# SPIT

Show Protocol Incorporated Trigger
Version 1



StandardPort: 15120 (individuell veränderbar)

## Prolog

Für den Ablauf einer komplexen Show müssen verschiedene Gewerke zusammenarbeiten. In der Regel verwendet jedes Gewerke eine eigene Software (Steuerungs-Hardware).
Aktuell werden SMPTE Timecodes oder MIDI Befehle verwendet um die Zusammenarbeit der unterschiedlichen Softwaren zu synchronisieren.

### SMPTE Timecode

Der SMPTE Timecode wird an zentraler Stelle erzeugt und an alle "angeschlossenen" Systeme übertragen. Er kann dazu verwendet werden, einen synchronen Start und eventuell einen Gleichlauf der unterschiedlichen Softwaren zu gewährleisten. Der eigentliche Ablauf ist jedoch jeder einzelnen Software überlassen.
Nachteile: Es kann nur ein zeitlich festgelegter Ablauf realisiert werden. Unterbrechungen oder Sprünge im Ablauf sind nicht einfach steuerbar.

### MIDI Befehle

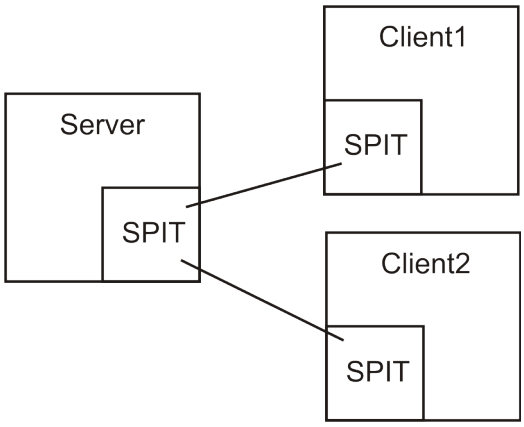
Für MIDI Befehle müssen in jeder Software MIDI Parameter definiert werden. Wird ein MIDI Befehl gesendet, wird auf der Gegenseite die dort festgelegte Aktion ausgeführt. Hier sind Unterbrechungen und Sprünge im Showablauf möglich.
Nachteile: Die Bedeutung der Parameter ist der jeweiligen Gegenseite nicht bekannt, bzw. muss dort jeweils händisch festgelegt werden. Dies macht die Bedienung anfällig für Fehler und kompliziert, ein automatischer Abgleich der möglichen Befehle/Aktionen ist nicht möglich.

### SPIT

Mit SPIT soll es ermöglicht werden, andere Softwaren über Trigger zu steuern. Diese Trigger haben einen sprechenden Namen, der frei vergeben werden kann. Hierbei gibt es eine Software/Hardware, die für die Verteilung der Trigger und für das Abfeuern der Trigger zuständig ist (SPIT Server) und andere Software/Hardware, die sich bei einem SPIT Server anmeldet und für die Ausführung bestimmter Aktionen zuständig ist (SPIT Client). Eine DMXSteuersoftware könnte somit über einen Trigger bei einer anderen Software eine Animation auslösen.

Die Trigger können flexibel erzeugt und verändern werden,  durch ihren sprechenden Namen sind sie leicht zuzuordnen.

Die Verbindung zwischen Client und Server wird über ein Netzwerk (LAN, WLan, Internet) hergestellt.




# Konzept

## SPIT Server

Der SPIT Server kann eine Software oder eine Hardware sein, die das SPIT Protokoll implementiert hat. Dies wäre z.B. eine DMX Steuersoftware, ein DMX Hardwarepult, etc.

Der SPIT Server ist die zentrale Stelle für alle SPIT Clients. Der SPIT Server sammelt alle Trigger und gibt diese an alle angemeldeten  SPIT Clients weiter.

Der SPIT Server ist die Stelle, an der Trigger abgefeuert werden.

Der SPIT Server arrangiert die Trigger in einer zeitlichen Abfolge (z.B. in einer Timeline), er kann zu jedem Zeitpunkt einen Trigger abfeuern. Er kann auch neue Trigger erzeugen, verändern oder vorhandene Trigger löschen.

## SPIT Client

Der SPIT Client kann eine Software oder Hardware sein, die das SPIT Protokoll implementiert hat. Dies wäre z.B. eine Videosoftware, eine Mischpult, etc.

Der SPIT Client bekommt über den SPIT Server alle Trigger gelistet, er kann aber auch eigene Trigger anlegen, die wiederum über den SPIT Server an alle anderen SPIT Clients weitergegeben werden.

Der SPIT Client kann einem Trigger eine Aktion zuweisen. Wird vom SPIT Server dieser Trigger abgefeuert, so wird bei dem SPIT Client die zugewiesene Aktion ausgeführt.

## Aufbau der SPIT Trigger (SPIT Typ – SPIT Objekt)

Ein Trigger besteht aus zwei Teilen:

1. SPIT Typ
   Der SPIT Typ beschreibt eine auszulösende Aktion/Funktion oder den Typ/Klasse einer Aktion/Funktion.
   Diese Aktion könnten im Ablauf mehrmals ausgeführt werden. Ein SPIT Typ löst keine Aktion aus, er beschreibt oder klassifiziert diese nur. Ein SPIT Typ weis auch nicht wie die Aktion genau ausgeführt wird, dies ist die Aufgabe der jeweiligen Software, die ein SPIT Objekt empfängt.
2. SPIT Objekt

Das SPIT Objekt ist das eigentliche auslösende Element. Es besitzt eine Referenz zu einem SPIT Typ. Es können mehrere SPIT Objekte mit derselben Referenz auf einen SPIT Typ existieren. Somit kann kann eine Aktion (beschrieben/klassifiziert durch den SPIT Typ) an unterschiedlichen Stellen ausgeführt werden.
Die Aktion kann in der Client Software beliebig zugeordnet werden, der referenzierte SPIT Typ ist keine zwingende Vorgabe, er dient zur Strukturierung.

Wenn der Server Trigger/Aktionen auslösen möchte, sendet er eine Play-Sequenz:

PlaySPITStart
PlaySPITObject, .. (für jedes auszulösende SPIT Objekt)
PlaySPITEnd

Umgekehrt kann auch der Client solche Play-Sequenzen senden, wenn der Server Remote SPIT Objekte zu Verfügung stellt.

**Kurzer Vergleich mit einem MIDI-Controller:**

MIDI

Bei einem MIDI Controller sendet der Controller Bytefolgen an einen angeschlossenen Client (Software, Mixer, etc.). Beim Client muss eingetragen sein, wie er auf bestimmte Byte-Sequenzen reagiert. Die Bytefolgen übermitteln keine inhaltlichen Informationen, sie sind in der MIDI-Spezifikation definiert, die ursprünglich für den Austausch von musikalischen Steuerinformationen entworfen wurde. Der informelle Austausch zwischen den verschiedenen Gewerken besteht aus dem Austausch von Bytefolgen, die bei jeder Änderung händisch neu eingetragen werden müssen.

SPIT

Der SPIT Server kann von dem SPIT Client alle SPIT Typen und SPIT Objekte abfragen, die in diesem angelegt wurden. Der Informationsaustausch geschieht mit sprechenden Beschreibungen. Im Server kann nun jederzeit eine Aktion auf dem Client ausgelöst werden, indem eine Play-Sequenz mit einer Referenz auf ein SPIT Objekt an den Client gesendet wird.

Umgekehrt kann auch der SPIT Server SPIT Typen und SPIT Objekte anlegen, die später im Client mit ausführbaren Aktionen verknüpft werden können. So kann auf beiden Seiten an einem Projekt gearbeitet werden.

Wird eine Verbindung zwischen einem SPIT Client und einem SPIT Server aufgebaut, so können alle SPIT Typen und SPIT Objekte synchronisiert werden. Somit sind Änderungen sehr einfach durchführbar und alle beteiligten Seiten haben immer alle Informationen zur Verfügung.

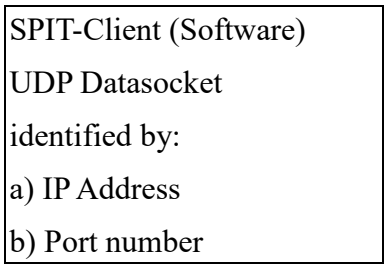
## Client-Software: SPIT Client Struktur

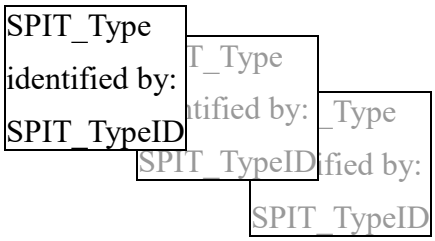
Der SPIT Client legt seine steuerbare Funktionalität über SPIT Typen offen. Diese Funktionalität kann sich pro geladenem Projekt unterscheiden. Die SPIT Typen werden über den SPIT Server auch anderen SPIT Clients zur Verfügung gestellt.

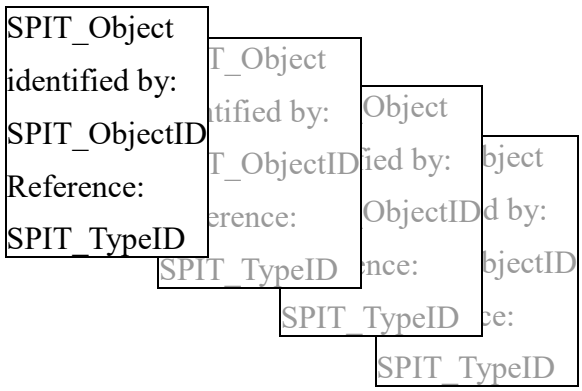
Die Verbindung zu einem SPIT Server wird über das UDP NetzwerkProtokoll ermöglicht, jedes UDP Paket beinhaltet eine SPIT-Nachricht (siehe weiter unten). Theoretisch wäre es auch denkbar die Verbindung über einen anderen Weg aufzubauen (z.B. ArtNet) und die SPIT-Nachrichten befinden sich im ArtNet-Triggerbefehl.

Wenn sich der SPIT Client mit einem SPIT Server verbindet, so tauschen beide ihre SPIT Typen und SPIT Objekte aus.

Unabhängig von der Art der Verbindung, besteht die logische Verbindung zwischen dem SPIT Projekt des Clients und dem korrespondierenden SPIT Projekt des Servers. So kann der SPIT Client durchaus auf einem anderen Rechner gestartet werden und das Client Projekt laden. Mit anderen Worten, der SPIT Client ist austauschbar, da der logische Austausch zwischen den SPIT Projekten geschieht.



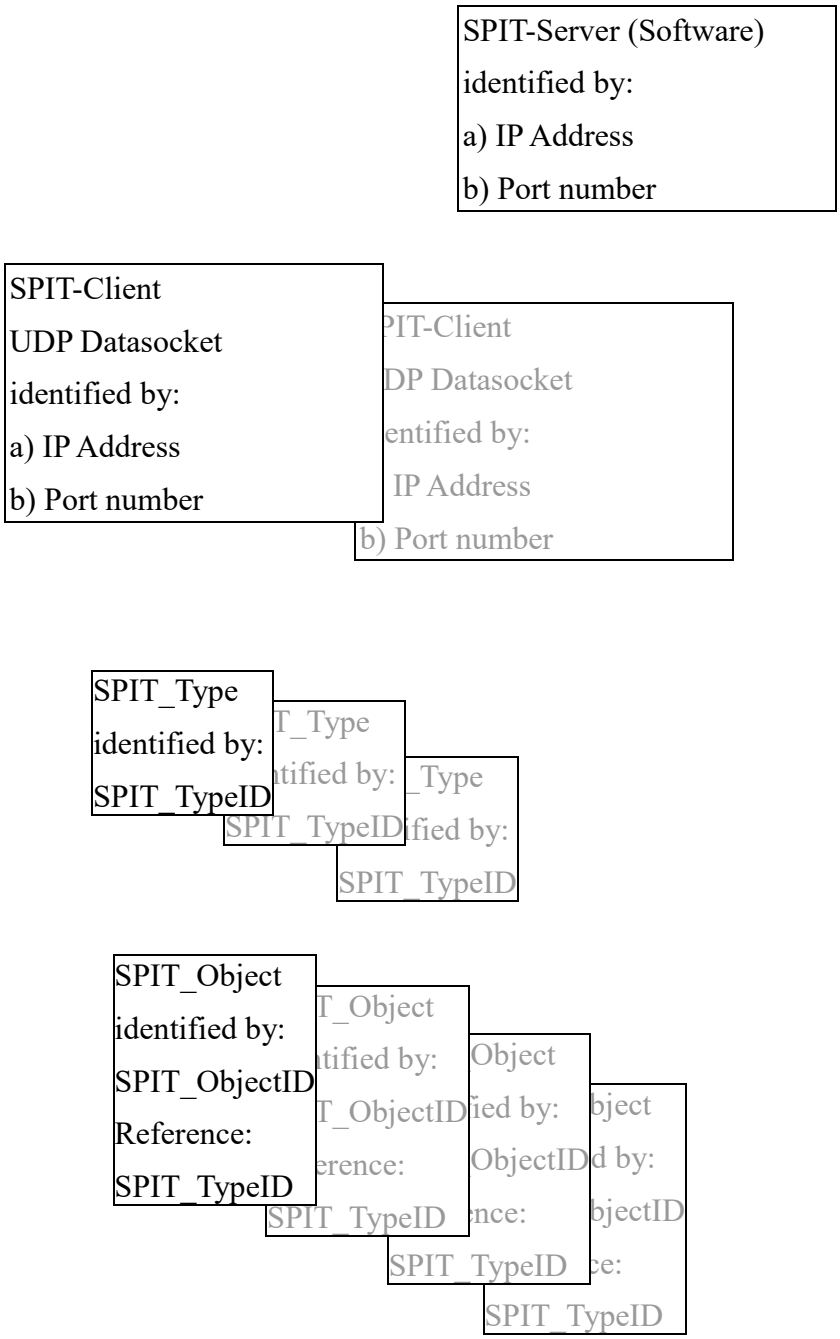

Jeder SPIT Client besitzt nach der Verbindung mit dem SPIT Server seine SPIT Typen / SPIT Objekte und die aller anderen angemeldeten Clients

## Server-Software: SPIT Server Struktur

Wenn sich ein SPIT Client mit dem SPIT Server verbindet, frägt der SPIT Server alle SPIT Typen und alle SPIT Objekte des SPIT Cients ab, diese können synchronisiert und an alle angemeldeten SPIT Clients verteilt werden.

Somit ist es möglich Veränderungen auf dem SPIT Server oder dem SPIT Client vorzunehmen, ohne dass beide verbunden sind.

Es können sich mehrere SPIT Clients mit einem SPIT Server verbinden. Nach der Synchronisation besitzen der SPIT Server und alle SPIT Clients dieselben SPIT Typen und SPIT Objekte.




Der SPIT Server sammelt alle SPIT Typen und SPIT Objekte von allen verbundenen SPIT Clients und gibt diese an alle Clients weiter.

## SPIT Server - Remote Steuerung

Erlaubt der SPIT Server, dass bestimmte Funktionen/Aktionen durch den SPIT Client ausgelöst werden, so übermittelt er den SPIT Clients die SPIT Typen und SPIT Objekte. Diese SPIT Typen und SPIT Objekte sind markiert, so dass der SPIT Client diese als SPIT Server Funktionen erkennt – siehe weiter unten (ReportSPITObject - 'ServerRemote').

Die (Remote)SPIT Objekte kann der SPIT Client in Play-Sequenzen an den SPIT Server senden und damit Aktionen auf dem Server auslösen.

## Übertragungswege

Eine SPIT Nachricht kann direkt über das UDP Netzwerkprotokoll gesendet werden. Dies ist auch der bevorzugte Weg, den Austausch von SPIT Nachrichten zwischen der Server- und der Clientsoftware zu ermöglichen.

Alternative Übertragungswege:

- TCP IP Sockets wären denkbar, aber schwerer zu programmieren.
- Die SPIT Nachrichten überschreiten nicht die Länge von 512 Bytes, somit könnten diese z.B. auch über den ArtNet Triggerbefehl gesendet werden.

## Anwendungsbeispiel

Die logische Verbindung besteht nicht zwischen Server und Client sondern zwischen dem Projekt, das der Client geladen hat, und dem Projekt, das der Server geladen hat – aus der Sammlung von SPIT Typen und SPIT Objekten.

***Beispiel Videosoftware***

Der SPIT Server ist die LiveShowSoftware.

Der SPIT Client ist eine Video-Software "MyVideo".

Im Server (LiveShowSoftware) wird ein Projekt "ServerVideo1" erstellt.

Im Server könnten nun schon SPIT Typen erzeugt und SPIT Objekte platziert werden.

Im Client (MyVideo) wird ein Projekt "ClientVideo1" erstellt.

Auch im Client könnten SPIT Typen und eventuell SPIT Objekte erzeugt werden.

Der Client verbindet sich mit dem Server. Beide tauschen nun Informationen über ihre Projekte (ProjektID/Name, ... - siehe ReportSPITProject) und ihre SPIT Typen und SPIT Objekte aus. Treten Konflikte auf, so können diese über den SPIT Server synchronisiert werden.

Beiden Parteien sind nun alle SPIT Typen und SPIT Objekte bekannt.

Ein SPIT Typ könnte z.B. ein bestimmtes Video oder eine Videosequenz („Hintergrundanimationen“) beschreiben.
Dabei muss nicht festgelegt sein, um welche Videos es sich genau handelt. Allein aus den Namen des SPIT Typen erschließt sich die logische Bedeutung. Um Namen ändern zu können, ohne dass die Zuordnung verloren geht, besitzt jeder SPIT Typ eine eindeutige ID.

Aus diesen SPIT Typen können nun im SPIT Server SPIT Objekte erstellt und zeitlich angeordnet werden. Die SPIT Objekte dürfen einen eigenen Namen bekommen, der eine genauere Spezifikation erlaubt und die SPIT Objekte besitzen Eigenschaften, wie Fade-In/Out Längen,

Intensität und mehr. Aus einem SPIT Typ können mehrere SPIT Objekte erzeugt werden (ein Video kann mehrmals abgespielt werden, oder die Videosequenz kann mehrmals als Hintergrund auftreten).

Besteht eine Verbindung zwischen SPIT Server und SPIT Client, so werden die Änderungen dem SPIT Client sofort mitgeteilt. Umgekehrt können auch in der SPIT Client Software Veränderungen vorgenommen werden, die sofort dem SPIT Server mitgeteilt werden.

Besteht keine Verbindung zwischen SPIT Server und SPIT Client, so entstehen Konflikte, die bei der nächsten Verbindung aufgelöst werden können.

Für jedes SPIT Objekt kann in der Client Software ("MyVideo") eine Aktion definiert werden. In der Videosoftware kann dies z.B. die Zuordnung eines bestimmten Videos und Beamers sein.

Das SPIT Objekt ist als eigentlicher 'Trigger' zu betrachten. Die SPIT Typen beschreiben die Möglichkeiten (auslösbaren Aktionen) für die Clientsoftware.

In der LiveShowSoftware sind die SPIT Objekte in einer Timeline angeordnet. Bewegt sich der Playcursor in ein SPIT Objekt, so wird eine Sequenz PlaySPITStart, PlaySPITObject, … , PlaySPITEnd an MyPyro gesendet und dort die Aktion (Abspielen von Video1, ..) ausgelöst.

# Aufbau von SPIT Nachrichten (Netzwerkprotokoll)

Jede SPIT Nachricht besteht aus einer Bytefolge die einen Header und einen Operationsteil besitzt.

SPIT Nachricht = HEADER + OPPART

Der HEADER besitzt eine feste Länge.

Der jeweilige OPPART besitzt eine feste Länge.

## Parameter

1. Strings werden in ISO 8859-15 vercodet, sie haben eine feste Länge und sind null (0x00) terminiert.
2. Integer werden immer in 4 Bytes (low byte first, little endian) aufgelöst.
3. Long wird immer in 8 Bytes (low byte first, little endian) aufgelöst
4. Boolean werden mit einem Byte (0x00 = false, >0x00 = true) umgesetzt.
5. Eine ID hat immer eine Länge von 41 Bytes (ISO 8859-15) und ist null terminiert.(max. 40 Character + 0x00), somit kann der ID String eine UUID (Universal Unique Identifier) aufnehmen. Eine UUID hat normalerweise 36 Character + eventuell 2 Klammerzeichen. Im Fall von RemoteObjekten kann die ID künstlich um zwei Character verlängert werden (wenn ein SPITObjekt unterschiedliche Remote Befehle besitzt, können mehrere RemoteObjekte angelegt werden, die die Befehlskennung an die ID anhängt haben).
6. Namen haben meist eine Länge von 64 Bytes und sind null terminiert.(max. 63 Character + 0x00)

## Header

| | | | | |
|---|---|---|---|---|
| SpitID | spit | String iso 8859-15 | 7 | ID always "#spit#" + Null (0x00) |
| SpitVersion | | int | 4 | Low byte first (little endian) |
| MessageNumber | | byte | 1 | Numbering the message. When 0xFF is reached, 0x00 is started again. |
| BytesCount | Bytes | int | 4 | Number of bytes of the Header |
| Server ID (Software ID/Name) | UUID | String iso 8859-15 | 41 | Max 40 characters + Null (0x00) |
| Client ID | UUID | String iso 8859-15 | 41 | Max 40 characters + Null (0x00) |
| OPFlag | | byte | 1 | Nummer des OPParts |
| | | | ***99*** | |

**SpitVersion**

Die verwendete SPIT Version

**MessageNumber (aktuell noch nicht verwendet)**

Wird SPIT über das Internet verwendet, so könnten sich UDP-Pakete überholen. Die MessageNumber kann zur Sortierung der Nachrichten verwendet werden.

**BytesCount**

Die Anzahl der Bytes des kompletten Headers

**ServerID**

Die ID des Servers, diese kann der Client für die Identifikation des Serverprogramms verwenden.

Die ServerID wird vom Server gesetzt, wenn er Nachrichten zum Client sendet.

In der Regel verbindet sich ein Client nur mit einem Server, in diesem Fall ist die ServerID redundant.

**ClientID**

Der Server verwendet diese ID zur Identifikation des sendenden Clients. Da sich mehrere Clients mit dem Server verbinden können, muss jeder Client eine eindeutige ID (UUID) erzeugen.

Die ClientID wird vom Client gesetzt, wenn er Nachrichten an den Server sendet.

**OPFlag**

Die Nummer des OPParts – siehe OPPART.

## OPPART

Der OPPart besteht aus der Beschreibung eines SPIT-Befehls. Der OPPART kann aus einem der folgenden Befehle bestehen:

# OP Verbindung

Es gibt für einen SPIT Client eine Möglichkeit alle SPIT Server innerhalb eines Netzwerkes automatisch zu finden – siehe Anhang: SAD ServerAutoDetect.

## Connect (OPFlag = 1)

Client -> Server

| | | | | |
|---|---|---|---|---|
| BytesCount | | int | 4 | Number of bytes of the OPPart |
| Client ComputerName | | String iso 8859-15 | 64 | Max 63 characters + Null (0x00) |
| Client Name | | String iso 8859-15 | 64 | Max 63 characters + Null (0x00) |
| User Login Name | | String iso 8859-15 | 64 | Max 63 characters + Null (0x00) |
| User Login Password | | String iso 8859-15 | 64 | Max 63 characters + Null (0x00) |
| ServerIP | | String iso 8859-15 | 40 | IP address over which the client has reached the server max. IP 6 address string= 39 chars + Null (0x00) |
| ServerPort | | int | 4 | port number over which the |

| | | | | |
|---|---|---|---|---|
| | | | | client has reached the server |
| | | | ***304*** | |

**Client ComputerName**

Der Name des Client Computers (HostName). Diese Information nutzt der Server für darstellerische Zwecke.

**Client Name**

Der Name der Client Software. Diese Information nutzt der Server für darstellerische Zwecke.

**User Login Name**

Benutzername mit dem sich der Client beim Server einloggt.

Das Server Programm kann vor unqualifizierter Benutzung geschützt werden, indem ein Benutzer Name und ein ein Benutzer Passwort abgeglichen wird.

Soll der Server offen (ungeschützt) sein, so wird dieser Parameter ignoriert.

**User Login Password**

Benutzerpasswort mit dem sich der Client einloggt

Soll der Server offen (ungeschützt) sein, so wird dieser Parameter ignoriert.

**ServerIP**

Die IP Adresse des Servers, über die der Client den Server erreicht hat.

Wurde ServerAutodetect verwendet, so kennt der Client anhand des empfangenen UDP-Paketes die IP Adresse über die er den Server erreichen kann.

Diese Information nutzt der Server für darstellerische Zwecke.

**ServerPort**

Portnummer über die der Client das Serverprogramm erreicht hat.

Diese Information nutzt der Server für darstellerische Zwecke.

<u>Verhalten:</u>

Nachdem die Nachricht gesendet wurde:

Client:

1. Warte auf ConnectAnswer des SPIT Servers

Nachdem die Nachricht empfangen wurde:

Server:

1. Sendet ConnectAnswer zum SPIT Client. Der Connected-Parameter kann auf true oder false gesetzt werden:
   true = der SPIT Server akzeptiert die Verbindung
   false = der SPIT Server weist die Verbindung zurück. (evtl. Sind die Login Parameter falsch, oder ....)
2. Wenn die Verbindung akzeptiert wird, sendet der SPIT Server ReportSPITProject mit den Projektdaten des SPIT Servers.

(das ist nicht ein SPIT Projekt , sondern eine Information über das Projekt des Servers, die dem SPIT Client zur Identifikation des SPIT Servers dienen kann)

## ConnectAnswer (OPFlag = 2)

Übertragungsrichtung: Server -> Client

| | | | | |
|---|---|---|---|---|
| BytesCount | Bytes | int | 4 | Number of bytes of the OP part |
| Server ComputerName | | String iso 8859-15 | 64 | Max 63 characters + Null (0x00) |
| Server Name | | String iso 8859-15 | 64 | Max 63 characters + Null (0x00) |
| Server Step Time | ms | int | 4 | Time period the server will send play sequences |
| Connected | | boolean | 1 | 0x00 = false<br>> 0x00 = true |
| | | | ***137*** | |

**Server Computername**

Computername des Servers. Diese Information nutzt der Client für darstellerische Zwecke.

**Server Name**

Name der Server Sofware. Diese Information nutzt der Client für darstellerische Zwecke.

**Server Step Time:**

Dies ist ein informeller Wert, in welchen Zeitabschnitten Abspielbefehle (PlaySPITStart,..., PlaySPITEnd) erfolgen können.

Ein Server könnte mehrere Player besitzen (Player für das Abspielen einer Timeline, Player für das Abspielen eines Jingles, einen Player für das Bewegen eines Faders, ..). Ein Player kann softwareseitig aus einem Workerthread bestehen oder einfach ein fiktiver Player sein, der nur bei bestimmte Events (Faderbewegung, Knopfdruck,..) in Kraft tritt.

Wenn der Server einen Player besitzt,, der z.B. eine Timeline abspielt, so kann hier die Zeit in Millisekunden zwischen zwei Abspielfolgen ( siehe PlaySPITStart, ... , PlaySPITEnd) angegeben werden. Dies entspricht sampling rate des Servers. Solch ein Player kann auch als MAIN-Player deklariert werden, der einen Timecode übermitteln kann.

Zu Beachten ist, dass hier Unterbrechungen und Sprünge im Ablauf der Timeline erlaubt sind. Es ist auch möglich, dass die Step Time nicht kontinuierlich eingehalten wird.

Besitzt der Server keinen Player, der eine zeitlich feste Folge von Abspielbefehlen liefert, kann hier 0 als Wert eingetragen werden.

Ein Player existiert im SPITProtokoll nicht als eigenständiges Objekt, es wird nur bei PlaySPITStart und PlaySPITEnd eine freie PlayerID übermittelt, um die Abspielfolgen unterschiedlicher 'Player' auseinanderhalten zu können.

**Connected**

Gibt an, ob der Server die Verbindung akzeptiert hat oder nicht.

- 0x00 (false) der Server hat die Verbindung abgelehnt

- >0x00 (true) der Server hat die Verbindung akzeptiert

Verhalten

Nach dem Senden der Nachricht:

Server:

1. Sende ReportSPITProject zum Client, wenn die Verbindung akzeptiert wurde (siehe Connect)

Nach dem Empfang der Nachricht:

Client:

1. Client wartet auf  ReportSPITProject

## Disconnect (OPFlag = 3)

Client->Server

Server->Client

| BytesCount | Bytes | int | 4 | Number of bytes of the OP part |
|---|---|---|---|---|
| | | | ***4*** | |

Verhalten:

Nach dem Senden der Nachricht:

Server and Client:

Beenden des UDPSockets

Nach dem Empfang der Nachricht:

Server and Client:

Beenden des korrespondierenden UDPSockets.

## Watchdog (OPFlag = 8)

Client->Server

Server->Client

Um zu prüfen ob die Gegenseite noch verbunden ist können sowohl der Server, als auch der Client Watchdog Nachrichten schicken. Die Gegenseite sollte sofort mit einer WatchdogAnswer Nachricht antworten. Kommt nach einer festgelegten Zeit (Watchdog timeout) keine WatchdogAnswer, so kann davon ausgegangen werden, dass die Gegenseite nicht mehr erreichbar ist.

Der Watchdog Timeout kann in jeder Software individuell festgelegt werden, da die Regel besteht, dass immer sofort eine WatchdogAnswer gesendet wird. Zu beachten ist, dass die Gegenseite mit der Bearbeitung eines anderen Befehls beschäftigt sein kann.

Als Standard Timeout werden in SPIT 1000 ms festgelegt.

| | | | | |
|---|---|---|---|---|
| BytesCount | Bytes | int | 4 | Number of bytes of the OP part |
| | | | ***4*** | |

Nach dem Empfang der Nachricht:

Server und Client:

1. Sende sofort eine WatchdogAnswer Nachricht

## WatchdogAnswer (OPFlag = 9)

Client->Server

Server->Client

Die ist die Antwort, die direkt auf eine Watchdog Nachricht gesendet werden sollte.

| | | | | |
|---|---|---|---|---|
| BytesCount | Bytes | int | 4 | Number of bytes of the OP part |
| | | | ***4*** | |

Nach dem Empfang der Nachricht

Server und Client:

1. Setze die Watchdog Wartezeit zurück

Es kann auch jede Reaktion (Empfang einer beliebigen Nachricht) der Gegenseite dazu genutzt werden die Watchdog Wartezeit zurückzusetzen.

# OP Projekt

## ReportSPITProject (OPFlag = 11)

| | | | | |
|---|---|---|---|---|
| BytesCount | Bytes count of this OPPart | int | 4 | Number of bytes of the OP part<br>low byte first |
| ProjectID (Dateiname des Projektes) | UUID | String iso 8859-15 | 41 | Max 40 characters + Null (0x00) |
| ProjectName | | String iso 8859-15 | 64 | Max 63 characters + Null (0x00) |
| FramesPerSecond | Frames per second | int | 4 | Low byte first<br>all time information are set in frames<br>This can be used to convert between time and frames |

| | | | | |
|---|---|---|---|---|
| | | | ***113*** | |

**ProjectID**

Es wird vielfach vorkommen, dass ein altes Projekt kopiert wird um daraus ein neues Projekt zu erstellen  Genauso kann aber auch ein Projekt unter einem anderen Namen abgespeichert werden. Im ersten Fall entsteht ein neues Projekt, die ProjectID müsste sich ändern. Im zweiten Fall bleibt das Projekt bestehen, die ProjectID darf sich nicht ändern. Softwareseitig können die beiden Fälle nicht unterschieden werden. Daher wird empfohlen, den Dateinamen eines Projektes als ID zu verwenden.

**ProjectName**

Ein sprechender Name für das jeweilige Projekt. In vielen Fällen wird dieser Name auch für das Abspeichern von Projekten verwendet werden.

**FramePerSecond**

Alle zeitlichen Angaben werden in der Einheit Frames übermittelt. Hier wird der Umrechnungsfaktor Frames pro Sekunde angegeben.

Zu beachten ist, dass in der Regel der Server diesen Wert bestimmt und der Client unter Umständen alle zeitlichen Werte neu berechnen muss.

Verhalten:

Nach dem Empfang der Nachricht

Client:

1. Sendet RequestSPITProject
2. Sendet ReportSPITProject

Server:

1. Sendet RequestProjectTranfer

## CloseSPITProject (OPFlag = 13)

Server->Client

Client -> Server

| | | | | |
|---|---|---|---|---|
| BytesCount | Bytes | int | 4 | Number of bytes of the OP part |
| Project ID | UUID | String iso 8859-15 | 41 | Max 40 characters + Null (0x00) |
| | | | 45 | |

Wenn entweder der Client oder der Server ein Projekt schließt, so muss die Gegenseite darüber informiert werden.

Das Schließen eines Projektes bedeutet nicht, dass auf der Gegenseite das Projekt gelöscht wird!

CloseSPITProject kann auftreten, wenn auf der Gegenseite:

1. ein neues Projekt erstellt wird
2. ein anderes Projekt geladen wird
3. das Programm beendet wird

Verhalten:

Beim Empfang dieser Nachricht:

Client:

1. Setze für alle SPIT Typen und SPIT Objekte das 'Confirmed' Flag auf false.

Server:

1. entferne aus allen SPIT Objekten den Client als Besitzer

Wenn ein Projekt geladen oder ein leeres Projekt erstellt ist:
Client:
1. Sendet ReportSPITProject
2. Sendet RequestProjectTransfer

Server:
1. Sendet ReportSPITProject

## RequestProjectTransfer (OPFlag = 15)

Server->Client

Client->Server

| | | | | |
|---|---|---|---|---|
| BytesCount | Bytes | int | 4 | Number of bytes of the OP part |
| | | | 4 | |

Um von der Gegenseite die SPIT Typen und SPIT Objekte abfragen zu können muss der Server oder der Client diese Nachricht senden.

Nachdem Senden der Nachricht:

Server und Client:

1. Warten auf die Sequenz StartProjectTransfer, ...., EndProjectTransfer

Nach dem Empfang der Nachricht:

1. Senden der Sequenz:
   a) StartProjectTransfer
   b) für jeden SPIT Typ eine ReportSPITType Nachricht ...
   c) für jedes SPIT Objekt eine ReportSPITObject Nachricht ...
   d) EndProjectTransfer

## StartProjectTransfer (OPFlag = 16)

Client->Server

Server->Client

| | | | | |
|---|---|---|---|---|
| BytesCount | Bytes | int | 4 | Number of bytes of the OP part |

| | | | | |
|---|---|---|---|---|
| | | | 4 | |

Gibt der Gegenseite bekannt, dass nun alle SPIT Typen und alle SPIT Objekte übertragen werden.

### EndProjectTransfer (OPFlag = 17)

Client->Server

Server->Client

| | | | | |
|---|---|---|---|---|
| BytesCount | Bytes | int | 4 | Number of bytes of the OP part |
| | | | 4 | |

Gibt der Gegenseite bekannt, dass die komplette Übertragung der SPIT Typen und SPIT Objekte beendet ist.

## SPIT Typ

Der SPIT Typ gibt an, um welche Art Trigger/Aktion es sich handelt.
In der Videosoftware könnte dies z.B. Der Name eines Videos oder eine Folge von Hintergrundanimationen sein.
In einer Steuerungssoftware könnte dies z.B. der Befehl 'nächste Szene', oder die Angabe einer bestimmten DMXLampe sein.
Bei einem Mischpult könnte dies der Regler eines bestimmten Eingangs sein.

Ein SPIT Typ kann im Show-Ablauf mehrmals verwendet werden. Der SPIT Typ ist noch kein vollständiger Trigger, er klassifiziert nur die auszulösende Aktion.
Der SPIT Typ  hat noch keine Parameter für Startzeit, Länge, FadeIn, FadeOut,...

Ein auslösbarer Trigger ist das SPIT Objekt, das einen SPIT Typ als Referenz beinhaltet – siehe SPIT Objekt. Ein SPIT Objekt besitzt weitere Parameter, die für seine Darstellung und die Ausführung einer Aktion wichtig sind.

### ReportSPITType ( OPFlag = 21)

| | | | | |
|---|---|---|---|---|
| BytesCount | Bytes | int | 4 | Number of bytes of the OP part |
| Confirmed | Boolean | | 1 | 0x00 = false<br>> 0x00 = true |
| SPIT_TypeID | UUID | String iso 8859-15 | 41 | Max 40 characters + Null (0x00) |
| Fixed | Boolean | | 1 | 0x00 = false<br>>= 0x01 = true |
| ServerRemote | Boolean | byte | 1 | >= 0x01 = true<br>the object runs on the server<br>the client can start the object (Remote) |

| | | | | |
|---|---|---|---|---|
| | | | | 0x00 = false<br>the object runs on the client<br>The server can start the object |
| ServerProjectInpended | Boolean | byte | 1 | 0x00 = false<br>> 0x00 = true |
| GroupName | String | String iso 8859-15 | 39 | Max 38 characters + Null (0x00)<br>Used to categorize the SPIT Types |
| Name | String | String iso 8859-15 | 64 | Max 63 characters + Null (0x00) |
| Default Delay | Frames | long | 8 | Low byte first |
| Default Duration | Frames | long | 8 | Low byte first |
| | | | ***168*** | |

**Confirmed**
Wird entweder vom Server oder vom Client ein SPIT Typ erzeugt oder geändert, so sollte ein ReportSPITType an die Gegenseite gesendet werden, hierbei wird in der Regel das Confirmed-Flag auf 'false' gesetzt. Die Gegenseite antwortet mit derselben Nachricht, aber hier ist das Confirmed-Flag auf 'true' gesetzt. Somit ist klar, dass die Gegenseite die Nachricht empfangen und umgesetzt hat.

Erlaubt die Gegenseite das Editieren von SPIT Typen nicht, so sendet diese ReportSPITType mit den alten Parametern und das Confirmed-Flag ist auf 'true' gesetzt.

Wird umgekehrt der Gegenseite nicht erlaubt den SPIT Typ zu ändern, so wird schon beim Senden von ReportSPIT_Type das Confirmed-Flag auf true gesetzt.

Siehe weiter unten Parameter Fixed

Somit gilt folgende einfache Regel:

*Immer wenn eine ReportSPITType Nachricht empfangen wird und das Confirmed-Flag auf 'true' gesetzt ist, werden die Parameter übernommen.*

**SPIT_TypeID**

Eine echte eindeutige ID für den SPIT Typ. Anhand dieser ID kann im SPIT Objekt eine Referenz erstellt werden und der SPIT Typ wird auf der Gegenseite erkannt.

**Fixed**

Falls ein Client feste unveränderbare SPIT Typen besitzt, so muss hier true (0x01) eingetragen werden.

Falls ein SPIT Typ geändert oder gelöscht werden darf, wird hier false (0x00) eingetragen.

**ServerRemote**

In der Regel gehören die SPIT Typen zu einem Client, sie beschreiben die möglichen Aktionen eines Clients. Aber auch der Server könnte dem Client Aktionen zur Verfügung stellen, um eventuell eine Fernbedienung durch den Client zu erlauben.

ServerRemote gibt an, wer die Aktion zur Verfügung stellt

**0x00 (false)**: Der Client besitzt den SPIT Typ, er kann vom Server über ein SPIT Objekt ausgelöst werden
>= **0x01 (true)**: Der Server besitzt den SPIT Typ, er kann vom Client über ein SPIT Objekt ausgelöst werden (Remote).

**ServerProjectIndepended**

Normalerweise werden die SPIT Typen innerhalb eines Projektes erzeugt. Im Fall von Remote SPIT Typen (und alle SPIT Objekte, die auf diese SPIT Typen referenzieren), kann es aber auch Typen/Objekte geben, die unabhängig von einem Projekt sind, z.B. Hauptlautstärke, Sprung zur nächsten Szene, ….

0x00 false = die SPIT Typ ist abhängig vom geladenen Projekt des Servers. Remote SPIT Typ und dessen SPIT Objeke sollten gelöscht werden, wenn der Server das Projekt schließt.

0x01 true = die SPIT Typ ist unabhängig vom geladenen Projekt des Servers. Remote SPIT Typ und dessen SPIT Objekte können erhalten bleiben, wenn der Server das Projekt schließt.

**GroupName**

Der Gruppenname kann für die Klassifizierung/Gruppierung von SPIT Typen verwendet werden.

**Name**

Ein sprechender Name für den Typ, die auslösbare Aktion.

**Default Delay**

In manchen Anwendungen ist das Starten einer Aktion nicht immer gleichbedeutend mit der sichtbaren Wirkung. Dieser Parameter gibt die Verzögerung in Frames zwischen dem Start einer Aktion und der sichtbaren Auswirkung der Aktion an. Der Wert ist ein Defaultwert, er kann im SPIT Objekt überschrieben werden.

In einer Videosoftware könnte dies die Zeit sein, bis ein Video geladen ist.

**Default Duration**

Manche Aktionen haben eine zeitliche Länge. Dieser Wert wird in Frames angegeben und kann im SPIT Objekt überschrieben werden.

Ein Video kann z.B. eine feste Länge besitzen.

## RemoveSPITType ( OPFlag = 23)

Server -> Client

Client -> Server

| | | | | |
|---|---|---|---|---|
| BytesCount | Bytes | int | 4 | Number of bytes of the OP part |
| SPIT_TypeID | UUID | String iso 8859-15 | 41 | Max 40 characters + Null (0x00) |
| | | | ***45*** | |

Wenn ein SPIT Typ gelöscht wird, so muss dies der Gegenseite mitgeteilt werden.

Wenn die Gegenseite das Editieren von SPIT Typen nicht erlaubt oder der SPIT Typ als fixiert definiert ist, so sollte die Gegenseite diese Nachricht nicht ausführen und ein ReportSPITType als Antwort auf diese Nachricht senden, damit der SPIT Typ wieder angelegt wird.

Der Server könnte eine UNDO Funktionalität besitzen, daher sollte der Client seine Einstellungen für den SPIT Typ nicht komplett entfernen, sondern diese mit der zugehörigen SPIT Typ ID zwischenspeichern. Sollte der SPIT Typ auf dem Server wieder hergestellt werden und der Server eine ReportSPITType Nachricht sendet, so kann beim Client die entsprechende Einstellung aus dem Zwischenspeicher wieder reproduziert werden. Wenn der Client ein gespeichertes Projekt lädt, kann der komplette Zwischenspeicher geleert werden.

# SPIT Objekt

Ein SPIT Objekt besitzt eine Referenz zu einem SPIT Typ (SPIT_TypeID) und weitere Parameter, die für die Ausführung einer Aktion wichtig sein können. Inwieweit diese Parameter verwendet oder interpretiert werden, ist der jeweiligen Software überlassen.

Ein SPIT Objekt kann es nur einmal geben. Es kann aber mehrere SPIT Objekte geben, die eine Referenz auf denselben SPIT Typ besitzen.

Um Aktionen auf der Gegenseite auszulösen wird an die Gegenseite eine Sequenz gesendet:

1. PlaySPITStart
2. eine Folge von PlaySPITObjekten, jeweils mit einer Referenz auf ein SPIT Objekt ...
3. PlaySPITEnd

Dies wird weiter unten genauer erklärt - PlaySequenz.

Für das Auslösen einer Aktion wird also nicht das SPIT Objekt selbst gesendet, sondern eine PlaySPITObject Nachricht mit der Referenz auf ein SPIT Objekt.

Da der Gegenseite das SPIT Objekt mit seinen Parametern bekannt ist, ist gewährleistet, dass alle Informationen für die Ausführung der Aktion vorhanden sind.

## ReportSPITObject (OPFlag = 31)

Server -> Client

Client -> Server

| | | | | |
|---|---|---|---|---|
| BytesCount | Bytes | int | 4 | Number of bytes of the OP part |
| Confirmed | Boolean | | 1 | 0x00 = false<br>> 0x00 = true |
| SPIT_ObjectID | UUID | String iso 8859-15 | 41 | Max 40 characters + Null (0x00) |
| Fixed | Boolean | | 1 | 0x00 = false<br>> 0x00 = true |
| ServerRemote | Boolean | byte | 1 | This is redundand to ServerRemote in ReportSPITType |
| Name | | String iso 8859-15 | 64 | Max 63 characters + Null (0x00) |
| SPIT_TypeID | UUID | | 41 | Max 40 characters + Null (0x00) |
| ServerParams | String | String iso 8859-15 | 64 | Max 63 characters + Null (0x00) |

| | | | | |
|---|---|---|---|---|
| | | | | should not be changed by opposite |
| FrameStart | Frames | long | 8 | |
| FrameLength | Frames | long | 8 | |
| FrameFadeInLength | Frames | long | 8 | |
| FrameFadeOutLength | Frames | long | 8 | |
| Valuefactor (Volume/Intensity/...) | ppm | int | 4 | Factor is multiplied by the 'Value' from PlaySPITObject to get the actual value of the object |
| ValueUse | Byte | byte | 1 | 0x00 = Value is discarded<br>0x01 = Value in Range (0 – 1000000 ppm)<br>0x02 = Value can be overridden |
| Track/Priority | Number | int | 4 | Layer level |
| Remark | | String iso 8859-15 | 64 | Max 63 characters + Null (0x00)<br>HTML tags can be used e.g.<br>new Line <br> |
| Delay | Frames | long | 8 | |
| Duration | Frames | long | 8 | |
| ActionAssigned | Boolean | | 1 | 0x00 = false<br>> 0x00 = true |
| | | | ***339*** | |

**Confirmed**
Wird entweder vom Server oder vom Client ein SPIT Objekt erzeugt oder geändert, so sollte ein ReportSPITObjekt an die Gegenseite gesendet werden, hierbei wird in der Regel das Confirmed-Flag auf 'false' gesetzt. Die Gegenseite antwortet mit derselben Nachricht, aber hier ist das Confirmed-Flag auf 'true' gesetzt. Somit ist klar, dass die Gegenseite die Nachricht empfangen und umgesetzt hat.

Erlaubt die Gegenseite das Editieren von SPIT Objekten nicht, so sendet diese ReportSPITObjekt mit den alten Parametern und das Confirmed-Flag ist auf 'true' gesetzt.

Wird umgekehrt der Gegenseite nicht erlaubt den SPIT Objekt zu ändern, so wird schon beim Senden von ReportSPIT_Object das Confirmed-Flag auf true gesetzt.

Somit gilt folgende einfache Regel:

*Immer wenn eine ReportSPITObject Nachricht empfangen wird und das Confirmed-Flag auf 'true' gesetzt ist, werden die Parameter übernommen.*

**SPIT_ObjectID**

Eine echte eindeutige ID für das SPIT Objekt. Anhand dieser ID wird das SPIT Objekt auf der Gegenseite erkannt.

**Fixed**
Ist Fixed auf 'true' gesetzt, so ist das SPIT Objekt fixiert, es kann von der Gegenseite nicht verändert werden, auch wenn der Gegenseite das Editieren von SPIT Objekten erlaubt ist.

**ServerRemote**

Dies ist eigentlich durch den SPIT Typ schon festgelegt und wird hier redundant übernommen, um eine etwas erleichtere Programmierung eines Clients zu ermöglichen..

In der Regel lösen die SPIT Objekte Aktionen auf dem SPIT Client aus.

Aber auch der SPIT Server könnte SPIT Objekte zur Verfügung stellen um eine Fernbedienung durch den Client zu ermöglichen. In diesem Fall sendet der Client eine Play-Sequenz und löst Aktionen auf dem Server aus (Remote).

ServerRemote gibt an, wer das Objekt/die Aktion zur Verfügung stellt

**0x00 (false)**: Auf dem Client wird das SPIT Objekt / Aktion ausgelöst, Der Server löst diese Aktion über eine Play-Sequenz aus.
**0X01 (true)**: Auf dem Server wird das SPIT Objekt / Aktion ausgelöst, Der Client löst diese Aktion über eine Play-Sequenz aus (Remote).

**ServerParams**

Hier können vom Server zusätzliche Paramter als String mitgeben werden, die er für die Verarbeitung/Darstellung des SPIT Objektes benötigt werden. Die Clients verändert diese NICHT! Die ServerParams solten von den Clients abgespeichert und beim Senden von ReportSPITObject mitgeliefert werden.

**FrameStart**

Der Startzeitpunkt eines SPIT Objektes in Frames. Verwendet der Server das SPIT Objekt in einer Timeline/festen Abfolge, so macht dieser Parameter Sinn. In allen anderen Fällen, wenn z.B. das SPIT Objekt nur ein Fader-Ereignis ist, kann hier 0 eingetragen werden.

**FrameLength**

Die Länge eines SPIT Objektes in Frames. Ein SPIT Objekt kann in einer gewissen Länge dargestellt werden. Oft wird dies der Dauer der Aktion entsprechen (siehe weiter unten – Duration), dies ist aber nicht zwingend erforderlich.

**FrameFadeInLength**

SPIT Objekte dürfen ein- und ausgefadet werden. Besteht die Aktion z.B. aus der Wiedergabe eines Films, so kann dieser in seiner Helligkeit ein- und ausgeblendet werden.

Zeit für das Einblenden in Frames,

**FrameFadeOutLength**

Zeit für das Ausblenden in Frames – siehe FadeInTime

**ValueFactor**

Hier kann ein Faktor, in PartsPerMillion (ppm), angegeben werden, mit dem der 'Value' Wert aus PlaySPITObject multipliziert werden kann um den aktuellen Wert des Objektes zu bestimmen.

'ValueFactor' darf 1000000 übersteigen, er könnte z.B. für das grundsätzliche Anheben oder Absenken der Lautstärke / Helligkeit einer Aktion verwendet werden.

Der Wert ist nicht zu verwechseln mit dem Value-Wert der PlaySPITObject Nachricht. Value gibt beim Abspielen eine Objektes den jeweils aktuellen Wert an, z.B. wenn ein Objekt ein- oder

ausgefadet wird.

Berechnung des Wertes eines Objektes:

Wert = 'Value' * ((float)'ValueFactor' / 1000000) – siehe PlaySPITObject.

**ValueUse**

Manche SPIT Clients verwenden die Wertangabe für das SPIT Objekt nicht. Ein Sprung zu einer Videosequenz wird keine Ein- oder Ausblendwerte berücksichtigen. Eine Software, die Animationen ein- oder ausblenden kann, wird die Wertangabe verwenden.

Grundsätzlich sendet der Server die Wertangabe mit und der Client entscheidet ob er die Wertangabe verwendet oder nicht.

ValueUse ist vor Allem bei den Remote Aktionen des SPIT Servers von Bedeutung. Hier teilt der Server dem SPIT Client mit, ob das SPIT Objekt nur ein einfacher Steuerbefehl (z.B. nächste Szene) ist, oder ob es sich um einen Befehl handelt, der Werte setzt (z.B. Faderstellung einer Lampe).

- ValueUse = 0x00 der Wert wird nicht verwendet
  Der SPIT Client kann das SPIT Objekt als Button darstellen.
- ValueUse = 0x01 der Wert wird in den Grenzen 0-100% (0 – 1000000 ppm) verwendet
  Der SPIT Client kann das SPIT Objekt als Regler / Fader darstellen.
- ValueUse = 0x02 der Wert kann übersteuert werden
  Der SPIT Client kann das SPIT Objekt als Scrollrad darstellen und in PlaySPITObject sollte 'ValueFlag' mit (0x01 increase oder 0x02 decrease) und 'Value' mit der Werteveränderung gesetzt sein.
  Es müssen die Werteveränderungen und nicht der absolute Wert gesendet werden.

**Track/Priority**

Falls der Server das SPIT Objekt in einer mehrspurigen Timeline darstellt, so könnte hier die Spurnummer eingetragen werden. Oder die Aktion besteht aus der Wiedergabe von Bildmaterial, das auf einer bestimmten Layerebene (Z-Ebene) dargestellt werden soll.

**Remark**

Hier können weitere Informationen/Anmerkungen übermittelt werden.

**Delay**

In manchen Anwendungen ist das Starten einer Aktion nicht immer gleichbedeutend mit der sichtbaren Wirkung. Dieser Parameter gibt die Verzögerung in Frames zwischen dem Start einer Aktion und der sichtbaren Auswirkung der Aktion an. Der Wert kann den Defaultwert des SPIT Typ überschreiben.

In einer Videosoftware könnte dies die Zeit sein, bis das im Client zugewiesene Video geladen ist.

Delay hat erstmal keine direjten Auswirkungen auf den Server, der Wert wird nur dargestellt.

**Duration**

Manche Aktionen haben eine zeitliche Länge. Dieser Wert wird in Frames angegeben und überschreibt den Defaultwert des SPIT Typ.

Ein Video kann z.B. eine feste Länge besitzen.

Duration hat erstmal keine direjten Auswirkungen auf den Server, der Wert wird nur dargestellt.

**ActionAssigned**

Hat ein Client einem SPIT Objekt eine Aktion zugewiesen, so wird ActionAssigned auf true gesetzt

(>= 0x01).

Der Server kann diese Information verwenden, um zu verhindern, dass ein anderer Client, dieses SPIT Objekt oder den zugehörigen SPIT_Typ löscht.

ActionAssigned darf nur auf dem Server ausgewertet werden. Der Client muss dies immer überschreiben, je nachdem, ob die Aktion bei ihm zugewiesen ist oder nicht.

Wird eine Aktion zugewiesen oder entfernt während der Client mit dem Server verbunden ist, so muss der Client eine ReportSPITObject Nachricht an den Server senden.

ACHTUNG: der Server setzt 'ActionAssigned' bei ReportSPITObject immer auf 0x00.

### RemovedSPITObject ( OPFlag = 33)

Server -> Client

Client -> Server

| | | | | |
|---|---|---|---|---|
| BytesCount | Bytes | int | 4 | Number of bytes of the OP part |
| SPIT_ObjectID | UUID | String iso 8859-15 | 41 | Max 40 characters + Null (0x00) |
| | | | ***45*** | |

Wenn ein SPIT Objekt gelöscht wird, so muss dies der Gegenseite mitgeteilt werden.

Wenn die Gegenseite das Editieren von SPIT Objekten nicht erlaubt oder das SPIT Objekt als fixiert definiert ist, so sollte die Gegenseite diese Nachricht nicht ausführen und ein ReportSPITObject als Antwort auf diese Nachricht senden, damit das SPIT Objekt wieder angelegt werden kann.

Der Server könnte eine UNDO Funktionalität besitzen, daher sollte der Client seine Einstellungen für das SPIT Objekt nicht komplett entfernen, sondern diese mit der zugehörigen SPIT Objekt ID zwischenspeichern. Sollte das SPIT Objekt auf dem Server wieder hergestellt werden und der Server eine ReportSPITObject Nachricht sendet, so kann beim Client die entsprechende Einstellung aus dem Zwischenspeicher wieder reproduziert werden. Wenn der Client ein gespeichertes Projekt lädt, kann der komplette Zwischenspeicher geleert werden.

## PlaySPIT-Sequenz - Das Auslösen von Aktionen

Mit den SPIT Typen und SPIT Objekten sind Aktionen mit ihren wichtigsten Abspielparametern klassifiziert.

Um eine Aktion auf der Gegenseite auszulösen wird folgende Sequenz gesendet:

1. PlaySPITStart
2. eine Folge von PlaySPITObject Nachrichten, jeweils mit einer Referenz auf ein SPIT Objekt
3. PlaySPITEnd

Alle SPIT Objekte, auf die in den PlaySPITObject Nachrichten referenziert wird, spielen ihre zugewiesenen Aktionen ab - der Status des SPIT Objektes wird verändert oder es wird einfach eine Aktion ausgelöst.

Es kann durchaus sein, dass zwischen PlaySPITStart und PlaySPITEnd keine PlaySPITObject Nachrichten enthalten sind.

Es wird in einem Server oder auch Client mehrere Instanzen geben, die für das Auslösen einer Aktion auf der Gegenseite zuständig sind, diese Instanzen werden hier Player genannt. Player können konkrete Objekte/Threads in einer Software sein oder sie können nur fiktiv existieren, z.B. wenn Ereignisse von einer Eventqueue bearbeitet werden – siehe Player Konzept weiter unten.

In der LiveShow-Software gibt es z.B. jeweils einen Player für die Timeline und jedes Jingle.

## PlaySPITStart (OPFlag = 37)

Server -> Client
Client -> Server (Remote)

| | | | | |
|---|---|---|---|---|
| BytesCount | Bytes | int | 4 | Number of bytes of the OP part |
| PlayerID | UUID | String iso 8859-15 | 41 | Max 40 characters + Null (0x00) |
| PlayerName | String | String iso 8859-15 | 64 | Max 63 characters + Null (0x00) |
| PlayerType | | byte | 1 | 0x02 = main continious<br>the player is the main player – the 'Time' can be used for time code<br>0x01 = additional continious<br>the player is an addtional continious player<br>0x00 = additional discrete<br>for events like fader movement, button click, ..<br>>0x02 reserved |
| PlayerIsRunning | | Boolean | 1 | 0x00 = false<br>the playcursor maybe only set once<br>> 0x00 = true<br>the playcursor is running |
| Time | Frames | long | 8 | Time inside of the project in frames |
| | | | ***119*** | |

### PlayerID

Die ID des Players - siehe Player Konzept weiter unten.

Es kann mehrere Player geben (einer für die Timeline, einer für Jingles, einer für Fader Bewegungen,..) und diese können gleichzeitig arbeiten. Dies bedeutet, dass Sequenzen (PlaySPITStart, PlaySPITObject, ..., PlaySPITEnd) durchmischt ankommen können.
Jeder Player besitzt eine eigene ID besitzt, so können die PlaySPITObject Nachrichten zugeordnet werden.

### PlayerName

Der Name des Players, dieser kann für Darstellungen und einfacher Fehlersuche verwendet werden.

**PlayerType**

Der Typ des Players - siehe Player Konzept weiter unten.

1. 0x02= Main Player
   der Player läuft kontinuierlich und liefert die zeitliche Position im Gesamtablauf.
   Dies wäre z.B. ein Player einer Timeline.
   -> der Time Parameter kann als TimeCode verwendet werden.
2. 0x01 = zusätzlicher kontinuierlicher Player
   Der Player läuft zwar kontinuierlich, aber der Start des Players kann irgendwann im Ablauf beginnen. Dies wäre ein Player, der z.B. Jingles abspielt.
3. 0x00 = zusätzlicher diskreter Player
   Der Player tritt nur bei bestimmten Ereignissen (z.B. bei Faderbewegungen, Mausklicks, ...) auf.

In der LiveShow-Software gibt einen Main-Player (0x02) für die Timeline und für jedes Jingle einen kontinuierlichen Player (0x01).

**PlayerIsRunning**

0x00 Der Player wurde nur auf eine bestimmte Position gesetzt und läuft nicht weiter.

> 0x00 Der Player läuft weiter

**Time**

Dieser Wert gibt die zeitliche Position im Ablauf an und kann als TimeCode verwendet werden, falls der Player ein Main-Player ist.

Bei allen anderen Playertypen kann dieser Wert vernachlässigt werden.

Beim Abspielen einer Timeline ist es durchaus erlaubt das Abspielen anzuhalten, in der Timeline zu springen oder bestimmte Positionen in der Timeline anzusteuern. Der Timecode liefert nicht unbedingt kontinuierlich fortschreitende Werte, sondern immer den aktuellen Spielstand der Timeline.

## PlaySPITObject ( OPFlag = 38)

Server -> Client
Client -> Server (Remote)

| | | | | |
|---|---|---|---|---|
| BytesCount | Bytes | int | 4 | Number of bytes of the OP part |
| PlayerID | Player ID, maybe an UUID | String iso 8859-15 | 41 | Max 40 characters + Null (0x00)<br>The server can have multiple players that play objects simultaneously |
| SPIT_ObjectID | UUID | String iso 8859-15 | 41 | Max 40 characters + Null (0x00) |
| FrameInsideObject | Frames | long | 8 | Time inside of the object |
| PreviousPosition | | boolean | 1 | 0x00 = false<br>normal playing of object<br>> 0x00 = true |

| | | | | |
|---|---|---|---|---|
| | | | | the object is played from an old position to fade out |
| Value | ppm | int | 4 | fade status, a value of an event, ..<br>or increase actual value<br>or decrease actual value<br>see - ValueFlag |
| ValueFlag | | | 1 | 0x00 = direct<br>0x01 = increase<br>0x02 = decrease<br><br>ox03 – 0xFF reserverd |
| | | | ***100*** | |

**PlayerID**

Die ID des Players.
Zur Erinnerung: es können mehrere Player gleichzeitig Aktionen auslösen, so dass PlaySPITObject Nachrichten durchmischt ankommen können. Daher sollte jeder Player eine eindeutige ID (softwareweit) erhalten. Dies kann eine festgelegte ID oder aber auch eine UUID sein.

**SPIT_ObjectID**

Die ID des SPIT Objektes

**FrameInsideObject**

Die aktuelle Zeit in Frames innerhalb des SPIT Objektes.

Anhand der Parameter FrameLength, FrameFadeInLength, FrameFadeOutLength des SPIT Objektes kann der Ein- oder Ausblendfaktor berechnet werden. Dieser kann aber im Parameter 'Value' (siehe weiter unten) schon berücksichtigt sein.

**PreviousPosition**

Werden Sprünge in einer Timeline durchgeführt, so könnten Fadeübergang für diese Sprünge definiert sein. Dies kann bedeuten, dass ein Objekt ab seiner vorherigen Position ausgeblendet wird, obwohl die aktuelle Position in der Timeline schon an ganz anderer Stelle ist.

1. 0x00 = das Objekt befindet sich an der aktuellen Position in der Timeline und wird normal abgespielt.
2. > 0x00 = Es wird ein Sprung / Übergang durchgeführt, das Objekt wird von seiner vorherigen Position ausgeblendet.

**Value (in ppm)**

Bei SPIT Objekten, die ein- oder ausgeblendet werden können ist dies das Produkt des Objektvolumens und des Ein- oder Ausblendwertes.

Bei Playern, die nicht kontinuierlich arbeiten, sondern nur auf Ereignisse reagieren, kann dies z.B. der Stellwert eines Faders sein.

Dabei gibt 'ValueFlag' ob 'Value' direkt übernommen wird, 'Value' zum aktuellen Wert addiert oder vom aktuellen Wert angezogen wird.

Der eigentliche Wert des Objektes errechnet sich aus dem:

'aktuellen Wert' * ((float)'ValueFactor' / 1000000),

'ValueFactor' stammt aus ReportSPITObject.

**ValueFlag**

Das ValueFlag gibt an, ob der Wert in 'Value'

- direkt übernommen wird (0x00 direct)
- zu dem aktuellen addiert wird (0x01 increase)
- vom aktuellen Wert abgezogen wird (0x02 decrease)

Hat ValueFlag einen anderen Wert als 0x00, 0x01 oder 0x02 so wird dieser als 0x00 interpretiert.

## PlaySPITEnd (OPFlag = 39)

Server -> Client
Client -> Server (Remote)

| | | | | |
|---|---|---|---|---|
| BytesCount | Bytes | int | 4 | Number of bytes of the OP part |
| PlayerID | UUID | String iso 8859-15 | 41 | Max 40 characters + Null (0x00) |
| | | | ***45*** | |

**PlayerID**

Die ID des Players

Nachdem Empfang von PlaySPITEnd wurden alle aktuell betroffenen SPIT Objekte (PlaySPITObject - SPITObjectID) übermittelt. Die Aktionen dieser SPIT Objekte können ausgeführt/aktualisiert werden.

Fall 1: Der Player ist ein kontinuierlicher Player (PlayerType 0x02 oder 0x01 in PlaySPITStart)

Eventuell sind noch SPIT Objekte aus der vorherigen Abspielsequenz aktiv. Sind diese SPIT Objekte in der aktuellen Abspielsequenz nicht mehr enthalten, so müssen deren Aktionen beendet werden.

Fall 2: der Player ist ein diskreter (Event-Player) (PlayerType 0x00 in PlaySPITStart)

Es werden nur die Aktionen der enthaltenen SPIT Objekte aktualisiert. Alle anderen Aktionen behalten ihren aktuellen Status.

Um herauszufinden, ob eine Objekt gerade gestartet oder beendet wurde, kann man sich die IDs der zuletzt gespielten Objekte merken (aus der letzten Play-Sequenz).

- Ist in der aktuellen Play-Sequenz ein Objekt dabei, das in der letzten Play-Sequenz nicht dabei war, so wurde das Objekt gerade gestartet.
- War in der letzten Play-Sequenz ein Objekt dabei, das in der aktuellen Play-Sequenz nicht dabei ist, so wurde das Objekt gerade beendet.

## ReportPlayStatus (OPFlag = 40)

Server->Client (REMOTE)
(Remote – Server sendet den aktuellen Status des SPIT Objektes an den Client,
umgekehrt zu PlaySPITObject, hier sendet der Client den zu ändernden Status an den Server)
Dies kann genutzt werden um z.B. im Server geänderte Faderstellungen an den Client zu übermitteln.

Client->Server??? nicht vorgesehen

| | | | | |
|---|---|---|---|---|
| BytesCount | Bytes | int | 4 | Number of bytes of the OP part |
| SPIT_ObjectID | UUID | String iso 8859-15 | 41 | Max 40 characters + Null (0x00) |
| FrameInsideObject | Frames | long | 8 | Time inside of the object |
| Value | ppm | int | 4 | Actual value of SPIT Object in ppm |
| | | | 57 | |

**SPIT_ObjectID**

Die ID des SPIT Objektes

**FrameInsideObject**

Die aktuelle Zeit in Frames innerhalb des SPIT Objektes.

**Value (in ppm)**

Wert, z.B. Faderstellung

Es sind unterschiedliche SPIT Objekte denkbar:

1. Das SPIT Objekt löst einfach nur eine Aktion aus, wie z.B. Sprung zu nächsten Szene.
   (ValueUse in ReportSPITObject ist auf 0x00 gesetzt)
2. Das SPIT_Objekt ist z.B. dafür gedacht Animationen, Bilder etc. ein- oder auszublenden.
   (ValueUse in ReportSPITObject ist auf 0x01 gesetzt)
   Der Wert 'Value' wird dann z.B. als Ein- oder Ausblendzustand interpretiert.
3. Das SPIT Objekt steuert eine regelbare Funktion, wie Fader, Bewegungen, etc.
   (ValueUse in ReportSPITObject ist auf 0x01 oder 0x02 gesetzt)
   Der Wert 'Value' wird dann z.B. als Reglerstellung interpretiert.

Im Fall 1. wird 'Value' nicht beachtet, in den Fällen 2. und 3. wird 'Value' beachtet. Das SPIT Objekt speichert hier den Wert ('Value' und/oder 'FrameInsideObject') als Zustand (Playstatus / Actionstatus) ab.

Im Fall der Remotesteuerung des SPIT Servers durch den SPIT Client, wird vom SPIT Client aus eine PlaySPIT-Sequenz an den Server gesendet. Wird eine z.B. eine Reglerstellung vom Client ferngesteuert, so wird in PlaySPITObject die Reglerstellung oder die Veränderung der Reglerstellung in 'Value' an den Server gesendet. Umgekehrt kann der Server seine aktuelle Reglerstellung (den Play- oder Actionstatus) an den Client senden – dies geschieht mit ReportPlayStatus. Der Client könnte dann in seiner grafischen Oberfläche den Regler anpassen.

## Das Player Konzept

Ein Server, der SPIT Aktionen auf einem Client auslösen kann, wird Aktionen aus unterschiedlichen

Situationen auslösen eventuell gleichzeitig aus mehreren Situationen. Umgekehrt gilt dies vielleicht auch für einen Client, der Remotefunktionen des Servers aufruft.

1. Der Server besitzt eine Art Timeline, die kontinuierlich abgespielt wird.
2. Der Server spielt manuell gesteuert bestimmte Sequenzen ab, z.B. Jingles
3. Der Server übermittelt nur den Wert eines Ereignisses, wir z.B. eine Faderbewegung, ein Mausklick, ...

In der Server Software wird es Instanzen geben, die diese Aufgaben bearbeiten. Für SPIT Clients ist es wichtig zu wissen von welcher Instanz (welcher Art Instanz) SPIT Abspielfolgen gesendet werden. Daher wird im SPIT Protokoll der fiktive Begriff 'Player' eingeführt.

**Fall 1.): Timeline oder kontinuierlicher Projektablauf**

In diesem Fall wird die Server Software eine (meist nur eine) Instanz besitzen, die in regelmäßigen Abständen, den Zustand der Timeline abfragt und übermittelt. Im Prinzip wird diese Instanz immer die aktuelle Position im Gesamtablauf kennen. Solch eine Instanz könnte man auch als Main-Instanz bezeichnen.

Solch eine Instanz kann eine Art Timecode übermitteln – die aktuelle Position im Gesamtablauf.

Im SPIT Protokoll taucht diese Instanz als Main Player auf. Wobei der Main Player kein eigenes Objekt ist, sondern immer wenn SPIT-Abspielsequenzen an die Gegenseite gesendet werden, wird in den PlaySPITxxx Nachrichten eine ID für die (Main-) Instanz eingetragen und in der PlaySPITStart Nachricht wird der Parameter PlayType auf 0x02 (main) gesetzt.

**Fall 2.) Kontinuierliches Abspielen von Sequenzen, die zu unterschiedlichen Zeiten starten können**

Dieser Fall tritt auf, wenn der Server z.B. Jingles ausführt, die jederzeit manuell ausgelöst werden können. Diese Aktionen haben keinen Bezug zum Gesamtablauf, können sich aber über eine gewisse Dauer erstrecken.

Hier wird es in der Server Software wiederum Instanzen geben, die Jingles abfragen und deren Zustand übermitteln.

Im SPIT Protokoll tauchen diese Instanzen als kontinuierliche Player auf. Diese Player sind keine eigenen Objekte, sondern immer wenn SPIT-Abspielsequenzen an die Gegenseite gesendet werden, wird in den PlaySPITxxx Nachrichten eine ID für die jeweilige Instanz eingetragen und in der PlaySPITStart Nachricht wird der Parameter PlayType auf 0x01 (continious) gesetzt.

**Fall 3.) Ein einzelnes Ereignis wird übermittelt (Zustand eines Faders, Start/Stopp-Befehl,..)**

Hier wird der Server auf ein Ereignis seiner Eventqueue (oder mehrerer Eventqueues) reagieren. Diese Aktionen besitzen keine Länge, sondern übermitteln einen Wert eines Ereignisses.

Im SPIT Protokoll tauchen diese Fälle als diskrete Player auf. Diese Player sind keine eigenen Objekte, sondern immer wenn SPIT-Abspielsequenzen an die Gegenseite gesendet werden, wird in den PlaySPITxxx Nachrichten eine ID für die jeweilige Eventqueue (Event) eingetragen und in der PlaySPITStart Nachricht wird der Parameter PlayType auf 0x00 (discrete) gesetzt.

**Bearbeitung in Fall 1.) und 2.)**

Da eine Timeline oder ein Jingle in regelmäßigen Abständen (Abtastrate) abgefragt werden und nur der jeweilige Zustand des Abtastschrittes übermittelt wird, gibt es kein STOPP für ein SPIT Objekt.

Beispiel:

Nehmen wir an, es gibt zwei SPIT Objekte (Objekt1 und Objekt2), die versetzt übereinander liegen.

Im Abtastschritt (x) sind beide Objekte erfasst, es wird folgende Abspiel-Sequenz an die Gegenseite gesendet:

PlaySPITStart, PlaySPITObject (Objekt1), PlaySPITObject (Objekt2), PlaySPITEnd.

Nehmen wir an, im nächsten Abtastschritt (x+1) befinden wir uns hinter dem Objekt1, aber noch mitten in Objekt2. Es wird folgende Abspiel-Sequenz an die Gegenseite gesendet:

PlaySPITStart, PlaySPITObject (Objekt2), PlaySPITEnd

Das Objekt1 ist nicht mehr dabei, hat aber vielleicht noch Zustandswerte aus Schritt (x). Da es im Schritt (x+1) aber nicht mehr mitspielt, sollten die Zustandswerte genullt werden.

Dies bedeutet, dass der Empfänger von Abspiel-Sequenzen sich immer (pro Player) die Objekte aus der vorhergehenden Abspiel-Sequenz merken muss und Objekte, die in der aktuellen Abspiel-Sequenz nicht mehr dabei sind auf einen Null-Zustand setzen muss.

**Bearbeitung in Fall 3.)**

In diesem Fall werden nur die Änderungen eines bestimmten Wertes übermittelt, der auch beibehalten werden soll, bis er wiederum geändert wird.

Hier bleibt der (geänderte) Zustand eines SPIT Objektes erhalten, auch wenn dies in der aktuellen Abspiel-Sequenz nicht mehr enthalten ist.

# SPIT Client Protokoll Ablauf

| SPIT_Client | |
|---|---|
| **Empfangen vom Server** | **Senden zum Server** |
| **Verbindung zum Server** | |
| | **Connect** |
| **ConnectAnswer** | |
| (Connected = true->) | **----** |
| (Connected = false->) | ---- |
| **Server meldet sein Projekt<br>(nach der Verbindung oder Server hat ein neues Projekt geladen)** | |
| **ReportSPITProject**<br>(setze für alle SPIT Typen und SPIT Objekte Confirmed = false) | |
| -> | **RequestProjectTransfer** |
| | **ReportSPITProject** |
| **Server meldet, dass er sein Projekt geschlossen hat** | |
| **CloseSPITProject**<br>(setze für alle SPIT Typen und SPIT Objekte Confirmed = false)<br>Der Server wird für alle im aktuellen Server Projekt vorhandenen SPIT Objekte und alle SPIT Typen die entsprechende RemoveSPITObject/Type Nachrichten senden. | |
| **Client öffnet ein neues Projekt** | |
| (altes Projekt wird geschlossen) | **CloseSPITProject** |
| **(**wenn ein neues Client Projekt geladen ist,<br>setze für alle SPIT Typen und SPIT Objekte Confirmed = false) | **ReportSPITProject**<br><br>**RequestSPITProject** |
| **Server sendet Projekt-Transfer** | |
| **StartProjectTransfer** | |
| **ReportSPITType**<br>**Wenn Source = 0x01 (Remote):**<br>füge den SPIT Typ der Remoteliste hinzu oder ändere SPIT Typ in der Remoteliste<br><br>**Wenn Source = 0x00:**<br>(SPIT Typ existiert nicht -> füge SPIT Typ hinzu -> setze Confirmed = true) | |

| | |
|---|---|
| (SPIT Typ existiert:<br>Parameter sind gleich -> setze Confirmed = true<br>Parameter unterschiedlich -> setze Confirmed = false) | (Server listet den SPIT Typ als Konflikt) |
| **ReportSPITObject**<br>**Wenn Source = 0x01 (Remote):**<br>füge das SPIT Objekt der Remoteliste hinzu oder ändere SPIT_ Object in der Remoteliste<br><br>**Wenn Source = 0x00:**<br>(SPIT Objekt existiert nicht -> füge SPIT Objekt hinzu, setze Confirmed = true)<br>(SPIT Objekt existiert:<br>Parameter sind gleich -> setze Confirmed = true<br>Parameter unterschiedlich -> setze Confirmed = false) | (Server listet das SPIT Objekt als Konflikt) |
| **EndProjectTransfer** | |
| **Projekt Transfer (Aufforderung kommt vom Server)** | |
| **RequestProjectTransfer** | |
| | **StartProjectTransfer** |
| (für alle SPIT Typen) | **ReportSPITType** |
| (für alle SPIT Objekte) | **ReportSPITObject**<br>(Ist dem SPIT Objekt eine Aktion zugewiesen, dann muss ActionAssigned >= 0x01, ansonsten ActionAssigned = 0x00 gesetzt sein) |
| | **EndProjectTransfer** |
| **Server hat einen SPIT Typ / ein SPIT Objekt hinzugefügt/geändert** | |
| **ReportSPITType**<br>**Wenn Source = 0x01 (Remote):**<br>füge den SPIT Typ der Remoteliste hinzu oder ändere SPIT Typ in der Remoteliste<br><br>**Wenn Source = 0x00:**<br>(SPIT Typ existiert – ändere SPIT Typ setze Confirmed = true)<br>(existiert nicht – füge SPIT Typ hinzu, setze Confirmed = true) | |
| **ReportSPITObject**<br>**Wenn im referenzierten SPIT Typ Source = 0x01 (Remote):**<br>füge das SPIT Objekt der Remoteliste hinzu oder ändere SPIT_ Object in der Remoteliste<br><br>**Wenn im referenzierten SPIT Typ Source = 0x00:** | |

| | |
|---|---|
| (SPIT Objekt existiert – ändere SPIT_ Object, setze Confirmed = true)<br>SPIT Objekt existiert nicht – füge SPIT_ Object hinzu, setze Confirmed = true) | |
| **Server hat einen SPIT Typ / ein SPIT Objekt gelöscht (eventuell durch das Schließen eines Projektes)** | |
| **RemoveSPITType**<br>(Lösche SPIT Typ,<br>eventuell lösche alle zugehörigen SPIT Objekte, dies wird aber normalerweise vom Server erledigt)<br>(ACHTUNG der Server könnte eine UNDO Funktion besitzen, die Client Einstellungen für den SPIT Typ könnten mit der zugehörigen SPIT Typ ID in einer Art Papierkorb zwischengespeichert werden, um bei Bedarf reproduziert zu werden) | |
| **RemoveSPITObject**<br>(Lösche SPIT Objekt)<br>(ACHTUNG der Server könnte eine UNDO Funktion besitzen, die Client Einstellungen für den SPIT Typ könnten mit der zugehörigen SPIT Object ID in einer Art Papierkorb zwischengespeichert werden, um bei Bedarf reproduziert zu werden) | |
| **Client fügt einen neuen SPIT Typ oder ein neues SPIT Objekt hinzu oder ändert einen SPIT Typ oder ein SPIT Objekt** | |
| (neuer SPIT Typ oder SPIT Typ geändert, setze Confirmed = false)<br>(Server wird mit ReportSPITType antworten -> setze Confirmed = true) | **ReportSPITType** |
| (neues SPIT Objekt oder SPIT Objekt geändert, setze Confirmed = false)<br>(Server wird mit ReportSPITObject antworten -> setze Confirmed = true) | **ReportSPITObject**<br>(Ist dem SPIT Objekt eine Aktion zugewiesen, dann muss ActionAssigned >= 0x01, ansonsten ActionAssigned = 0x00 gesetzt sein) |
| **Client löscht einen SPIT Typ oder ein SPIT Objekt** | |
| (SPIT Typ gelöscht)<br>(vom Server können mehrere RemoveSPITObject Nachrichten kommen) | **RemoveSPITType** |
| (SPIT Objekt gelöscht) | **RemoveSPITObject** |
| **Client weist einem SPIT Objekt eine Aktion zu oder entfernt die zugewiesene Aktion** | |
| (setze ActionAssigned >= 0x01,<br>setze Confirmed = false)<br>oder<br>(setze ActionAssigned = 0x00, | **ReportSPITObject** |

| | |
|---|---|
| setze Confirmed = false)<br>(Server antwortet mit ReportSPITObject.<br>**Achtung: der Server setzt immer ActionAssigned = 0x00**) | |
| **Verbindung wird beendet / Client wird beendet** | |
| **Disconnect**<br>(setze für alle SPIT Typen und SPIT Objekte Confirmed = false)<br>Dies kann beim Schließen des Servers passieren. | |

# SPIT Client Projekt speichern

Da der SPIT_Client über den SPIT Server auch SPIT Typen und SPIT Objekte von anderen SPIT Clients bekommt, die bei ihm eventuell nicht verwendet werden, könnten sich nicht verwendete SPIT Typen und SPIT Objekte ansammeln.

**Vorsichtige Empfehlung:**

Der SPIT Client sollte nur Folgendes speichern:

- SPIT Typen, die der Client selbst erzeugt hat
- SPIT Objekte, die der Client selbst erzeugt hat
  und die referenzierten SPIT Typen
  (Es kann sein, dass ein Client ein SPIT Objekt erzeugt, das auf einen fremden SPIT Typ verweist)
- SPIT Objekte, denen eine Aktion zugewiesen ist und deren referenzierten SPIT Typen
  (es kann sein, dass einem fremden SPIT Objekt eine Aktion zugewiesen wurde

Alle anderen SPIT Typen oder SPIT Objekte werden bei einer Verbindung zum Server erneut übertragen.

# SPIT Server Protokoll Ablauf

| SPIT_Server | |
|---|---|
| **Empfangen** | **Senden** |
| **Client versucht sich zu verbinden** | |
| **Connect** | |
| (Login failed ->) | **ConnectAnswer (Connected = true)**<br>(nur zum sendenden Client) |
| (Login ok ->) | **ConnectAnswer (Connected = false)**<br>(nur zum sendenden Client)<br><br>**ReportSPITProject**<br>(nur zum sendenden Client)<br>*Client Reaktion: Client sendet RequestProjectTransfer und ReportSPITProject* |
| **Client meldet sein Projekt** | |
| **ReportSPITProject** | |
| | **RequestProjectTransfer**<br>(nur zum sendenden Client) |
| **Client meldet, dass er sein Projekt geschlossen hat** | |
| **CloseSPITProject**<br>(entferne aus allen SPIT Objekten den Client als Besitzer) | |
| **Server lädt eine neues Projekt** | |
| Der Server sollte vorher eine Sicherheitsabfrage durchführen, damit die Client Projekte gespeichert werden können.<br>(altes Projekt wird geschlossen) | **CloseSPITProject**<br>(zu allen Clients)<br>Für jedes SPIT Objekt<br>**RemoveSPITObject**<br>Für jeden SPIT Typ<br>**RemoveSPITType** |
| (Wenn das neue Projekt geladen ist) | **ReportSPITProject**<br>*Client Reaktion: Client sendet RequestProjectTransfer und ReportSPITProject* |
| **Client sendet Projekt Transfer** | |
| **StartProjectTransfer** | |
| **ReportSPITType**<br>(Wenn der SPIT Typ nicht existiert oder die Parameter sich unterscheiden, liste den SPIT Typ als Konflikt) | -- |

| | |
|---|---|
| (Wenn der SPIT Typ existiert und die Parameter gleich sind, setze Confirmed = true) | **ReportSPITType**<br>**(**Confirmed immer = true) |
| **ReportSPITObject**<br>(Wenn das SPITObject nicht existiert oder die Parameter unterschiedlich sind, liste das SPIT Objekt als Konflikt)<br><br>(Wenn das SPITObject existiert und die Parameter gleich sind,<br>setze Confirmed = true<br>Wenn AssignedAction >= 0x01, weise dem SPIT Objekt den sendenden Client als Besitzer zu) | ---<br><br>**ReportSPITObject**<br>(ActionAssigned immer = 0x00,<br>Confirmed immer = true) |
| **EndProjectTransfer** | |
| **Projekt Transfer (Aufforderung kommt vom Client)** | |
| | **StartProjectTransfer** (nur an sendenden Client) |
| (für alle SPITTypen)<br><br>(für alle Remote SPIT Typen des Servers) | **ReportSPITType** (nur an sendenden Client)<br>(Confirmed immer = true,<br>SourceFlag immer = 0x00)<br>**ReportSPITType** (nur an sendenden Client)<br>(Confirmed immer = true,<br>SourceFlag immer = 0x01) |
| (für alle SPIT Objekte)<br><br>(für alle Remote SPIT Objekte des Servers) | **ReportSPITObject** (nur an sendenden Client)<br>(AssignedAction immer = 0x00,<br>Confirmed immer = true,<br>SourceFlag immer = 0x00)<br>**ReportSPITObject** (nur an sendenden Client)<br>(AssignedAction immer = 0x00,<br>Confirmed immer = true,<br>SourceFlag immer = 0x01) |
| | **EndProjectTransfer** (nur an sendenden Client) |
| **Client hat einen SPIT Typ / ein SPIT Objekt hinzugefügt/geändert** | |
| **ReportSPITType**<br>(Wenn der SPIT Typ in den Konflikten auftritt> lösche Konflikt)<br><br>(Wenn der SPIT Typ existiert und die Parameter gleich sind)<br><br>(Wenn der SPITType existiert und die Parameter unterschiedlich sind, ändere SPIT Typ)<br><br>(Wenn der SPIT Typ nicht existiert, füge SPIT Typ hinzu | ---<br><br>**ReportSPITType** (nur an sendenden Client)<br>(Confirmed immer = true)<br><br>**ReportSPITType** (an alle Clients)<br>(Confirmed immer = true)<br><br>**ReportSPITType** (an alle Clients)<br>(Confirmed immer = true) |
| **ReportSPITObject** | |

| | |
|---|---|
| (Wenn das SPIT Objekt in den Konflikten auftritt> lösche Konflikt)<br><br>(Wenn das SPIT Objekt existiert und die Parameter gleich sind)<br><br>(Wenn das SPIT Objekt existiert und die Parameter unterschiedlich sind, ändere SPIT Objekt)<br><br>(Wenn das SPIT Objekt nicht existiert, füge SPIT Objekt hinzu)<br><br>ACHTUNG:<br>• war AssignedAction >= 0x01, wird der sendende Client als Besitzer hinzugefügt<br>• war AssignedAction = 0x00, wird der sendende Client als Besitzer entfernt | ---<br><br>**ReportSPITObject** (nur an sendenden Client)<br>(AssignedAction immer = 0x00,<br>Confirmed immer = true)<br>**ReportSPITObject** (an alle Clients)<br>(AssignedAction immer = 0x00,<br>Confirmed immer = true)<br><br>**ReportSPITObject** (an alle Clients)<br>(AssignedAction immer = 0x00,<br>Confirmed immer = true) |
| **Client hat einen SPIT Typ / ein SPIT Objekt gelöscht** | |
| **RemoveSPITType**<br>(Lösche alle SPIT Objekte, die diesen SPIT Typ referenzieren,<br>Lösche SPIT Typ) | **RemoveSPITObject** (an alle Clients)<br>(für alle betroffenen SPIT Objekte)<br>**RemoveSPITType** (an alle Clients) |
| **RemoveSPITObject**<br>(Lösche SPIT Objekt) | **RemoveSPITObject** (an alle Clients) |
| **Server fügt einen neuen SPIT Typ oder ein neues SPIT Objekt hinzu<br>oder ändert einen SPIT Typ oder ein SPIT Objekt** | |
| (neuer SPIT Typ oder SPIT Typ geändert, setze Confirmed = true) | **ReportSPITType** (an alle Clients)<br>(Confirmed immer = true)<br>(Wenn der SPIT Typ der Fernbedienung des Servers dient SourceFlag = 0x01, ansonsten SourceFlag = 0x00) |
| (neues SPIT Objekt oder SPIT Objekt geändert, setze Confirmed = true) | **ReportSPITObject** (an alle Clients)<br>(AssignedAction immer = 0x00,<br>Confirmed immer = true)<br>(Wenn das SPIT Objekt der Fernbedienung des Servers dient SourceFlag = 0x01, ansonsten SourceFlag = 0x00) |
| **Server löscht einen SPIT Typ oder ein SPIT Objekt** | |
| (Lösche alle SPIT Objekte, die diesen SPIT Typ referenzieren,<br>Lösche SPIT Typ) | **RemoveSPITObject** (an alle Clients)<br>(für alle betroffenen SPIT Objekte)<br>**RemoveSPITType** (an alle Clients) |
| (SPIT Objekt gelöscht) | **RemoveSPITObject** (an alle Clients) |
| **Client beendet seine Verbindung** | |

| | |
|---|---|
| **Disconnect**<br>(entferne aus den Konflikten alle Konflikte, die diesen Client betreffen)<br>(entferne aus allen SPIT Objekten den Client als Besitzer) | |
| **Server wird beendet** | |
| | **Disconnect**<br>Es wird kein CloseSPITProject gesendet, damit die Clients ihren aktuellen Zustand behalten. |

## SPIT Server Projekt speichern

Der SPIT Server speichert alle vorhandenen SPIT Typen und SPIT Objekte. Der Benutzer muss eventuell vor dem Speichern aufräumen.

# Anhang: SAD Server Auto Detect
# (SPIT Server Auto Detect)

**Client**
Der Client sendet in Abständen per UDP folgenden Inhalt:
#SAD##,##spit#
jeweils an:
alle IP4-Broadcast Adressen des Rechners
und
an die Multicast IP6 Adresse "ff02::1"

Zum Inhalt:
*#SAD#* ist der Precode
*#,#* ist der LineSeparator
*#spit#* ist eine Kennung für SPIT server

**Server**
Der Server tritt der Multicastgroup "ff02::1" bei.

Sobald ein UDP-Paket von einem SPIT Client empfangen wird:

1. erkennt der Server die IP-Adresse und den Port aus dem UDP-Paket (UDP-Protokoll)
2. der Server sendet eine Antwort mit folgendem Inhalt an die IPAddresse/Port des Clients:

#SAD##,##spit##,#*server task name*#,#*serverid*#,#*computername*#,#*serverport*#,#*clientIP*
Zum Inhalt:
*#SAD#* ist der Precode
*#,#* ist der LineSeparator
*#spit#* ist eine Kennung für SPIT server
*server task name* ist der sprechende Name für #spit#
*serverid* ist eine ID für den Aufgabenbereich des Servers (in der Regel: #spit#)
*computername* ist der Computername auf dem der Server läuft
*serverport* ist der Port über den der Client den Server erreichen kann
*clientIP* ist die IP Adresse des Clients über die der Client den Server erreicht hat
**Anmerkung:**
Der liveShow SPIT Server verwendet den Port: 15120

# Inhaltsverzeichnis