

# Software Challenge XML-Dokumentation Piranhas

Ziel dieser Dokumentation ist es, die XML-Schnittstelle der Softwarechallenge festzuhalten.

Wir freuen uns über sämtliche Verbesserungsvorschläge. Die Dokumentation kann [direkt auf GitHub editiert](#) werden, einzige Voraussetzung ist eine kostenlose Registrierung bei GitHub. Ist man angemeldet, kann man ein Dokument auswählen (ein guter Startpunkt ist die Datei [index.adoc](#) welche Verweise auf alle Sektionen der Dokumentation enthält) und dann auf den Stift oben rechts klicken. Alternativ auch gern eine E-Mail an [svk@informatik.uni-kiel.de](mailto:svk@informatik.uni-kiel.de).

## Einleitung

Wie in den letzten Jahren wird zur Client-Server Kommunikation ein XML-Protokoll genutzt. 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 Spielservers 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. == Spiel betreten Wenn begonnen wird mit dem Server zu kommunizieren, muss zuallererst

<protocol>

gesendet werden, um die Kommunikation zu beginnen.

# Ohne Reservierungscode

Betritt ein beliebiges offenes Spiel:

```
<join gameType="swc_2019_piranhas" />
```

Sollte kein Spiel offen sein, wird so ein neues erstellt. Je nachdem ob `paused` in `server.properties` `true` oder `false` ist, wird das Spiel pausiert gestartet oder nicht. Der Server antwortet darauf mit:

- ROOM\_ID Id des GameRooms

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

Alle administrativen Clients werden ebenfalls darüber benachrichtigt und erhalten folgende Nachricht:

- ROOM\_ID Id des GameRooms

```
<joinedGameRoom roomId="ROOM_ID" existing="false" />
```

Falls bereits ein GameRoom offen war, ist dementsprechend `existing` `true`.

# Mit Reservierungscode

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

## Join mit RC

- RC Reservierungscode

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

## Welcome Message

Der Server antwortet darauf nur, wenn der zweite Client ebenfalls verbunden ist:

- ROOM\_ID Id des GameRooms
- COLOR Spielerfarbe also `red` oder `blue`
- STATUS GameState wie in [Status](#)

```

<joined roomId="ROOM_ID" />
<room roomId="ROOM_ID">
  <data class="welcomeMessage" color="COLOR"></data>
</room>
<room roomId="ROOM_ID">
  <data class="memento">
    STATUS
  </data>
</room>

```

# Züge senden

## Der Move

Der Move ist die Antwort auf den MoveRequest des Servers.

### MoveRequest

#### ROOM\_ID

ID des GameRooms, dient der Zuordnung der Antworten des Clients.

```

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

```

### Senden

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

#### ROOM\_ID

ID des GameRooms, aus dem MoveRequest.

#### ZUG

Zug wie in ZUG

```

<room roomId="ROOM_ID">
  ZUG
</room>

```

## ZUG

#### X

X-Koordinate des zu ziehenden Spielsteines oder des Zielfeldes

## Y

Y-Koordinate des zu ziehenden Spielsteines oder des Zielfeldes

## Z

Z-Koordinate des zu ziehenden Spielsteines oder des Zielfeldes

## COLOR

Farbe des zu setzenden Spielsteines (RED/BLUE)

## TYPE

Typ des zu setzenden Spielsteines (ANT, BEE, BEETLE, GRASSHOPPER, SPIDER)

Es gibt drei Arten von Zügen. Entweder man setzt einen eigenen noch nicht gesetzten Stein auf ein Zielfeld:

```
<data class="setmove">
  <piece owner="COLOR" type="TYPE" />
  <destination x="X" y="Y" z="Z"/>
</data>
```

Oder man bewegt einen eigenen Stein auf dem Spielfeld:

```
<data class="dragmove">
  <start x="X" y="Y" z="Z"/>
  <destination x="X" y="Y" z="Z"/>
</data>
```

Oder man kann weder setzen noch ziehen, dann ist es erlaubt, auszusetzen:

```
<data class="missmove" />
```

## Debughints

Zügen können Debug-Informationen beigefügt werden:

```
<hint content="S" />
```

Damit sieht beispielsweise ein Zug so aus:

```

<room roomId="ROOM_ID">
  <data class="dragmove">
    <start x="X" y="Y" z="Z"/>
    <destination x="X" y="Y" z="Z"/>
    <hint content="Dies ist ein Hint." />
    <hint content="noch ein Hint" />
  </data>
</room>

```

## 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:

### memento

- ROOM\_ID Id des GameRooms
- STATUS Gamestate wie in [Status](#)

```

<room roomId="ROOM_ID">
  <data class="memento">
    STATUS
  </data>
</room>

```

### 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 [Spieler](#) definiert
- board Das Spielbrett, wie in [Spielbrett](#) definiert
- undeployed Noch nicht gesetzte Spielsteine, siehe [Nicht gesetzte Spielsteine](#)

```

<state class="state" turn="Z" startPlayerColor="S" currentPlayerColor="C">
  red
  blue
  [board]
  [undeployed]
</state>

```

# Spieler

- c Farbe (red/blue)
- C Farbe (RED/BLUE)
- N Name des Spielers

```
<c displayName="N" color="C" />
```

# Spielbrett

- FIELD Ein Spielfeld wie in [Spielfeld](#) definiert.

```
<board>
  <fields>
    <null/>
    ...
    <null/>
    <field x="-5" y="2" z="3" isObstructed="false"/>
    ...
    FIELD
    ...
    <field x="-5" y="5" z="0" isObstructed="false"/>
  </fields>
  ...
</board>
```

`<board>` enthaelt 11 `<fields>` Tags die wiederrum 11 Eintraege haben. Das ganze repraesentiert ein 11x11 2D-Array. Da das Spielfeld aber nicht rechteckig ist sondern die Form eines Hexagons hat, sind nicht alle Array-Eintraege durch Felder belegt, sondern manche durch `<null/>`. Das Feld mit den X- und Y-Koordinaten `x` und `y` ist im Array an der Stelle `(x+5, y+5)` gespeichert.

# Spielfeld

- X X-Koordinate
- Y Y-Koordinate
- Z Z-Koordinate
- OBSTRUCTED Blockiert (true) oder nicht (false)
- PIECES Spielsteine (falls auf dem Feld vorhanden). Oberster Spielstein kommt als letztes in der Liste. Siehe [Spielstein](#).

```
<field x="X" y="Y" z="Z" isObstructed="OBSTRUCTED">
  PIECES
</field>
```

## Spielstein

- OWNER Farbe (RED/BLUE)
- TYPE Insektentyp (ANT, BEE, BEETLE, GRASSHOPPER, SPIDER)

```
<piece owner="OWNER" type="TYPE"/>
```

## Nicht gesetzte Spielsteine

```
<undeployedRedPieces>
  <piece owner="RED" type="BEE"/>
  ...
</undeployedRedPieces>
<undeployedBluePieces>
  <piece owner="BLUE" type="BEE"/>
  ...
</undeployedBluePieces>
```

## Zug-Anforderung

Eine einfache Nachricht fordert zum Zug auf:

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

## Fehler

Ein “spielfähiger” Client muss nicht mit Fehlern umgehen können. Fehlerhafte Züge beispielsweise resultieren in einem vorzeitigen Ende des Spieles, das im nächsten gesendeten Gamestate erkannt werden kann (siehe [Spielergebnis](#)).

- MSG Fehlermeldung

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

## Spiel verlassen

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

- ROOM\_ID Id des GameRooms

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

## Spielergebnis

Zum Spielende enthält der Spieler das Ergebnis:

- ROOM\_ID Id des GameRooms
- R1, R2 Text, der den Grund für das Spielende erklärt
- CAUSE1, CAUSE2 Grund des Spielendes  
(REGULAR/LEFT/RULE\_VIOLATION/SOFT\_TIMEOUT/HARD\_TIMEOUT)
- WP1, WP2 Siegpunkte der jeweiligen Spieler, 0 verloren, 1 unentschieden, 2 gewonnen
- NAME Anzeigename des Spielers
- COLOR Farbe des Siegers
- S3 S1 oder S2 je nachdem wer gewonnen hat



```

<room roomId="ROOM_ID">
  <data class="result">
    <definition>
      <fragment name="Gewinner">
        <aggregation>SUM</aggregation>
        <relevantForRanking>true</relevantForRanking>
      </fragment>
      <fragment name="Ø freie Felder">
        <aggregation>AVERAGE</aggregation>
        <relevantForRanking>true</relevantForRanking>
      </fragment>
    </definition>
    <score cause="CAUSE1" reason="R1">
      <part>WP1</part>
      <part>S1</part>
    </score>
    <score cause="CAUSE2" reason="R2">
      <part>WP2</part>
      <part>S2</part>
    </score>
    <winner class="player" displayName="NAME" color="COLOR" />
  </data>
</room>

```

## Spielverlauf

Der Server startet (StandardIp: localhost 13050).

Nun gibt es zwei Varianten ein Spiel zu starten, eine durch einen Administratorclient die andere durch hinzufügen der Spieler zu einen Spieltyp:

### Variante 1 (AdminClient Mit Reservierungscode)

Ein Client registriert sich als Administrator mit dem in server.properties festgelegten Passwort p:

```

<protocol><authenticate passphrase="p" />

```

Dann kann ein Spiel angelegt werden:

```

<prepare gameType="swc_2019_piranhas">
  <slot displayName="p1" canTimeout="false" shouldBePaused="true" />
  <slot displayName="p2" canTimeout="false" shouldBePaused="true" />
</prepare>

```

Der Server antwortet darauf mit einer Nachricht, die die ROOM\_ID und Reservierungscode für die

beiden Clients enthält:

```
<protocol>
  <prepared roomId="871faccb-5190-4e44-82fc-6cdcbb493726">
    <reservation>RC1</reservation>
    <reservation>RC2</reservation>
  </prepared>
```

Der Administratorclient kann nur ebenfalls als Observer des Spiels genutzt werden, indem ein entsprechender Request gesendet wird. Dadurch wird das derzeitige Spielfeld (**memento**) ebenfalls an den Administratorclient gesendet.

```
<observe roomId="871faccb-5190-4e44-82fc-6cdcbb493726" />
```

Clients die auf dem Serverport (localhost 13050) gestartet werden können so über diesen Code joinen.

```
<protocol>
  <joinPrepared reservationCode="RC1" />
```

```
<protocol>
  <joinPrepared reservationCode="RC2" />
```

## Variante 2 ((eventuell) ohne AdminClient Ohne Reservierungscode)

Die Clients wurden auf dem Serverport (Standard: localhost 13050) gestartet.

Sie können sich mit folgender Anfrage einen bereits offenen Spiel gleichen Typs beitreten oder, falls kein Spiel des Typs vorhanden selbst eines starten:

```
<protocol>
  <join gameType="swc_2019_piranhas" />
```

Der Server antwortet mit:

```
<protocol>
  <joined roomId="871faccb-5190-4e44-82fc-6cdcbb493726" />
```

## Weiterer Spielverlauf

Der Server antwortet jeweils mit der WelcomeMessage ([Welcome Message](#)) und dem ersten GameState ([memento](#)) sobald beide Spieler verbunden sind.

```
<room roomId="871faccb-5190-4e44-82fc-6cdcbb493726">
  <data class="welcomeMessage" color="red" />
</room>
<room roomId="871faccb-5190-4e44-82fc-6cdcbb493726">
  <data class="memento">
    <state class="state" startPlayerColor="RED" currentPlayerColor="RED" turn="0">
      <red displayName="Red" color="RED"/>
      <blue displayName="Blue" color="BLUE"/>
      <board>
        <fields>
          <null/>
          <null/>
          <null/>
          <null/>
          <null/>
          <field x="-5" y="0" z="5" isObstructed="false"/>
          <field x="-5" y="1" z="4" isObstructed="false"/>
          <field x="-5" y="2" z="3" isObstructed="false"/>
          <field x="-5" y="3" z="2" isObstructed="false"/>
          <field x="-5" y="4" z="1" isObstructed="false"/>
          <field x="-5" y="5" z="0" isObstructed="false"/>
        </fields>
        <fields>
          <null/>
          <null/>
          <null/>
          <null/>
          <field x="-4" y="-1" z="5" isObstructed="false"/>
          <field x="-4" y="0" z="4" isObstructed="false"/>
        </fields>
        ...
      </fields>
    </board>
  </state>
</data>
</room>
```

Der erste Spieler erhält dann eine Zugaufforderung ([MoveRequest](#)), falls in `server.properties` `paused` auf `false` gesetzt wurde. Falls das Spiel pausiert ist, muss das Spiel durch einen Administratorclient gestartet werden:

Verbinden des Administratorclients (falls es sich um die erste Kontaktaufnahme zum Server

handelt, ansonsten <protocol> weglassen).

```
<protocol>
  <authenticate passphrase="examplepassword" />
```

Pausierung aufheben:

```
<pause roomId="871facb-5190-4e44-82fc-6cdcbb493726" pause="false" />
```

Daraufhin wird der erste Spieler aufgefordert einen Zug zu senden:

```
<room roomId="871facb-5190-4e44-82fc-6cdcbb493726">
  <data class="sc.framework.plugins.protocol.MoveRequest" />
</room>
```

Der Client des CurrentPlayer sendet nun einen Zug (ZUG):

```
<room roomId="871facb-5190-4e44-82fc-6cdcbb493726">
  <data class="setmove">
    <piece owner="RED" type="BEETLE" />
    <destination x="-2" y="0" z="2"/>
  </data>
</room>
```

So geht es abwechselnd weiter, bis zum Spielende (Spielergewinn). Die letzte Nachricht des Servers endet mit:

```
</protocol>
```

Danach wird die Verbindung geschlossen.