**Team**: 05, Eugen Deutsch, Ralf von der Reith

**Aufgabenaufteilung**:

1. Ralf: Entwurf Server und Nachrichtenverwaltung (HBQ, DLQ)
2. Eugen: Entwurf Client, Prüfung von Server & Nachrichtenverwaltung.

**Quellenangaben**: -/-

**Bearbeitungszeitraum**:

1. 01.04.2017 – 2 Stunden
2. 02.04.2017 – 1 Stunde
3. 03.04.2017 – 3 Stunden
4. 04.04.2017 – 2 Stunden
5. 05.04.2017 – 4 Stunde
6. 06.04.2017 – 3 Stunden

➔ 15 Stunden

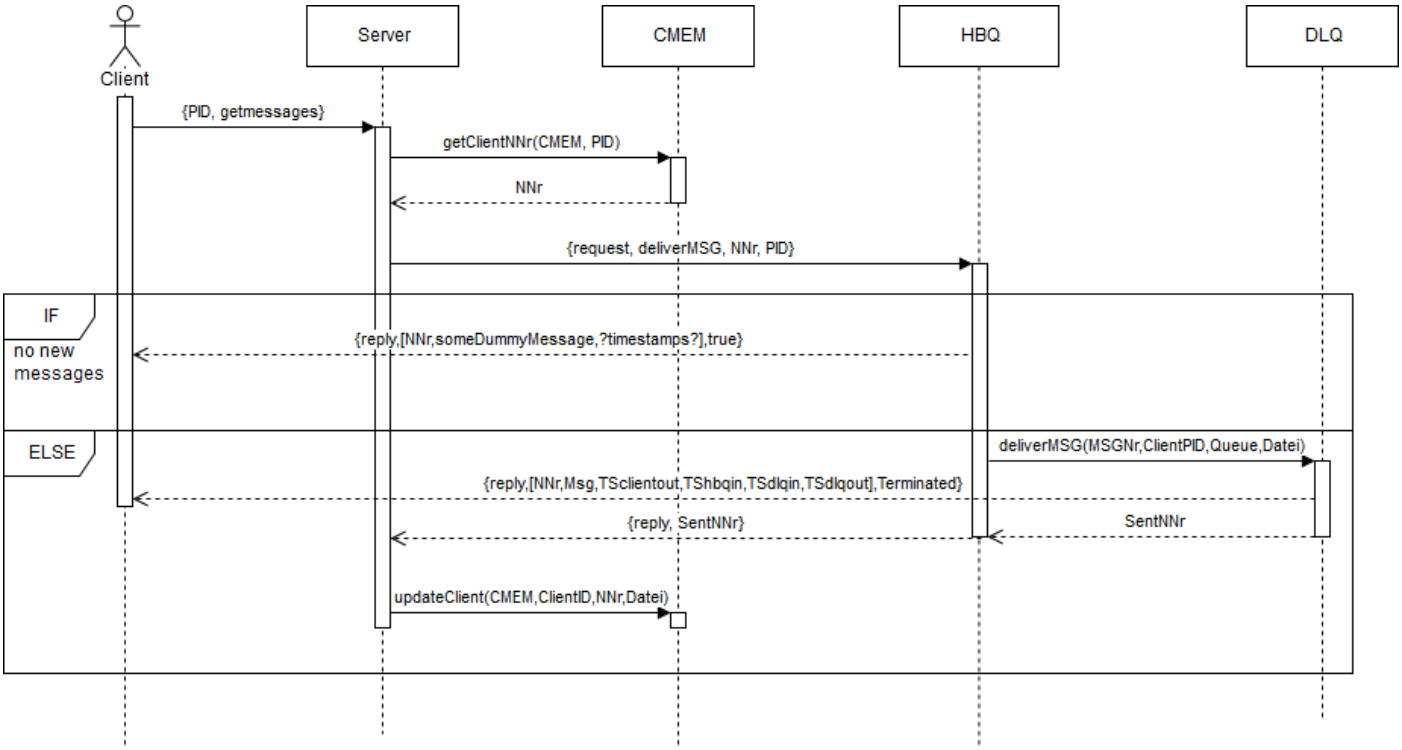
**Aktueller Stand**: Der Entwurf ist fertig

**Änderungen des Entwurfs**: -/-

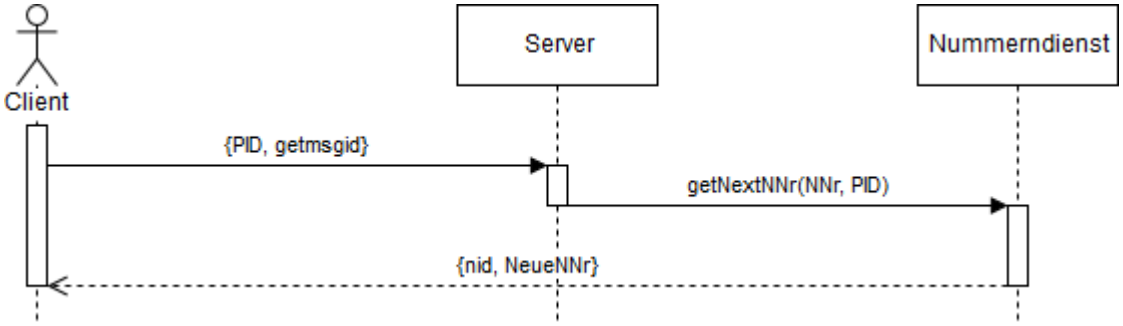
## Erläuterung des Ablaufs (Server)

Der Server besteht aus der Nachrichtenvermittlung und dem CMEM, einem Speicher für die Leserklienten, der nur vom Server angesprochen wird. Bei Anfrage eines Lesers nach einer Nachricht fragt dieser beim CMEM nach der nächsten Nachrichtennummer für den gegebenen Klienten. Sofern der Leser seit X Sekunden keine Nachrichten mehr angefragt hat, wird dieser vergessen und wie ein neuer Klient behandelt, sodass die erste Nachrichtennummer gewählt wird (niedrigste Nummer in DQ).

Nun fragt er in der Nachrichtenverwaltung nach der Nachricht, die dann dem Klienten entweder die entsprechende Nachricht übermittelt und einen Flag mitliefert, der zeigt, ob dies die letzte für ihn war, oder eine Dummynachricht verschickt, die aussagt, dass es für diesen Leser seit der letzten Anfrage keine neuen Nachrichten gibt.




Ansonsten verschickt er bei Anfrage eines Redakteurs eine einzigartige, aufeinanderfolgende und bei 1 beginnende Nachrichtennummer und nimmt von diesem die Nachrichten entgegen, die er dann an die Holdbackqueue weiterleitet.




Der Server terminiert, nachdem er zulange keine Anfragen bekommen hat.

## Erläuterung des Ablaufs (Nachrichtenverwaltung)

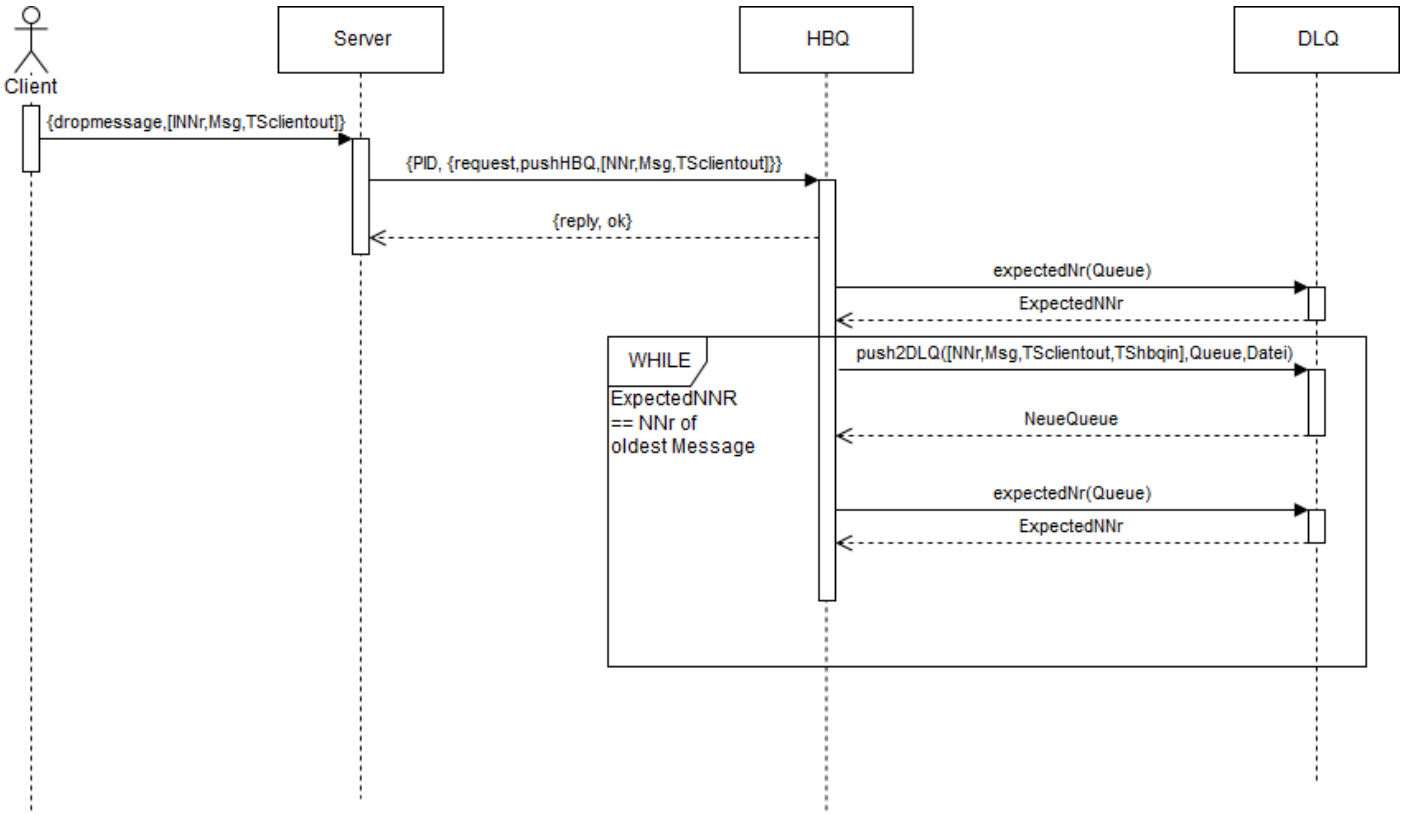
Die Nachrichtenverwaltung besteht aus einer Holdbackqueue und einer Deliveryqueue. Beide sind als Listen mit Tupeln realisiert, welche die Nummer, den Inhalt, sowie die verschiedenen Zeitstempel der Nachrichten enthalten (=> [(Nachrichtennummer,  Nachricht, TSclientout, TShbqin[, TSdlqin)]*).

*TSdlqin wird nur in der Delivery-Queue verwendet.

Die Holdbackqueue enthält die Nachrichten der Redakteurklienten. Nachrichten aus der HBQ dürfen nicht an die Clients weitergegeben werden.

Die Delivery-Queue enthält alle Nachrichten, die derzeit an die Leserklienten versendet werden können. Sie hat eine feste Maximalgröße. Wird beim Speichern einer Nachricht in der DLQ die Maximalgröße überschritten, wird die älteste Nachricht verworfen.

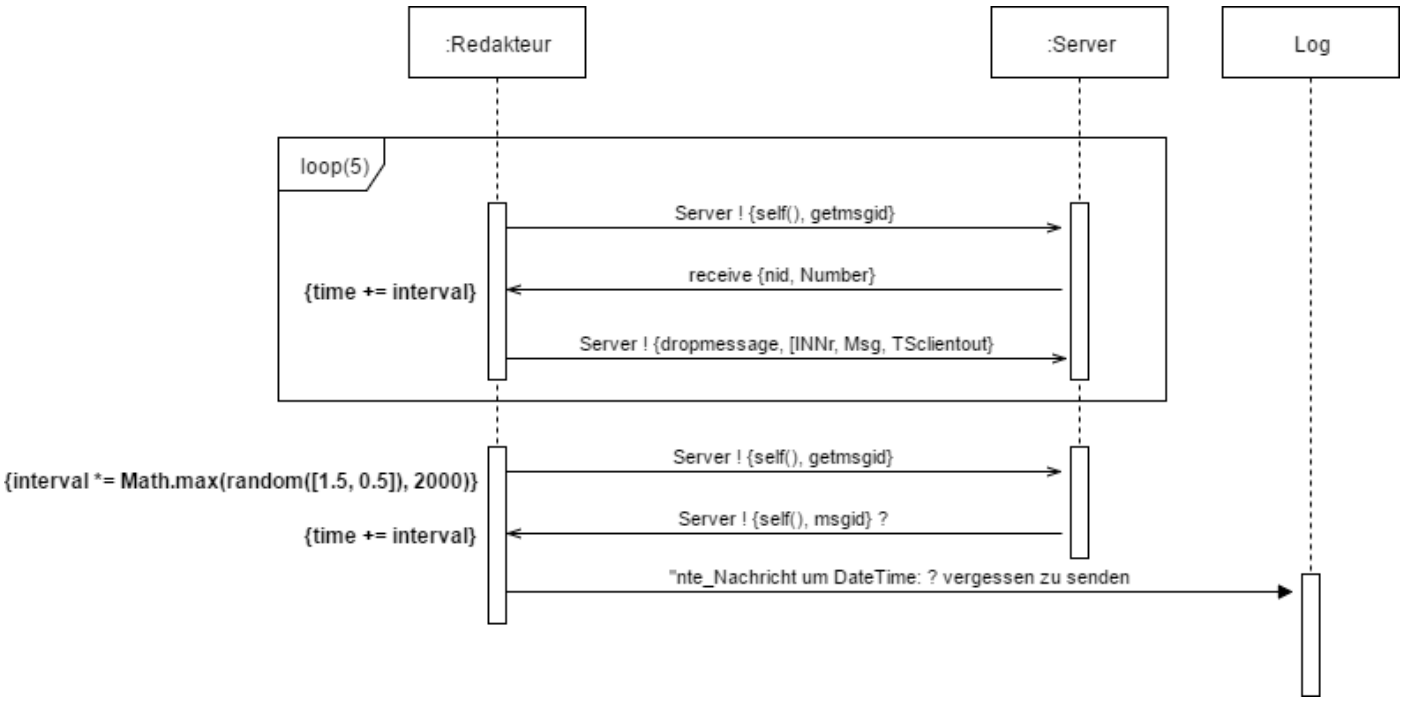
Die DLQ kann nicht direkt, sondern nur über die HBQ angesprochen werden.



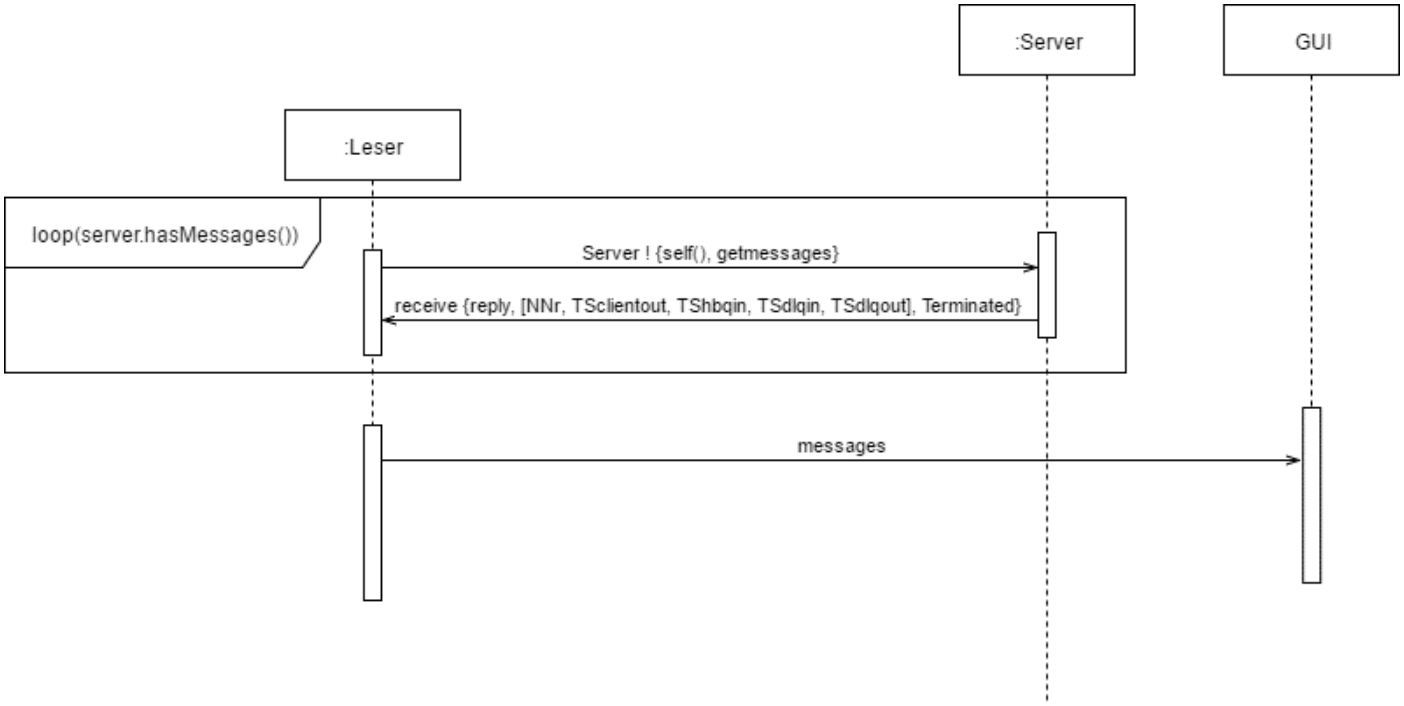

## Erläuterung des Ablaufs (Client)

Der Client hat zwei Rollen, zwischen denen er wechselt und agiert in einem einzelnen eigenständigen Prozess.

Die erste Rolle ist der Redakteur. Als solcher schickt er dem Server eine Anfrage bzgl. einer Nachrichtennummer. Nun wartet er auf diese und zählt von hieraus die Zeit hoch. Sobald er die Nummer erhalten hat, wartet er auf den Ablauf des Sendeintervalls und schickt dann die Nachricht, die Nummer und die aktuelle Systemzeit (/Sendezeit) an den Server. Diese Schleife aus Anfrage, Warten und Senden wird 5 Mal wiederholt. Danach wird das Intervall entweder um 50% erhöht, oder verkleinert (zufällig), wobei das Mindestintervall 2 Sekunden ist. Außerdem wird nun auch wieder wie zuvor nach der Nachrichtennummer gefragt, ohne dann eine Nachricht abzuschicken. Dies wird entsprechend in der Log-Datei vermerkt. Danach wechselt der Klient in die Rolle des Lesers.




Als Leser wird der Server solange nach Nachrichten gefragt, bis er keine hat. Diese merkt sich der Client und fügt den Nachrichten, die durch seinen Redakteur erstellt wurden „*******“ an. Außerdem vergleicht er die in- und out-Zeiten und vermerkt die Zeitdifferenz in dem Text.




Der Client terminiert, nach einer bestimmten (client.cfg) Zeit.

# Schnittstellen (Server)

*getmessages:*

**Format**: {**SenderPID**, getmessages}

**SenderPID**: PID des Senders

**Funktion**: Sendet die nächste Nachricht an den Klienten.

Der Server schaut nach, welche Nachricht zuletzt an den Client gesendet wurde und sendet dann die Nachricht mit der nächsthöheren Nummer an den Client zurück.

Ist ein Client dem Server noch nicht bekannt, wird die älteste, verfügbare Nachricht versandt.

Sind keine neuen Nachrichten verfügbar, wird eine Dummy-Nachricht mit **Terminated** = true an den Client gesendet.

**Die Antwort hat folgendes Format**:

{reply, **MSG_LIST**, **Terminated**}

**MSG_List** = [NNr, Msg, TSclientout, TShbqin, TSdlqin, TSdlqout]

**NNr**: Eindeutige Nummer

**Msg**: Text der Nachricht

**TSclientout**: Zeitstempel (3er Tupel aus erlang:now()) - Sendezeitpunkt vom Redakteur.

**TShbqin**: Zeitstempel (3er Tupel aus erlang:now()) - Ankunftszeitpunkt in der holdbackqueue

**TSdlqin**: Zeitstempel (3er Tupel aus erlang:now()) - Ankunftszeitpunkt in der Deliveryqueue

**TSdlqout**: Zeitstempel (3er Tupel aus erlang:now()) - Sendezeitpunkt von der Deliveryqueue

**Terminated**: Ein boolean Flag. True, wenn das die letzte Nachricht war und sonst False.

Im Falle von True wird die Abfrage erstmal beendet.

*dropmessage*

**Format**: {**SenderPID**, dropmessage, [**NNr**, **Msg**, **TSclientout**]}

**SenderPID**: PID des Senders

**NNr**: Nummer der Nachricht

**Msg**: Text der Nachricht

**TSclientout**: Zeitstempel bei Versand der Nachricht durch den Client

**Funktion**: Hinterlegt eine Nachricht auf dem Server.

Die Nachricht wird zur Speicherung an die Nachrichtenverwaltung delegiert. Details: Siehe Nachrichtenverwaltung.

**Antwort des Servers**:

**Format**: {reply, ok}

*getmsgid*

**Format**: {**SenderPID**, getmsgid}

**SenderPID**: PID des Senders

**Funktion**: Sendet die nächstverfügbare eindeutige Nachrichtennummer an den Klienten.

**Antwort des Servers**:

**Format**: {nid, **NNr**}

**NNr**: Nachrichten-ID.

**Andere:**

Alle Nachrichten die keinem der oben genannten Formate entsprechen, werden gesondert vermerkt.

## Schnittstellen (CMEM)

*initCMEM(RemTime, Datei)* → **CMEM**

**RemTime**: Zeit in Sekunden, nach der ein Client vergessen wird.

**Datei**: Datei, die fürs Logging genutzt werden kann.

**CMEM**: Ein leerer CMEM.

**Funktion**: Initialisiert den Klientenspeicher und gibt einen leeren CMEM zurück.

*delCMEM(CMEM)* → **ok**

**CMEM**: Der zu löschende CMEM.

**Funktion**: Löscht den CMEM und gibt ok zurück.

*updateClient(CMEM,ClientID,NNr,Datei)* → **NeuerCMEM**

**CMEM**: Der aktuelle CMEM.

**ClientID**: ID des Klienten, der aktualisiert werden soll.

**NNr**: Die Nummer der zuletzt an den Klienten gesendeten Nachricht.

**Datei**: Datei, die fürs Logging genutzt werden kann.

**NeuerCMEM**: der aktualisierte CMEM.

**Funktion**: Speichert bzw. aktualisiert den Eintrag des Klienten ClientID mit der neuen Nachrichtennummer.

*getClientNNr(CMEM,ClientID)* → **NNr**

**CMEM**: der aktuelle CMEM

**ClientID**: ID des angefragten Klienten.

**NNr**: Nächste Nummer, die der Client erhalten darf.

**Funktion**: Gibt die nächsterwartete Nachrichtennummer für diesen Klienten zurück.

## Schnittstellen (Holdback-Queue/HBQ)

*initHBQ*

**Format**: {**SenderPID**, {request, initHBQ}}

**SenderPID**: PID des Senders

**Funktion**: Initialisiert die HBQ und DLQ. Nach Initialisierung ist die HBQ leer.

**Antwort der HBQ**:

**Format**: {reply, ok}

*pushHBQ*

**Format**: {**SenderPID**, {request, pushHBQ, [**NNr**, **Msg**, **TSclientout**]}

**SenderPID**: PID des Senders

**NNr**: Nummer der erhaltenen Nachricht.

**Msg**: Inhalt der erhaltenen Nachricht.

**TSclientout**: Zeitstempel bei Versand der Nachricht durch den Client.

**Funktion**: Fügt der Nachricht einen Zeitstempel bei Eintragung in die HBQ an und hinterlegt diese Nachricht in der HBQ.

**Antwort der HBQ:**

**Format**: {reply, ok}

*DeliverMSG*
**Format**: {**SenderPID**, {request, deliverMSG, **NNr**, **ToClient**}}

**SenderPID**: PID des Senders

**NNr**: Nummer der zu sendenden Nachricht

**ToClient**: PID des Clients, an den die Nachricht gesendet werden soll

**Funktion**: Sendet die Nachricht mit der Nummer NNr an den Client ToClient. Ist die Nachricht mit der Nummer NNr nicht mehr in der DLQ hinterlegt, wird die älteste Nachricht versendet.

**Antwort der HBQ:**

**Format**: {reply, **SendNNr**}

**SendNNr**: Nummer der versendeten Nachricht

*dellHBQ*
**Format**: {**SenderPID**, {request, dellHBQ}}

**Funktion**: Terminiert den Prozess der HBQ.

**Antwort der HBQ:**

**Format**: {reply, ok}

## Schnittstellen (Delivery-Queue/DLQ)

*initDLQ(Size, Datei)* → **DLQ**

**Size**: Größe der zu erstellenden DLQ.

**Datei**: Datei, die fürs Logging genutzt werden kann.

**DLQ**: eine leere DLQ.

**Funktion**: Erstellt eine leere DLQ mit fester Maximalgröße und gibt diese zurück.

*delDLQ(Queue)* → **ok**

**Queue**: die zu löschende DLQ

**Funktion**: Löscht die DLQ und gibt ok zurück.

*expectedNr(Queue)* → **NNr**

**Queue**: die aktuelle DLQ

**NNr**: die als nächstes erwartete Nachrichtennummer.

**Funktion:** Gibt die als nächstes erwartete Nachrichtennummer zurück.

*push2DLQ([NNr, Msg, TSclientout, TShbqin], Queue, Datei)* → **NeueDLQ**

**NNr**: Nummer der zu speichernden Nachricht

**Msg**: Inhalt der zu speichernden Nachricht

**TSclientout**: Zeitstempel bei Versand der Nachricht durch den Client

**TShbqin**: Zeitstempel bei Speichern der Nachricht in der HBQ

**Queue**: aktuelle DLQ

**Datei**: Datei, die für Logging verwendet werden kann.

**NeueDLQ**: die modifizierte DLQ inklusive der zu speichernden Nachricht

**Funktion**: Speichert die Nachricht in die DLQ und hängt einen Zeitstempel für den Eingang der Nachricht in die DLQ an.

*deliverMSG(MSGNr, ClientPID, Queue, Datei)* → **SentNNr**

**MSGNr**: Nummer der zu versendenden Nachricht

**ClientPID**: PID des Clients, an den die Nachricht versendet werden soll

**Queue**: aktuelle DLQ

**Datei**: Datei, die für Logging verwendet werden kann

**SentNNr**: Nummer der tatsächlich versendeten Nachricht.

**Funktion**: Sendet die Nachricht mit der Nummer MSGNr an den Client mit der PID ClientPID. Ist diese nicht verfügbar, wird die älteste Nachricht der DLQ versendet. Außerdem wird der Nachricht der Zeitstempel der Versandzeit mitgegeben.

**Format der Nachricht an den Client:**

siehe Server::getmessages

## Sonstiges zur Nachrichtenverwaltung und Server

**server.cfg**
{**latency**, 42}. -> Die maximale Inaktivitätszeit des Servers in Sekunden.
{**clientlifetime**,4}. -> Wie lange sich der Server einen Klienten merkt, bis er ihn vergisst.
{**servername**, wk}. -> Name des Servers.
{**hbqname**, hbq}. -> Name der Holdbackqueue.
{**hbqnode**, 'hbqNode@KI-VS'}. -> Name der Node.
{**dlqlimit**, 42}. -> Die maximale Anzahl der Nachrichten in der DLQ.

Die Ausgabedatei für die Nachrichtenverwaltung wird „HB-DLQ@<Node>.log“ heißen.

Die Ausgabedatei für den Server wird „Server@<Node>.log“ heißen.

## Schnittstellen (Leser)

*ServerId ! {self(), getmessages}*
**Funktion**: Sendet eine Anfrage an den Server.

**Antwort des Servers:**

**Format**: {reply, **MSG_List**, **Terminated**}

**MSG_List** = [NNr, Msg, TSclientout, TShbqin, TSdlqin, TSdlqout]

**NNr**: Eindeutige Nummer

**Msg**: Aktuelle Textzeile

**TSclientout**: Zeitstempel (3er Tupel aus erlang:now()) - Sendezeitpunkt vom Redakteur.

**TShbqin**: Zeitstempel (3er Tupel aus erlang:now()) - Ankunftszeitpunkt in der holdbackqueue

**TSdlqin**: Zeitstempel (3er Tupel aus erlang:now()) - Ankunftszeitpunkt in der Deliveryqueue

**TSdlqout**: Zeitstempel (3er Tupel aus erlang:now()) - Sendezeitpunkt von der Deliveryqueue

**Terminated**: Ein boolean Flag. True, wenn das die letzte Nachricht war und sonst False.

Im Falle von True wird die Abfrage erstmal beendet.

Erwartet eine aktuelle Textzeile vom Server und stellt diese in der GUI dar. Der bisherigen Nachricht wird die aktuelle Systemzeit angehängt und in der log ausgegeben. Es wird vorher geprüft, ob die Nachricht aus der Zukunft kommt und die Differenz wird entsprechend der Zeichenkette angefügt.

## Schnittstellen (Redakteur)

*Server ! {self(), getmsgid}*
Fragt den Server nach der eindeutigen Nachrichtennummer.

**Antwort des Servers:**

**Format**: {nid, **Number**}

**Number**: Die eindeutige Nachrichtennummer.

*Server ! {dropmessage, [INNr, Msg, TSclientout]}*
**INNr**: Eindeutige Nachrichtennummer.

**Msg**: Nachricht, siehe Format.

**TSclientout**: Sendezeit (3er Tupel aus erlang:now()) der Nachricht.

**Format der Ausgabe**: <ClientNummer>-<NodeName>-<PID>-KLC: <n>te_Nachricht. C Out: <Zeitstempel>| gesendet

## Sonstiges zum Client

**client.cfg**
{**clients**, 9}. -> Anzahl der Klienten
{**lifetime**, 42}. -> Lebenszeit eines Klienten in Sekunden
{**servername**, wk}. -> Servername
{**servernode**, 'server@KI-VS'}. -> Servernode
{**sendeintervall**, 3}. -> Anfangsintervall, in dem gesendet wird

Die Ausgabedatei wird „client <Nummer><Node>.log“ heißen.