Netzwerk-Protokoll

TextureSync

Version	1.0.0
Datum	03.04.2019
Autor	Robin Willmann
Projektmitglieder	Hendrik Schutter, Lukas Fürderer, Robin Willmann, Jannik Seiler

Inhaltsverzeichnis

1 Grundsätzliches

Es wird eine Client-Server-Architektur verwendet. Um das Netzwerkprotokoll möglichst einfach und debuggbar zu halten, bietet sich JSON über TCP an. Dieses wird in eine eigene Paketstruktur verpackt, um so auch große Binär-Daten (z.B. Texturen) über die selbe Verbindung zu übertragen.

Eine Verbindung wird immer vom Client initiiert. Nach jeder Anfrage kann der Client die Verbindung für weitere Anfragen offen halten oder diese zur Beendigung schließen.

Der Server schließt Verbindungen bei Verletzungen des Protokolls oder wenn mindestens 10 Minuten lang kein Datenaustausch mehr stattgefunden hat.

1.1 Ports

Der Server verwendet **TCP-Port 10796** für eingehende Verbindungen. Es wird sowohl IPv6 als auch IPv4 akzeptiert.

1.2 Paketformat

Daten werden über TCP gesendet. Da TCP stream-based ist, wird folgende Struktur verwendet, um Pakete zu emulieren.

```
<Payload-Typ   : 1 byte>
<Reseviert     : 3 bytes>
<Payload-Länge : 4 bytes>
<Payload       : Payload-Länge bytes>
```

Alle Zahlenwerte werden als Big-Endian übertragen.

Mögliche Payload-Typen sind:

Typ	Maximales Payload
0 = Error	1024 Bytes (Optionale Fehlerbeschreibung, UTF-8)
1 = JSON	16 MiB
2 = Binary	512 MiB

Wird das maximale Payload überschritten, wird die Verbindung sofort geschlossen. Dies dient dazu, zu verhindern, dass ein Teilnehmer mehr Daten entgegen nimmt, als dieser im RAM behalten kann.

2 Befehle

2.1 Definitionen

Im Folgenden sind sind Datentypen für JSON definiert, welche in dem Protokoll wiederverwendet werden:

Für *String*, *Number*, *Array von* <..> siehe JSON-Standart.

UUID ::= <String>

UUID nach Version 4

Beispiele

- "a78c59fc-4198-421a-8ba4-db232ad7b91e"
- "1f010407-130f-432c-8463-6c61fdfb8c14"
- "ecb109bb-d9d6-494d-9d5e-b1e44734e20d"

Format ::= "png" | "jpeg"

Dateiformat

Beispiele

- "png"
- "jpeg"

Resolution ::= [<Number>, <Number>]

Die erste Nummer stellt die Weite in Pixeln dar, die Höhe in Pixeln wird durch die zweite Nummer repräsentiert.

Beispiele

- [1024, 1024]
- [2048, 512]
- [13, 400]

Tag ::= <String>

Stellt ein Tag dar. Kann Groß- und Kleinbuchstaben beinhalten.

Hinweis: Vergleiche von Tags sind nicht Case-Sensitiv. Die Darstellung in der UI jedoch unter Umständen schon.

Beispiele

- "Holz"
- "mEtALL"
- "Chesse Cake"

Date ::= <String>

im Format "yyyy-MM-dd", siehe Javadoc unter *java.text.SimpleDateFormat* für mehr Informationen.

Beispiele

- "2019-03-04"
- "2017-12-21"

Hash ::= <String>

Sha256-Hash von z.B. Texturdaten oder anderen Binärdaten, in Hexadezimal-Darstellung. Kann Groß- oder Kleinbuchstaben enthalten. Dies wird genutzt, um auf diese zu verweisen.

Beispiele

- "a98f43a976e5b501961635b981022ebaf98321b97055ead4d8d4de55114015e7"
- "02a08f7d697a93937cc5ace273a534c2eb021ae76b7c15ba146d279d57898893"
- "A6A04ADC2E6D580B8E37CE8F4784652BE6D668EC1FB340B971DD8E8A582CE6BC"
- "7bdc65d8550b0A4FBC899550bbda87DAA2E780D618A66a1F7813967ECF6C0831"

Texture ::= {
    id: <UUID>,
    name: <String>,
    tags: <Array von <Tag>>,
    format : <Format>,
    resolution: <Resolution>,
    added_on: <Date>,
    texture_hash: <Hash>
}

Stellt einen Textur-Eintrag mit Metadaten dar.

2.2 Ping

Dieser Befehl dient zum Überprüfen der Verbindung.

Client sendet nach Schema:

```
type = JSON
{
    "ping": {}
}
```

Server antwortet nach Schema:

```
type = JSON
{
    "pong": {}
}
```

2.3 Query

Client sendet nach Schema:

Zusammenhänge Eingaben werden als <String> in einem Array übertragen.

```
type = JSON
{
    "query": {
        "query" : <Array of <String>>
    }
}
```

Server antwortet nach Schema:

```
type = JSON
<Array of <Texture>>
```

2.4 Get Texture

Client sendet nach Schema:

```
type = JSON
{
    "get_texture": {
        "id" : <UUID> | null,
        "name" : <String> | null,
    }
}
```

Hierbei muss entweder das Feld "id" oder das Feld "name" gesetzt werden. Andernfalls wird type=Error gesendet,

Der Server antwortet nach Schema [Textur gefunden]:

```
type = JSON
<Texture>
```

Der Server antwortet nach Schema [Textur unbekannt]:

```
type = JSON
null
```

2.5 Get Texture File

Client sendet nach Schema:

```
type = JSON
{
    "get_texture_file": {
        "texture_hash" : <Hash>,
    }
}
```

Der Server antwortet nach Schema [Textur-Datei gefunden]:

```
type = Binary
Textur-Datei
```

Der Server antwortet nach Schema [Textur-Datei unbekannt]

```
type = Error
Fehlerbeschreibung = "texture not found"
```

2.6 Get Texture Preview

Client sendet nach Schema:

```
type = JSON
{
    "get_texture_preview": {
        "texture_hash" : <Hash>,
    }
}
```

Der Server antwortet nach Schema [Textur-Datei gefunden]:

```
type = Binary
Textur-Preview
```

Der Server antwortet nach Schema [Textur-Datei unbekannt]:

```
type = Error
Fehlerbeschreibung = "texture not found"
```

2.7 Replace Texture

Client sendet nach Schema:

```
type = JSON
{
    "replace_texture": {
        "old": <Texture> | null,
        "new": <Texture> | null,
    }
}
```

Diese Anfrage dient dazu alte Texturen zu löschen und neue hinzufügen. Ein Löschen und gleichzeitiges Hinzufügen, ergibt ein Update der Textur.

Falls "old" != null, wird die hier angegebene Textur gelöscht. Wird diese nicht exakt gleich vorgefunden, schlägt diese Anfrage fehl (type = Error, Fehlerbeschreibung = "texture not found"). In diesem Fall wird "new" nicht berücksichtigt.

Falls "new" != null, wird die hier angegebene Textur zum System hinzugefügt. Sollte die angegebene "new.id" oder der angegebene "new.name" schon vorhanden sein, schlägt diese Anfrage fehl (type = Error).

Die Fehlerbeschreibung lautet "id already in use" für eine doppelte Id bzw. "name already in use" für einen doppelten Namen.

Diese Semantik wurde gewählt, damit ein Update atomar ist und doppelte Anfragen zu Fehlern führen.

Der Server antwortet nach Schema ["texture_hash" bekannt]:

```
type = JSON
true
```

Die Anfrage wird damit beendet.

Der Server antwortet nach Schema ["texture_hash" unbekannt]:

```
type = JSON
{
    "get_texture_file": {
        texture_hash : <Hash>,
    }
}
```

Woraufhin der Client die Textur-Datei sendet:

```
type = Binary
Textur-Datei
```

Der Server bestätigt dies dann mit:

```
type = JSON
true
```

3 Changelog

Version	Änderung
0.9.0	-
0.9.1	Formulierung und Rechtschreibung
0.10.0	Füge Ping-Befehl hinzu.
0.11.0	Server darf Verbindungen schließen, Fehlerbeschreibungen festgelegt
1.0.0	Review fertig.

4 Anmerkungen

Dieses Dokument wird ergänzt, falls das WK#4 (Automatische Konfiguration des Clients) umgesetzt wird.