Netzwerk-Protokoll

TextureSync

Version	1.4.0
Datum	25.04.2019
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.

Eine Automatisches Auffinden des Servers wird in Abschnitt 4 beschreiben.

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. (Siehe unten, Fehlerhandhabung für Details.)
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 ::= [<Weite:Number>, <Höhe: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 ::= [<Jahr:Number>, <Monat:Number>, <Tag:Number>]

Beispiele

- [2019, 3, 4] für 4. März 2019
- [2017, 12, 21] für 21. Dezember 2017

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 Fehler *400 Bad Request* 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
404 File Not Found.
```

2.6 Get Texture Preview

Client sendet nach Schema:

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

Das Feld "desired_format" gibt an, in welchem Dateiformat das Antwort-Previewbild sein wird.

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

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

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

```
type = Error
404 File 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 (Fehler: *409 Conflict*). 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 (Fehler: *409 Conflict*).

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

*Hinweis: Um zu überprüfen, ob ein Name bereits vergeben ist, sollte **Get Texture** verwendet werden. (IDs werden sowieso zufällig generiert.)*

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 Fehlerhandhabung

Fehlerbeschreibungen sind nach folgendem Schema aufgebaut:
<Fehlercode in Dezimal, Ascii><Leerzeichen><Fehlertext>
Bsp:

- 500 Internal Server Error

Die Fehlercodes sind an HTTP-Statuscodes angelehnt.

Das Folgende ist eine Aufstellung möglicher Fehlernachrichten, auf die in teilen oben verwiesen wurde. Der **Fehlertext kann von dieser Aufstellung abweichen**, um genauere Informationen wiederzugeben. Ein Vergleich muss daher immer anhand der Fehlernummer durchgeführt werden.

Nummer	Text	Beschreibung
400	Bad Request.	Anfrage hat ungültiges Format. Hierzu zählen: • Fehlerhalte Semantik. • Falsches Encoding.
404	File Not Found.	Unbekannte Textur-Datei angefordert.
409	Conflict.	Name oder Id bereits vorhanden. Oder Löschversuch fehlgeschlagen, wegen möglicher Inkonsistenz.
500	Internal Server Error.	Unerwartbare seltene Sonstige Fehler. Z.b. Festplatte nicht lesbar etc.
501	Not Implemented.	Funktionalität wurde noch nicht implementiert. Dieser Fehlercode wird nur in der Entwicklungsphase verwendet.

4 Automatische Konfiguration (Auto Connect) WK#4

Im folgenden wird der Mechanismus beschrieben mit dem der Client automatisch die Verbindung zum Server findet.

Verwendet hierfür wird IPv6 Multicast verwendet.

4.1 Vorgenweise des Servers

Der Server öffnet einen Socket und registriert sich auf **UDP-Port 10796**.
Anschließend *joined* er die **Multicast-Gruppe ff02::dd42::c0fe.** Wie sich an der Multicast-Adresse lesen lässt, ist diese Site-Local, kann also innerhalb eines gesamten lokalen Netzwerkes verwendet werden.

Der Server wartet nun auf ein UDP-Packet mit Inhalt „**TextureSync**“ (UTF-8, ohne Null am Ende). Erhält er ein solches Packet, antwortet er mit einem 2-Byte-großen Packet an die Absender-Adresse seine Tcp-Port-Nummer (siehe 1.1) als 16bit Big-Endian Wert.

4.2 Vorgenweise des Clients

Der Client findet den Server, indem er ein UDP-Packet mit Inhalt „**TextureSync**“ (UTF-8, ohne Null am Ende) an die Multicast-Adresse **ff02::dd42::c0fe** Port **10796** sendet**.** Er erhält eine Antwort vom Server, mit der Tcp-Port-Nummer als 16bit Big-Endian Wert. Auf die erhaltene TCP-Port-Nummer und die Absender-Adresse des Server verbindet sich nun der Client. Somit wurde eine Verbindung aufgebaut.

Erhält der Client innerhalb von **400 ms** keine Antwort, versucht er noch **9 weitere** Mal. Ansonsten gilt das Auffinden des Servers als gescheitert.

5 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.
1.1.0	+ Fehlerhandhabung: Genauere Aufstellung in extra Abschnitt.
1.2.0	Füge "desired_format" zu Get Texture Preview hinzu.
1.3.0	Ändere Format des Datums.
1.4.0	Hinzufügen: Automatische Konfiguration des Clients, Anmerkung hirzu am Ende entfernt.