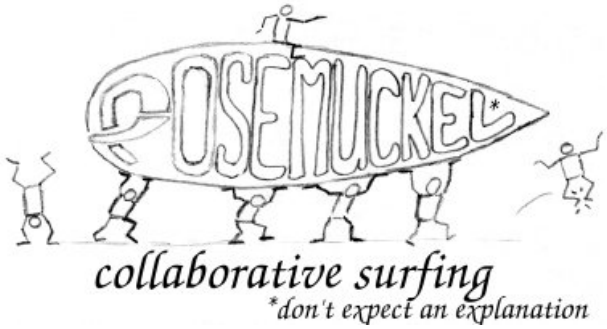

Posemuckel DevRef
(Developer's Reference)

0. Einleitung (Informatiker fangen bei Null an zu zählen.)

In diesem Text wird die Struktur vom Posemuckel Client und Server beschrieben. Beide nutzen teilweise gemeinsame Programmteile, die sich aber ausschließlich in `posemuckel.common` befinden. Bei diesen gemeinsamen Teilen handelt es sich hauptsächlich um einfache Stringverarbeitung, Konfigurationsmodule sowie Klassen zum Parsen und Formatieren von Netzwerknachrichten. Hier wird, nach einem Überblick über Client und Server, der Lauf der `ADD_BUDDY` Nachricht und deren Antwort durch die Anwendung verfolgt, um die wesentlichen Programmteile am Beispiel zu erläutern.
Client und Server kommunizieren über ein textbasiertes Protokoll, welches etwas an HTTP erinnert. Es werden Methoden spezifiziert, die durch Schlüsselwörter gekennzeichnet sind. Neben einer HTTP-ähnlichen Anfrage-Antwort-Semantik gibt es allerdings auch Nachrichten, die ohne Anfrage vom Server an einen oder mehrere Clients gesandt werden. Das Protokoll ist im RFC0815 spezifiziert, welches in einer Textdatei im Verzeichnis doc der Quellen liegt.

1. Überblick zum Client

Die `main`-Methode des Client befindet sich in `posemuckel.client.net.Client`. Der Client verwendet eine Reihe von Singletons, um auf bestimmte Programmteile ständig zugreifen zu können. Dazu gehört eine Abstraktion der Verbindung zum Server `posemuckel.client.net.ClientConnection`, eine globale Konfiguration `posemuckel.common.Config` und eine Instanz von `posemuckel.client.net.ThreadLauncher`, die Referenzen auf alle Threads hält, um diese zentral beenden zu können. Das Datenmodell in Form der Klasse `posemuckel.client.model.Model` verhält sich in der Applikation wie ein Singleton.
Der Client macht intensiven Gebraucht von Threads. Ähnlich wie auch im Server gibt es einen Thread, der permanent vom Socket liest (`posemuckel.client.net.RecvMessage`) und einen, der für das Schreiben (`posemuckel.client.net.SendMessage`) auf den Socket zuständig ist. Darüber hinaus gibt es noch weitere Threads, die gestartet werden, um spezifische Aufgaben zu übernehmen, wie zum Beispiel für das Encoding und Deconding von Screenshots der Webseiten oder das verzögerte Senden der `READING`-Nachricht.

2. Datenmodell des Clients

In der Klasse `posemuckel.client.model.Model` werden einige, für den Client global wichtige

Daten im Arbeitsspeicher gehalten. Dazu gehört

- der lokale Benutzer (Instanz von `posemuckel.client.model.User`) mit einer Liste der offenen Einladungen (Instanz von `posemuckel.client.model.ProjectList`) und der Buddyliste (Instanz von `posemuckel.client.model.MemberList`)
- das aktuell offene Projekt des Benutzers (Instanz von `posemuckel.client.model.Project`) mit dessen Webtrace und dem Foldersystem
- eine Liste aller dem Client bekannten Projekte (Instanz von `posemuckel.client.model.ProjectList`),
- eine Liste aller dem Client bekannten Benutzer (Instanz von `posemuckel.client.model.UsersPool`),
- eine Liste aller Chats, die diesen Client betreffen (in einem Hash)

Das `Model` verhält sich in der Applikation wie ein Singleton: es gibt nur eine Instanz davon. In einer Instanz von `Model` gibt es genau einen `User` und höchstens ein offenes Projekt.
Um Daten aus der Datenbasis zu lesen oder die Daten zu ändern, kann mit Hilfe der `posemuckel.client.model.DatabaseFactory` von einer Klasse aus dem Paket `posemuckel.client.model` auf die aktuelle Instanz von `posemuckel.client.model.Database` zugegriffen werden.
Wenn die Datenbasis von außerhalb des Clients geändert wird (Beispiel: ein anderer Anwender erstellt ein neues Projekt), muss der Client über Änderungen informiert werden, um seine Daten aktualisieren zu können. Zu diesem Zweck registriert sich eine Instanz einer Klasse, die das Interface `posemuckel.client.model.InformationReceiver` implementiert, bei einer `Database`. Als `InformationReceiver` dient im Datenmodell eine Instanz von `posemuckel.client.model.Model`.
Hierdurch kann man die eigentliche Datenbasis, auf der der Client operiert, austauschen, was für Testzwecke relevant ist. Im Normalfall wird ein Objekt vom Typ `posemuckel.client.net.Netbase` zu dieser Registrierung verwendet. Deshalb wird hier diese Implementierung der `Database`-Schnittstelle beschrieben.

3. Geführter Rundgang durch den Client

Um sich einen Überblick zu verschaffen, macht es Sinn, sich parallel zur Lektüre dieses Textes die entsprechenden Teile des Quellcodes anzusehen.
Es gibt zwei ständig laufende Threads im Client, die lesend bzw. schreibend auf den Client-Socket zugreifen. Dabei handelt es sich um Objekte der Klassen `posemuckel.client.net.SendMessage` und `posemuckel.client.net.RecvMessage`.

3.1 Senden einer Nachricht

Das Senden einer Nachricht wird von einer GUI-Instanz bzw. vom Benutzer initialisiert. Es gibt eigentlich nur eine Ausnahme, die in dem Senden der verzögerten `READING` Nachricht besteht.
Damit die GUI beim Senden von Nachrichten nicht blockiert, wird dies als paralleler Thread durchgeführt, der von Anfang bis zum Ende des Programmes existiert.
Aus der GUI werden zum Senden Methoden des `Model` aufgerufen bzw. Methoden der Objekte `User`, `Person`, `Project` etc. Hier soll beispielhaft das Hinzufügen eines fremden Benutzers in die Buddy-Liste betrachtet werden.
Wenn der Benutzer über das grafische Interface diese Aktion durchführen will, wird die `run`-Methode von `posemuckel.client.gui.actions.AddBuddyAction` ausgeführt. Dort wird über das zentrale `Model`-Objekt auf den `User` und über diesen auf die eigene Buddy-Liste im Client zugegriffen. Diese ist ein Objekt vom Typ `posemuckel.client.model.MemberList`. Die Methode `addBuddy` von `MemberList` wird nun aufgerufen und dabei der Benutzername des Benutzers angegeben, der als Buddy hinzugefügt werden soll. (`addBuddy` funktioniert nur bei

Instanzen von `MemberList`, die vom Anwender editierbar sind)

3.1.1 Tasks

Die Tasks im Client sind ein wesentlicher Bestandteil der Kommunikation. Für alle Nachrichten an den Server, deren Antwort der gesendeten Nachricht zugeordnet werden müssen, ist ein Task erforderlich. Die Mutter aller Tasks ist die Klasse `posemuckel.client.model.Task`. Dabei handelt es sich um eine abstrakte Basisklasse, deren grundlegende Funktionalität in den entsprechend abgeleiteten Klassen implementiert wird. In Task ist aber die Methode `execute` implementiert, welche die Methode `work` der entsprechenden Objekte aufruft, um die eigentliche Arbeit zu erledigen. Die Methoden `work` und `update` sind hier von besonderer Bedeutung.
Beim Senden der `ADD_BUDDY`-Nachricht wird also die Methode `work` des `posemuckel.client.model.BuddyTask` aufgerufen. Dabei wird eine ID übergeben, die anzeigt, dass es sich hier um das Hinzufügen eines Buddy handelt. Diese ID wird im Task gespeichert, so dass in diesem Task bekannt ist, welche Nachricht gesendet wurde. Außerdem speichert die Task alle Daten (wie den Namen des zukünftigen Buddys), die zur Verarbeitung der Antwort der `Database` nötig sind. Die Task merkt sich also die Anfrage, die an die `Database` gestellt wird. Dadurch können die Informationen, die in der Antwort enthalten sein müssen, auf ein Minimum beschränkt werden. Danach wird eine von der Nachricht abhängige Methode von `Netbase` aufgerufen. Beim Hinzufügen eines Buddy ist dies `addBuddy`.

3.1.2 Netbase

In der eigentlichen Datenbasis des Clients wird nun zunächst einmal eine Referenz auf den Task zusammen mit einer eindeutigen Nachrichten-ID gespeichert. So können die Tasks beim Empfang der Antwortnachricht mit gleicher ID wiedergefunden werden und es kann zum Beispiel gleichzeitig mehrere offene `BuddyTask` geben. Die Nachrichten-ID wird zusammen mit allen weiteren für die Nachricht nötigen Parametern einer Methode von `posemuckel.client.net.ClientMessage` übergeben.

3.1.3 ClientMessage

In der beim Hinzufügen eines Buddies aufgerufenen Methode `addBuddy` von ClientMessage wird nun die Formatierung der Nachricht vorgenommen, wobei eine Hilfsklasse aus `posemuckel.common` zum Einsatz kommt, so dass die Trennzeichen der Nachrichtenbestandteile zentral für Client und Server geändert werden können. Ist die Nachricht fertig für das Senden, dann wird sie in der Methode `add2sendqueue` in die Sendewarteschlange eingefügt. Diese Warteschlange ist nichts anderes als ein Vector von Strings, da es sich um textbasiertes Protokoll handelt. Die Warteschlange dient hier als Schnittstelle für den Datenaustausch zwischen Threads, weshalb der möglicherweise wartende Sende-Thread erstmal mit einem `notify` aufgeweckt werden muss.

3.1.4 SendMessage

In der `run` Methode von `posemuckel.client.net.SendMessage` findet dann der konkurrierende Zugriff auf die Warteschlange statt. Falls der Vector nicht leer ist, wird das erste Element entfernt und mit der Methode `sendToServer` in den Socket (für eine TCP-Verbindung) geschrieben.
Sind diese Schritte ausgeführt, wandert die Nachricht zum Server und wird dort verarbeitet. Speziell im Falle der `ADD_BUDDY` Nachricht erwartet der Client nach RFC0815 als Antwort eine `NEW_BUDDY`, `ACCESS_DENIED` oder eine `ERROR` Nachricht.
Bis zu diesem Zeitpunkt wurden die Daten im Datenmodel des Clients nicht geändert. Die Buddyliste enthält demnach noch nicht den neuen Buddy. Durch diese Strategie werden keine undo-Operationen benötigt.

3.2 Empfangen einer Nachricht

Beim Empfang von Nachrichten fängt alles in der Klasse `posemuckel.client.net.RecvMessage` an. Eine Instanz hiervon läuft in einem eigenen Thread und liest in einer Schleife ständig vom Socket, solange dieser offen ist. Der Socket wird beim Login des Anwenders (bzw. der Registrierung) geöffnet und erst beim Schließen der Applikation geschlossen, um dem Server zu ermöglichen, Nachrichten von anderen Anwendern (wie zum Beispiel Chatnachrichten) an den Client weiterzuleiten.

3.2.1 ServerMessage

Die gelesenen Nachrichten werden dem Message-Handler des Clients übergeben. Die Klasse `posemuckel.client.net.ServerMessage` des Clients ist von dem allgemeinen Nachrichten-Parser `posemuckel.common.MessageHandler` abgeleitet und dessen Methode `eat_up_ServerPacket` ruft die nachrichtenspezifischen Klassenmethoden auf. Hierzu werden erstmal alle Datenfelder der Nachrichten bis zum Methoden- bzw. Nachrichtennamen eingelesen. Dann wird im Falle der `NEW_BUDDY` Nachricht des Servers die Methode `newBuddy` von `ServerMessage` aufgerufen.
In den jeweiligen nachrichtenspezifischen Methoden des Message-Handlers werden wichtige Aufgaben erledigt. Dazu gehört zunächst das Einlesen der restlichen, nachrichtenspezifischen Parameter und dann natürlich deren Prüfung hinsichtlich der RFC-Konformität. Ist eine Nachricht korrupt, muss dies durch eine `posemuckel.common.InvalidMessageException` angezeigt werden.In der Regel wird die Nachrichten-ID einer Anfrage vom Server unverändert zurückgeschickt.
Bei der `NEW_BUDDY` Nachricht wird noch der Online-Status des hinzugefügten Benutzers gesendet. Also wird dieser eingelesen. Den Namen des Buddys hat der Server nicht gesendet, da sich die Task diesen Namen gemerkt hat. Um diese gelesenen Informationen nun weiterzureichen, wird die Methode `update` von `Netbase` aufgerufen, wobei Nachrichten-ID und Parameter der Nachricht übergeben werden.

3.2.2 Schon wieder Netbase

`Netbase` kann nun aus seinen internen Datenhalden den passenden Task zur Nachrichten-ID rauskramen. Das passiert letztlich in `removeTask` von `Netbase`.Von dem gefundenen `Task`-Objekt wird eine Methode `update` aufgerufen.

3.2.3 Schon wieder BuddyTask

In der Methode `update` des Tasks wird nun in Abhängigkeit von gesendeter und empfangener Nachricht eine Aktion ausgeführt. Dabei gibt es mehrere solcher Methoden und üblicherweise enthält diejenige mit dem Integer-Argument eine switch-Anweisung mit zwei Argumenten, die IDs für gesendete und empfangene Nachrichten sind. Im Falle einer `NEW_BUDDY` Nachricht vom Server wird jedoch die Variante mit dem String-Parameter aufgerufen. Die Task konstruiert aus dem lokal gespeicherten Namen und dem übergebenen Onlinestatus ein Instanz von `Person` und ruft die Methode `confirmAddMember` der Buddyliste (die eine Instanz von `MemberList` ist) auf. Dabei wird die eben konstruierte `Person` als Parameter übergeben.
In der aufgerufenen Methode von `MemberList` wird nun die `Person` zu den Buddys hinzugefügt.

3.2.4 Events

In der entsprechenden `MemberList`-Methode wird nun die GUI über das asynchrone Ereignis der `NEW_BUDDY` Nachricht informiert, indem dort ein Event abgefeuert wird. Im Client ist ein eigenes Event-Management implementiert, welches sich unter `posemuckel.client.model.event.*` finden lässt.Im speziellen Fall von `MemberList` werden in der GUI Listener vom Typ `MemberListListerner` bzw. `MemeberListAdapter` implementiert. Diese werden in der GUI dann über die Methode `addListener` der `MemberList` registriert.

Bei der NEW_BUDDY Nachricht passiert nun folgendes: Verschiedene Teile der GUI implementieren den passenden Listener. So zum Beispiel der posemuckel.client.gui.MemberListProvider. Beim Aufruf der Methode confirmAddMembervon MemberList werden jeweils deren Methoden memberAdded aufgerufen und dabei ein Objekt vom Typ MemberListEvent übergeben. Der entsprechende Code in der GUI kann nun also den neuen Buddy mit seinem korrekten Online-Status in einer passenden Tabelle anzeigen.

4. Überblick zum Server

Die main-Methode des Servers befindet sich in der Klasse posemuckel.server.Server. Hier wird der Server mit der Konfiguration geladen, die von der Klasse posemuckel.common.Config gestellt wird und wartet anschliessend über ein Objekt der Klasse ServerSocket auf Verbindungsanfragen von Clients.

Für jeden Client, der sich mit dem Server verbindet, wird eine Instanz der Klasse posemuckel.server.ServerProcess erzeugt. Diese Instanz ist ein Thread, der laufend auf Anfragen seines Clients wartet und diese von einem Objekt der Klasse posemuckel.server.ClientMessage verarbeiten lässt.

Das Objekt der Klasse posemuckel.server.ClientMessage ist von der Klasse posemuckel.common.MessageHandler abgeleitet und ruft seine zur Client-Anfrage passende Methode auf. In dieser Methode werden die nötigen Datenmodell-Zugriffe (auf Objekte der Klassen posemuckel.server.Model und posemuckel.server.DB) durchgeführt und die geeignete Antwortnachricht bestimmt.

Für die Formatierung der Antwortnachrichten ist ein Objekt der Klasse posemuckel.server.ServerMessage zuständig. Die Nachrichtenübertragung an den oder die Clients wird von einem Thread (Objekt der Klasse posemuckel.server.SendMessage) übernommen, der laufend Nachrichten aus einer Warteschlange holt und diese verschickt.

Damit Einladungs-Mails verschickt werden können, wird vom Server ein Objekt der Klasse posemuckel.server.SendMail benutzt. Dieses verwendet einen lokalen MTA für den e-Mail-Versand.

5. Datenmodell des Servers

5.1 MYSQL-Datenbank

Für die persistente Datenhaltung wird eine MYSQL-Datenbank eingesetzt. Auf die Daten der Datenbank wird über ein Objekt der Klasse posemuckel.server.DB zugegriffen.

Folgende Tabellen sind in der MYSQL-Datenbank enthalten:

- user: Enthält Informationen über den Benutzernamen (nickname), Nachnamen (LastName), Vornamen (firstName), e-Mail-Adresse (email), Benutzerkommentar (user_comment), Passwort (password), IP-Adresse des Benutzerclients (user_ip), Sprache des Benutzers (lang), Geschlecht (gender), Wohnort (location), Benutzerhash (hash) und Status (logged_in)
- ratings: Enthält Informationen über das Projekt, in dem bewertet wird (project_id), den Namen des Bewerters (user_nickname), die Bewertung einer Website (rating), Kommentar zur Bewertung (rating_notes), die bewertete URL (url_id) und den Zeitpunkt der Bewertung (rating_timestamp)
- chat_progress: Enthält Informationen über die Identität des Chats (chat_id), den eingegebenen Text (phrase), den Zeitpunkt der Erstellung (progress_timestamp) und den

Namen des Verfassers (user_nickname)
- chat: Enthält Informationen, ob der Chat privat ist (private_chat), wer den Chat eröffnet hat (chat_owner) und von wem der Chat geschlossen wurde (chat_closed_by)
- projects: Enthält Informationen über den Projektchat (project_chat), den Titel des Projektes (project_title), die Projektbeschreibung (project_description), die Teilnehmerzahl (count_members), die maximale Teilnehmerzahl (max_members), den Eigentümer des Projektes (project_owner), den Projekttyp (project_type) und das Erstellungsdatum (project_date)
- project_invitedusers: Enthält Informationen zu welchem Projekt (project_id) welcher Benutzer (invited_user) eingeladen worden ist. Ausserdem wird angegeben, ob die Einladung schon bestätigt (invitation_confirm) und beantwortet (invitation_answered) worden ist
- members: Enthält Informationen welcher Benutzer (user_nickname) an welchem Projekt (project_id) teilnimmt
- buddies: Enthält Informationen welcher Benutzer (user_nickname) welche Buddies (buddy_nickname) hat
- url: Enthält Informationen über die Adresse (address) und den Titel (title) einer URL
- folders: Enthält Informationen über das Projekt des Ordners (project_id), den Namen des Ordners (name), den Elternordner (parent_folder) und, ob das der Ordner für unsortierte URLs ist (unsorted_folder)
- folder_urls: Enthält Informationen welcher Ordner (folder_id) welche URLs (url_id) enthält
- user_urls: Enthält Informationen welcher Benutzer (user_nickname) welche URL (url_id) in welchem Projekt (project_id) zu was für einem Zeitpunkt (url_timestamp) bewertet hat und welche URL vorher besucht wurde (referred_by_url)
- user_chat: Enthält Informationen an was für einem Chat (chat_id) welche Benutzer (user_nickname) teilnehmen

5.2 Die Klasse posemuckel.server.Model

Das Model ist ein Singelton, das wie ein Cache alle relevanten Daten enthält, auf die häufig zugegriffen werden muss; dadurch wird die Datenbank entlastet.
Dazu besitzt es Datenstrukturen, in denen die Clientdaten (Objekte der Klasse posemuckel.server.ClientInfo), die Projektdaten (Objekte der Klasse posemuckel.server.ProjectInfo) und die Chatdaten (Objekte der Klasse posemuckel.server.ChatInfo) gehalten werden und bietet Methoden an, um auf diese Daten zuzugreifen.

6. Geführter Rundgang durch den Server

Es wird nun davon ausgegangen, dass in einem Client eine ADD_BUDDY-Nachricht verschickt wurde, und diese im Server ankommt.

6.1 Empfangen der Nachricht

Wie oben angedeutet wird jedem Client bei seinem jeweils ersten Verbindungsversuch mit dem Server ein Objekt der Klasse posemuckel.server.ServerProcess zugeordnet, das in seiner run-Methode auf Nachrichten des Clients wartet.
Der dem Client zugeordnete ServerProcess fängt die ADD_BUDDY-Nachricht über die Methode eat_up_ClientPacket(BufferedReader in) ab. Diese Methode entschlüsselt die Nachricht nach dem RFC0815-Protokoll und ruft die dazu passende Methode addBuddy(String hash, String id, String count, BufferedReader in) der Klasse posemuckel.server.ClientMessage auf.

6.2 Verarbeitung der Nachricht

In der Methode `addBuddy(String hash, String id, String count, BufferedReader in)` wird die `ADD_BUDDY`-Nachricht verarbeitet. Dazu werden zunächst die notwendigen Zugriffe auf das Datenmodell des Servers durchgeführt. In unserem Fall wird die Methode `addBuddy(String user, String buddy)` der Klasse `posemuckel.server.DB` aufgerufen, um die Eintragungen in der MYSQL-Datenbank vorzunehmen.
Danach wird die Formatierung und das Senden der Antwortnachricht veranlasst.

6.3. Formatierung der Antwortnachricht

Objekte der Klasse posemuckel.server.ServerMessage sind für das Formatieren der Antwortnachrichten gemäß dem RFC0815-Protokoll zuständig. Von der Methode `addBuddy (String hash, String id, String count, BufferedReader in)` aus, wird die Methode `generic(Vector<ClientInfo> recievers, String id, String msgname, String[] data)` des `ServerMessage`-Objektes aufgerufen. Diese erzeugt in unserem Fall eine `NEW_BUDDY`-Nachricht, die in die Sende-Warteschlange des Servers eingefügt wird.

6.4. Senden der Antwortnachricht

Im Server läuft ständig ein Sendethread (Objekt der Klasse `posemuckel.server.SendMessage`), der nacheinander alle Nachrichten, die in der Sende-Warteschlange landen, herausnimmt und an die angegebenen Empfänger verteilt.