* Each file is separated into messages.
* I have not seen a message split into many files, because the files we have are 16M at max, and our messages are 4M at max. I don’t know how Rabbit would handle that.
* The chaining of messages is done through an int64 which contains the length of ‘this’ message.
  1. For instance, the first 8 bytes of ‘snap1’ have a value (little endian) of 0x5635.
  2. In ‘snap2’, if we go to 0x5635 + 8, we get to 0x563D. There we get a 0xFF (end of message), and either the end of the file, or the length of the next message, here 0x29F7.
* Each message is separated into parts/blobs.
* When handling a new message, I look for ‘rabbit\_framing\_amqp\_0\_1’, from where I can get the start of the blobs chain.
  1. I could have tried to decode the kind of ‘structure’ Rabbit writes, but decided against it, since variable-length strings are involved and complicate things. Using the prefix is easier.
* Message blobs are chained backwards, i.e. the first one we encounter in a chain is the last blob in the message.
* Following the ‘rabbit…’ prefix, we have 2 int32 values:
  1. In ‘snap1’, we have
     + 0x6C then ‘1’ (green rectangle): This is the number of parts/blobs in this message.
     + 0x6D then ‘0x555B’ (blue rectangle): This is the length of this blob, starting right here.
  2. In ‘snap3’, we have
     + 0x6C then ‘2’: So we have 2 parts.
     + 0x6D then ‘0xE665’: The length of the first blob.
  3. In ‘snap4’, at the offset where ‘snap3’ points to, we have
     + 0x6D then ‘0x1FFF8’: The length of the second and last blob of this message.
  4. All we have to do is concatenate the blobs to obtain the final message.
* We have to handle 2 kinds of messages.
  1. The ones pushed by Cmf (from C++).
     + There is an ‘ascii’ prefix (more on this below), followed by a zipped binary encoding of the message.
  2. The ones pushed by Papicon (Push Api Consumer) (from Java).
     + There is a special ‘rabbit property header’, that will trigger a special handling in the Cmf reader. The payload itself is a zipped json. The json represents the same document as the ones pushed from C++, only less compact.
* The payload is not important. It is re-published as is. What we care about is the name of the target queue.
* For ‘Cmf’ messages, like in ‘snap1’, I have to extract the queue name from something like “i81i{4'1waybt3'url41'ces.gmproductione2yq29g7-koflgbm4.Dpm.Doc3'veru20171020u3'zipbt}”
* For ‘papicon’ messages, like in ‘snap5’, I have to extract the queue name from something like “{url:ces.gmproductione2yq29g7-koflgbm4.Dpm.Doc,method:Process,zip:true}”
* For messages fitting into index (\*.idx) files, there is no chaining. I use almost the same algo though, looking for ‘rabbit\_framing\_amqp\_0\_9\_1’ and fetching the message just a few bytes further.