



Software-Challenge 2017

XML-Schnittstellendokumentation

Mississippi Queen

Stand: 9. September 2016

Inhaltsverzeichnis

1. Einleitung	1
1.1. Beispiel-Definition	2
I. Client → Server	3
2. Spiel betreten	3
2.1. Ohne Reservierung	3
2.2. Mit Reservierungscode	3
3. Zge senden	3
3.1. Der Move	3
3.2. Zug	3
3.3. ACTION	4
3.4. Debughints	4
II. Server → Client	5
4. Raum beigetreten	5

5. Willkommensnachricht	5
6. Spielstatus	5
6.1. memento	5
6.2. Status	5
6.3. Spieler	6
6.4. Spielbrett	7
6.5. Spielfeld	7
6.6. Letzter Zug	7
7. Zug-Anforderung	7
8. Fehler	8
9. Spiel pausiert	8
10. Spiel verlassen	9
11. Spielergebnis	9
 III. berblick	 10

1. Einleitung

Zum Inhalt

Wie in den letzten Jahren wird zur Client-Server Kommunikation ein XML-Protokoll genutzt¹. In diesem Dokument wird die Kommunikationsschnittstelle definiert, sodass ein komplett eigener Client geschrieben werden kann. Es wird hier nicht die vollständige Kommunikation dokumentiert bzw. definiert, dennoch alles, womit ein Client umgehen können muss, um spielfähig zu sein.

An wen richtet sich dieses Dokument?

Die Kommunikation mit dem Spielserver ist für diejenigen, die aufbauend auf dem Simpleclient programmieren, unwichtig. Dort steht bereits ein funktionierender Client bereit und es muss nur die Spiellogik entworfen werden.

Nur wer einen komplett eigenen Client entwerfen will, beispielsweise um die Programmiersprache frei wählen zu können, benötigt die Definitionen.

Hinweise

Falls Sie beabsichtigen sollten, diese Kommunikationsschnittstelle zu realisieren, sei darauf hingewiesen, dass es im Verlauf des Wettbewerbes möglich ist, dass weitere, hier noch nicht aufgeführte Elemente zur Kommunikationsschnittstelle hinzugefügt werden. Um auch bei solchen Änderungen sicher zu sein, dass ihr Client fehlerfrei mit dem Server kommunizieren kann, empfehlen wir Ihnen, beim Auslesen des XML jegliche Daten zu verwerfen, die hier nicht weiter definiert sind.

Die vom Institut bereitgestellten Programme (Server, Simpleclient) nutzen eine Bibliothek um Java-Objekte direkt in XML zu konvertieren und umgekehrt. Dabei werden XML-Nachrichten nicht mit einem newline abgeschlossen.

Wie das Dokument zu lesen ist

Es finden sich für einzelne Arten von Nachrichten kurze Definitionen. Dabei ist ein allgemeiner XML-Code gegeben, bei dem Attributwerte durch Variablen ersetzt sind, die bei dem Code kurz erläutert werden.

Eingebettete XML-Elemente werden hier allgemein als

`{TAGNAME}`

geschrieben, wenn an der Stelle eine beliebige Anzahl von *TAGNAME*-Tags eingebettet sein *kann* und als

`[TAGNAME]`

¹siehe auch: [Allgemeine Dokumentation: Die Schnittstelle zum Server](#)

, wenn an der Stelle ein *TAGNAME*-Tag optional stehen kann.

1.1. Beispiel-Definition

Die Definitionen von a- und b-Tags unten lassen unter anderem folgende a-Tags zu:

```
- <a name="Sinnvoll">
  </a>
- <a name="Sinnvoll">
  <b anzahl="2"/>
</a>
- <a name="Hans">
  <b anzahl="2"/>
  <b anzahl="2"/>
  <b anzahl="2"/>
</a>
```

Definition 'b'

```
<b anzahl="2">
```

Definition 'a'

N Ein beliebiger Name

b ein b-Tag wie in obiger Definition

```
<a name="N" >
{b}
<\a>
```

Teil I.

Client → Server

2. Spiel betreten

2.1. Ohne Reservierung

Betritt ein beliebiges offenes Spiel:

```
<join gameType="swc_2017_mississippi_queen"/>
```

2.2. Mit Reservierungscode

Ist ein Reservierungscode gegeben, so kann man den durch den Code gegebenen Platz betreten.

RC Reservierungscode

```
<joinPrepared reservationCode="RC"/>
```

3. Zge senden

3.1. Der Move

Der Move ist der allgemeine Zug, der in verschiedenen Varianten gesendet werden kann.

RID ID des Raumes

ZUG Informationen ber den Zug

```
<room roomId="RID">  
  ZUG  
</room>
```

3.2. Zug

ACTION eine Teilaktion des Zuges

```
<data class="move">  
  <actions>  
    ACTION  
    ..  
  </actions>  
</data>
```

```

    ACTION
  </actions>
</move>

```

3.3. ACTION

I Nummer der Aktion

M Beschleunigung $M \in \{-5, -4..5\}$

N Anzahl der Drehungen $N \in \{-3..3\}$ positiv für gegen, negativ für mit dem Uhrzeigersinn

V Anzahl der Felder die in Bewegungsrichtung zurückgelegt werden (-1 für Rückwärts)

D Richtung in die abgedrängt wird. $D \in \{RIGHT, UP_RIGHT, UP_LEFT, LEFT, DOWN_LEFT, DOWN_RIGHT\}$

```

<acceleration order="I" acc="M"/>
<turn order="I" direction="N"/>
<step order="I" distance="V"/>
<push order="I" direction="D"/>

```

3.4. Debughints

Zügen können Debug-Informationen beigegeben werden:

S Informationen zum Zug als String

```

<hint content="S"/>

```

Damit sieht beispielsweise ein Laufzug so aus:

```

<room roomId="RID">
  <data class="move">
    <actions>
      <acceleration order="0" acc="1"/>
      <step order="1" distance="3"/>
      <turn order="2" direction="-1"/>
      <step order="3" distance="1"/>
      <push order="4" direction="LEFT"/>
    </actions>
    <hint content="Dies ist ein Hint."/>
    <hint content="noch ein Hint"/>
  </data>
</room>

```

Teil II.

Server → Client

RID ID des Raumes, in dem das Spiel stattfindet

4. Raum beigetreten

```
<joined roomId="RID"/>
```

5. Willkommensnachricht

Der Spieler erhält zu Spielbeginn eine Willkommensnachricht, die ihm seine Farbe mitteilt.

C Spielerfarbe (red/blue)

```
<room roomId="RID">
  <data class="welcomeMessage" color="C"/>
</room>
```

6. Spielstatus

Es folgt die Beschreibung des Spielstatus, der vor jeder Zugaufforderung an die Clients gesendet wird. Das Spielstatus-Tag ist dabei noch in einem *data*-Tag der Klasse *memento* gewrappt:

6.1. memento

status wie in 6.2 definiert

```
<room roomId="RID">
  <data class="memento">
    status
  </data>
</room>
```

6.2. Status

Z aktuelle Zugzahl

S Spieler, der das Spiel gestartet hat (RED/BLUE)

C Spieler, der an der Reihe ist (RED/BLUE)

red, blue wie in 6.3 definiert

board Das Spielbrett, wie in 6.4 definiert

lastMove Letzter gettigter Zug (nicht in der ersten Runde), wie in 6.6 definiert

condition Spielergebnis, wie in 11 definiert; nur zum Spielende

```
<state class="state" turn="Z" startPlayer="S" currentPlayer="C">
  red
  blue
  [board]
  [lastMove]
  [condition]
</state>
```

6.3. Spieler

C Farbe (RED/BLUE)

N Anzeigename

P Punktekonto

X X Koordinate des Spielers

Y Y Koordinate des Spielers

D Ausrichtung des Spielers, $D \in \{RIGHT, UP_RIGHT, UP_LEFT, LEFT, DOWN_LEFT, DOWN_RIGHT\}$

S Geschwindigkeit des Spielers

K Kohleeinheiten des Spielers

T Teilabschnitt, auf dem sich der Spieler befindet

PA Anzahl der Passagiere

```
<C displayName="N" color="C" points="P" x="X" y="Y"
direction="D" speed="S" coal="K" tile="T" passenger="PA"/>
```


6.4. Spielbrett

FIELD Ein Spielfeld wie in 6.5 definiert.

```
<board>
  <tiles>
    <tile visible="true" index="0" direction="0">
      {FIELD}
    ..
    {FIELD}
  </tiles>
  ..
</tiles>
</board>
```

6.5. Spielfeld

Type Typ des Feldes (WATER/BLOCKED/PASSENGER_i/GOAL/SANDBAR/LOG).
($i \in \{0..5\}$ gibt die Richtung an)

X X-Koordinate des Feldes

Y Y-Koordinate des Feldes

P Punkte des Feldes

```
<field type="Type" x="X" y="Y" points="P"/>
```

6.6. Letzter Zug

Der letzte Zug ist ein Move (siehe hierzu 3.2).

ACTIONS Teilzge des Zuges (siehe hierzu 3.3).

```
<lastMove>
  ACTIONS
</lastMove>
```

7. Zug-Anforderung

Eine einfache Nachricht fordert zum Zug auf:

```
<room roomId="RID">
  <data class="sc.framework.plugins.protocol.MoveRequest"/>
</room>
```

8. Fehler

Ein „spielfhiger“ Client muss nicht mit Fehlern umgehen knnen. Fehlerhafte Zge beispielsweise resultieren in einem vorzeitigen Ende des Spieles, das im nchsten gesendeten Gamestate erkannt werden kann (siehe 11).

MSG Fehlermeldung

```
<room roomId="RID">
<error message="MSG">
<originalRequest>
Request, der den Fehler verursacht hat
</originalRequest>
</error>
</room>
```

9. Spiel pausiert

Ein „spielfhiger“ Client muss Pausierungsnachrichten nicht beachten, da er nur auf Aufforderungen (Zug-Aufforderung siehe 7) des Servers handelt.

C Farbe (RED/BLUE)

N Anzeigename

P Punktekonto

X X Koordinate des Spielers

Y Y Koordinate des Spielers

D Ausrichtung des Spielers $D \in \{RIGHT, UP_RIUGHT, UP_LEFT, LEFT, DOWN_LEFT, DOWN_RIGHT\}$

S Geschwindigkeit des Spielers

K Kohleeinheiten des Spielers

T Spielabschnitt auf dem sich der Spieler befindet

PA Anzahl der Passagiere

```
<room roomId="RID">
<data class="paused">
<nextPlayer class="player" displayName="N" color="C" points="P" x="X" y="Y"
direction="D" speed="S" coal="K" tile="T" passenger="PA"/>
</data>
</room>
```

10. Spiel verlassen

Wenn ein Client den Raum verlässt, bekommen die anderen Clients eine entsprechende Meldung vom Server.

```
<left roomId="RID"/>
```

11. Spielergebnis

Zum Spielende enthält der Spielstatus eine *Condition*, der das Spielergebnis entnommen werden kann:

W Gewinner (RED/BLEUE), bei Unentschieden wird kein winner mitgeschickt.

R Text, der den Grund für das Spielende erklärt

```
<condition winner="W" reason="R"/>
```

Teil III.

berblick

Hier nochmal ein kurzer berblick, der etwas genauer zeigt, wie die Kommunikation ablaufen kann.

1. Ein Spiels server startet ein Spiel und wartet auf den Client (die Clients).
2. Der Client stellt eine TCP Verbindung zum Spiels server her (er kennt dessen IP-Adresse und Port)
3. Die Verbindung ist aufgebaut und der Client sendet

```
<protocol>
  <join gameType="swc_2017_mississippi_queen"/>
```

4. Der Server sendet

```
<protocol>
  <joined roomId="65d3d704-87d9-42eb-8995-51c3e6324513"/>
```

5. der Client merkt sich die roomId.
6. der Server startet das Spiel und sendet

```
<room roomId="65d3d704-87d9-42eb-8995-51c3e6324513">
  <data class="welcomeMessage" color="blue"/>
</room>
```

7. der Client merkt sich seine Spielerfarbe.
8. der Server sendet das Ausgangsspielfeld:

```
<room roomId="65d3d704-87d9-42eb-8995-51c3e6324513">
  <data class="memento">
    <state class="state" turn="0" startPlayer="RED" currentPlayer="RED">
      <red displayName="Spieler 1" color="RED" points="0" x="-1" y="1"
        direction="RIGHT" speed="1" coal="6" tile="0" passenger="0"/>
      <blue displayName="Spieler 2" color="BLUE" points="0" x="-1" y="-1"
        direction="RIGHT" speed="1" coal="6" tile="0" passenger="0"/>
    <board>
      <tiles>
```

```

    <tile visible="true" index="0" direction="0">
      <field type="WATER" x="0" y="19" points="0"/>
      <field type="WATER" x="0" y="20" points="0"/>
      <field type="WATER" x="0" y="21" points="1"/>
      <field type="WATER" x="0" y="22" points="1"/>
      <field type="WATER" x="0" y="23" points="1"/>
      ...
    </tile>
    ..
    <tile visible="true" index="5" direction="3">
      ...
      <field type="WATER" x="34" y="5" points="3"/>
      <field type="GOAL" x="34" y="6" points="4"/>
      <field type="GOAL" x="34" y="7" points="4"/>
      <field type="GOAL" x="34" y="8" points="4"/>
      <field type="WATER" x="34" y="9" points="3"/>
    </tile>
    <fields>
  </board>
</state>
</data>
</room>

```

9. da der andere Spieler anfgt folgt noch eine *memento*-Nachricht vom Server, die den Spielstatus nach dem ersten Zug von Spieler 1 beinhaltet.
10. es folgt die erste Zug-Aufforderung vom Server:

```

<room roomId="65d3d704-87d9-42eb-8995-51c3e6324513">
  <data class="sc.framework.plugins.protocol.MoveRequest"/>
</room>

```

11. Der Client antwortet mit einem Zug:

```

<room roomId="65d3d704-87d9-42eb-8995-51c3e6324513">
  <data class="move">
    <actions>
      <acceleration order="0" acc="2"/>
      <step order="1" distance="3"/>
    </actions>
  </data>
</room>

```

12. so geht es weiter...

13. die letzte Nachricht enthält ein

`</protocol>`

Daraufhin wird die Verbindung geschlossen