a User to connect to. This is Feature which might be from interest for a custom Consumer to revive Information later on.

Also it is important to understand, that the compared to the SecurityClient in Producer, a Consumer holds the key and also the algorithm the reason for that is, that using a Stream and Queue greatly differs and with this Design decision it is not necessary to also create a SecurityStreamConsumer. However it is not needed to give the Consumer a key. Without a Key the Consumer won´t try to verify a HMAC Message.

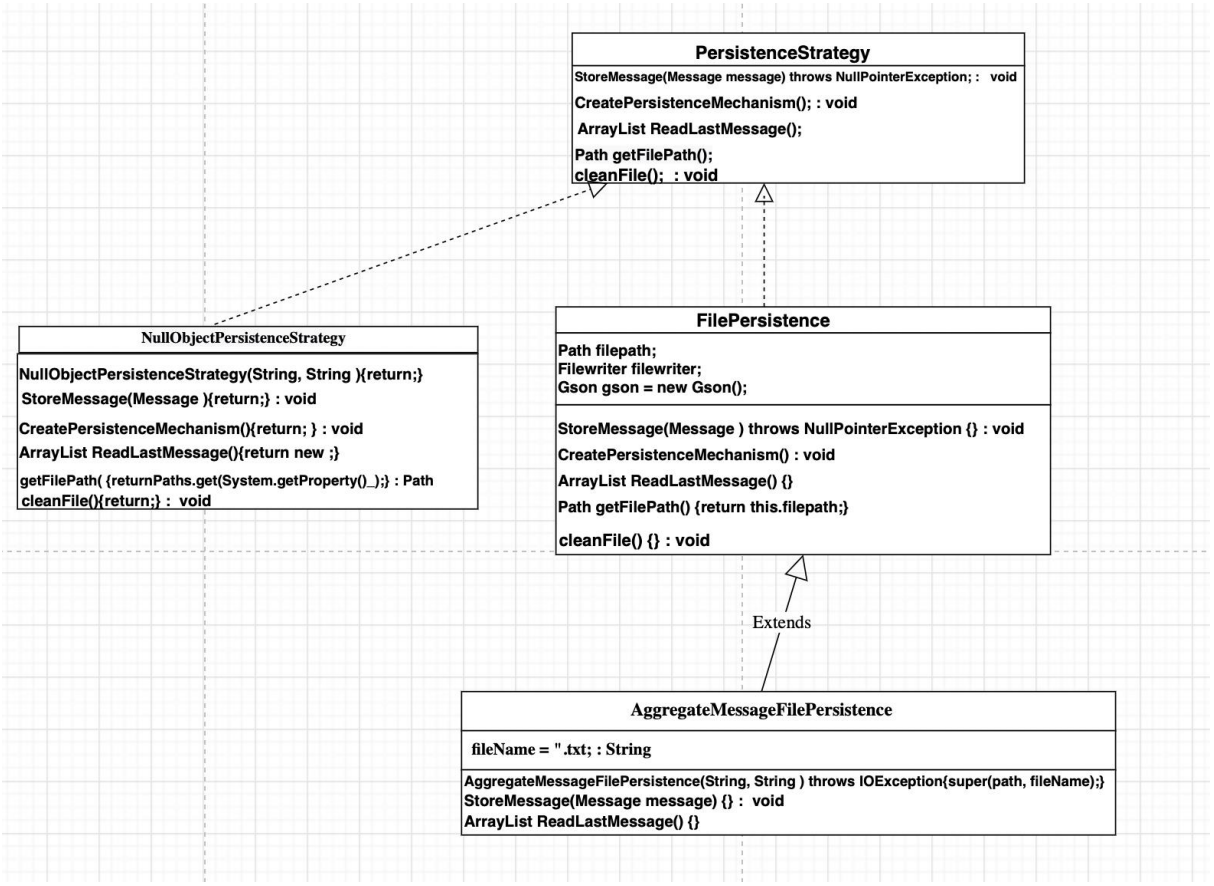
Client

For uniformity purpose there is also a Client in a Consumer, but essentially a Consumer and a Consumer.Client does not differ.

StreamClient

As mentioned in the Consumer, a Stream and a Queue greatly differ so it was necessary to implement a StreamClient. Since a Streamclient can read from a Stream at any point an unexpected shutdown recovery is not needed. Instead the Streamclient remembers it´s last offset which reduces the load of the Server, since it does not have to track the offset of Consumers.

Persistence Strategy



The Persistence Strategy is something a Consumer and a Producer utilize to be able to persistently store Data. In this Project there are the following Implementations
(Since an event is stored in a Message both terms mean the same in this context) :

NullObjectPersistenceStrategy

It follows the NullObjectPattern and simply does nothing in it´s methods. It is recommended to use this one if you can live with missing Data, but compared to the other implementations it does not stress the HDD since it does not actually persistently store Data in the Filesystem.

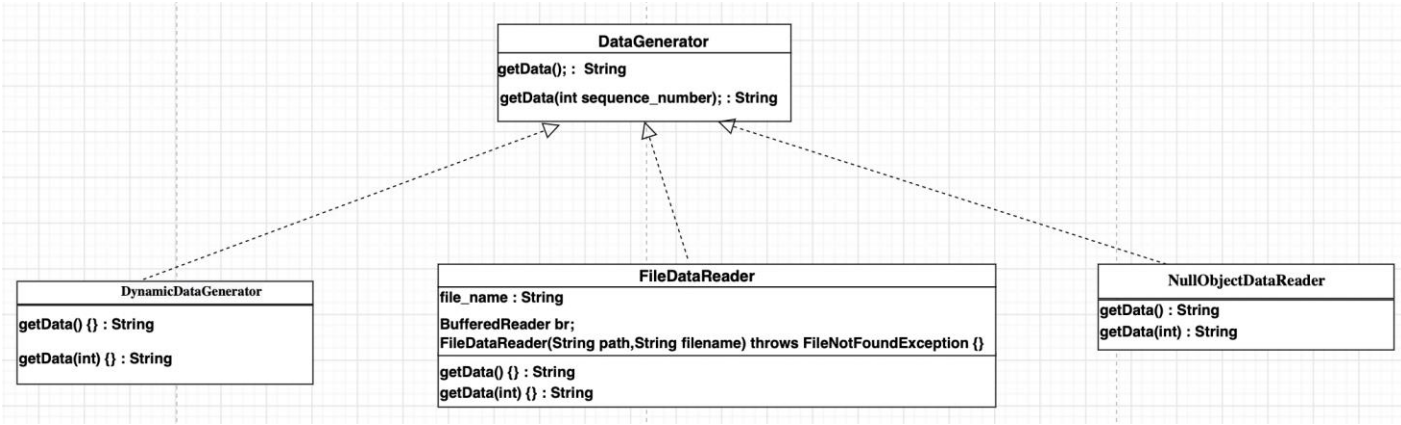
FilePersitenceStrategy

Stores always the last created Event in a File.

AggregateMessageFilePersitenceStrategy

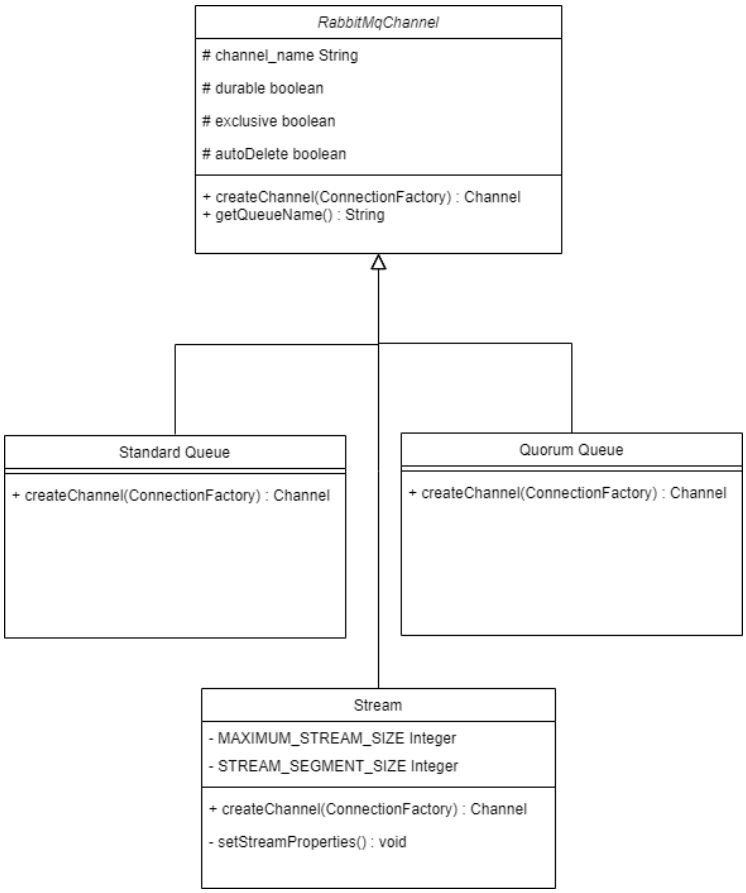
Stores every Message into a File (appends the File)

DataGenerator



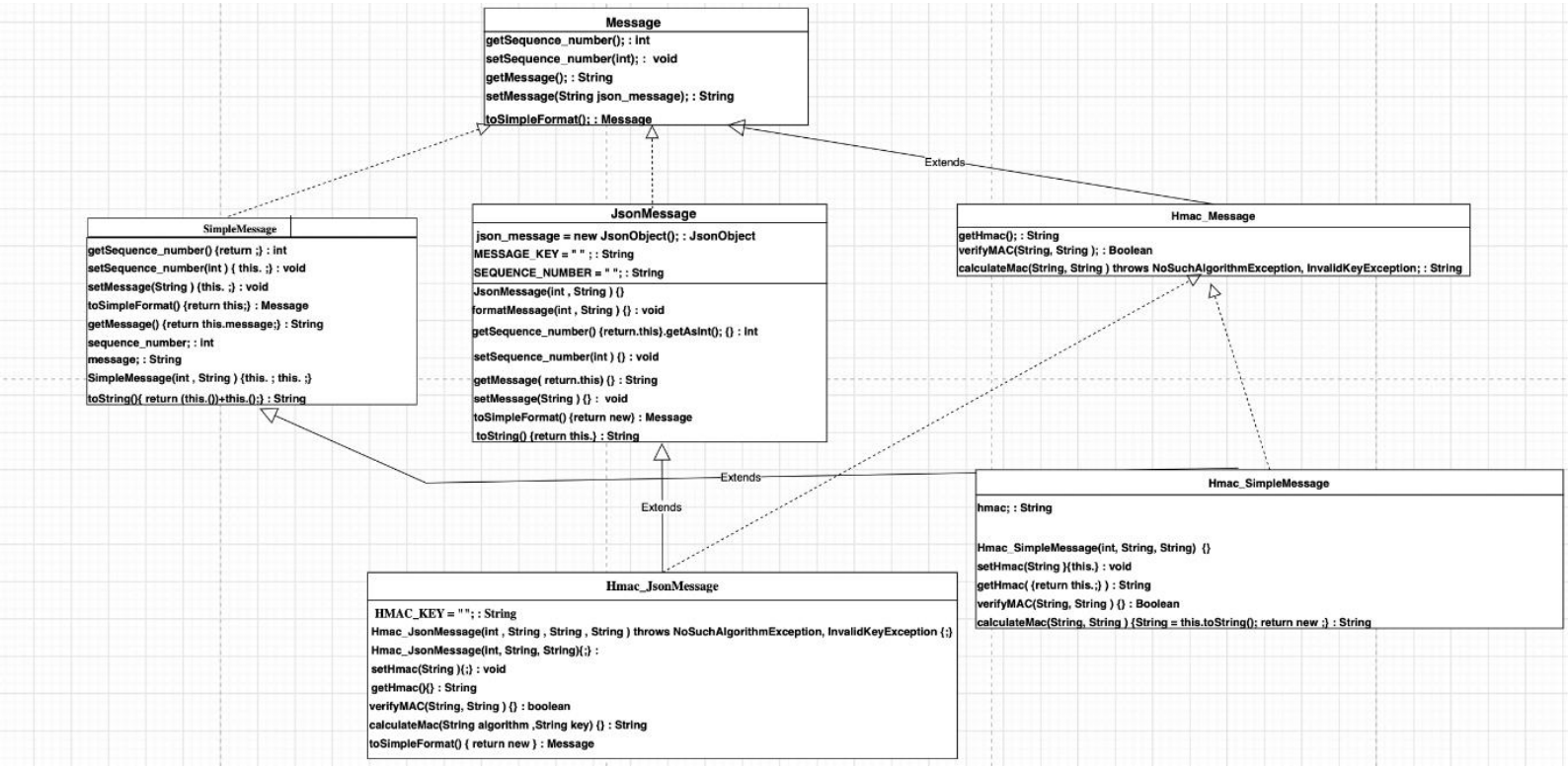
The Datagenerator is the interface which is used by the Producer to create or get Data. For our Purpose the DataGenerator just returns more or less random Data or reads it from a File.

RabbitMQChannel



A RabbitMQChannel is a Wrapper which makes it easier to use the different types of Queues/Stream which RabbitMQ provides. Both the Producer and Consumer needs a RabbitMQChannel in order to be able to connect to the RabbitMQ Server.

Message



A Message is used by the Producer and Consumer. It is used to have an uniformed Interface so get the Message(string) (the event itself) and Sequence Number. While the SimpleMessage is working the JsonMessage is not. For it work you would have to implement another serialize function.

An Hmac_Messgae is an extended Message which extends the functionality of the Message with an MAC (Message authentication Code). With this it is possible to verify if a Message was tempered with or not.