SPIT

Show Protocol Incorporated Trigger
Version 1

StandardPort: 15120 (changable)

Prolog

For the course of a complex show different trades have to work together. As a rule, every trades uses their own software (control hardware).
Currently, SMPTE timecodes or MIDI commands are used to synchronize the cooperation of different software.

SMPTE Timecode

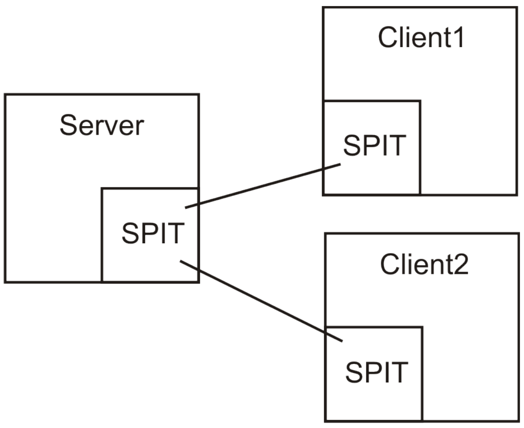
The SMPTE timecode is generated centrally and transmitted to all "connected" systems. It can be used to ensure a synchronous start and possibly synchronization of the different software. However, the actual process is left to each individual software.
Disadvantages: Only a timed sequence can be realized. Interruptions or jumps in the process are not easily controllable.

MIDI Commands

For MIDI commands, MIDI parameters must be defined in each software. If a MIDI command is sent, the action specified there will be executed on the other side. Here are interruptions and jumps in the show process possible.
Disadvantages: The meaning of the parameters is not known to the respective opposite party, or must be specified manually there. This makes the operation prone to errors and complicated, an automatic adjustment of possible commands / actions is not possible.

SPIT

With SPIT, it should be possible to control other software via triggers. These triggers have a meaningful name that can be freely assigned. There is a software / hardware that is responsible for the distribution of the triggers and for the triggering of the triggers (SPIT Server) and other software / hardware, which logs on to a SPIT server and is responsible for the execution of certain actions (SPIT Client). A DMX control software could thus trigger an animation on another software.
The triggers can be created and changed flexibly, they are easy to classify by their name.
The connection between client and server is established via a network (LAN, WLAN, Internet).



Concept

SPIT Server

The SPIT server may be software or hardware that has implemented the SPIT protocol. This would be e.g. a DMX control software, a DMX hardware console, etc.
The SPIT server is the central point for all SPIT clients. The SPIT server collects all triggers and forwards them to all registered SPIT clients.
The SPIT server is the place where triggers are fired.
The SPIT server arranges the triggers in a timed sequence (e.g., in a timeline), it can fire a trigger at any time. It can also create new triggers, modify or delete existing triggers.

SPIT Client

The SPIT client may be software or hardware that has implemented the SPIT protocol. This would be e.g. a video software, a mixer, etc.
The SPIT client receives all triggers via the SPIT server, but it can also create its own triggers, which in turn are forwarded to all other SPIT clients via the SPIT server.
The SPIT client can assign an action to a trigger. If this trigger is fired by the SPIT server, the assigned action is performed on the SPIT client.

Aufbau der SPIT Trigger (SPIT Typ – SPIT Objekt)

A trigger consists of two parts:

1. SPIT Typ
   The SPIT type describes an action / function to be triggered or the type / class of an action / function.
   This action could be executed several times in the process. A SPIT type does not trigger any action, it only describes or classifies it. Also, a SPIT type does not know exactly how the action will be executed, this is the task of the particular software that receives a SPIT object.
2. SPIT Object
   The SPIT object is the actual triggering element. It has a reference to a SPIT type. Multiple SPIT objects with the same reference to a SPIT type can exist. Thus, an action (described / classified by the SPIT type) may be performed at different locations.

The action can be freely assigned in the client software, the referenced SPIT type is not a mandatory specification, it is used for structuring.

If the server wants to trigger actions, he sends a play sequence:

PlaySPITStart
PlaySPITObject, .. (for each SPIT object to be triggered)
PlaySPITEnd

Conversely, the client can also send such play sequences if the server provides Remote SPIT objects.

Short comparison with a MIDI controller:

MIDI

For a MIDI controller, the controller sends byte strings to a connected client (software, mixer, etc.). The client must be entered how it reacts to specific byte sequences. The byte strings do not convey any content information, they are defined in the MIDI specification originally designed for the exchange of musical control information. The informal exchange between the different trades consists of the exchange of byte sequences, which have to be re-entered manually with each change.

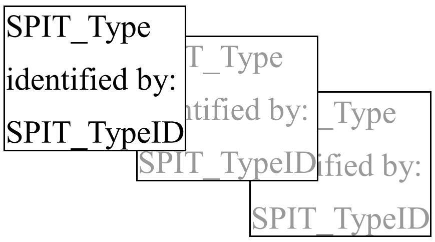
SPIT

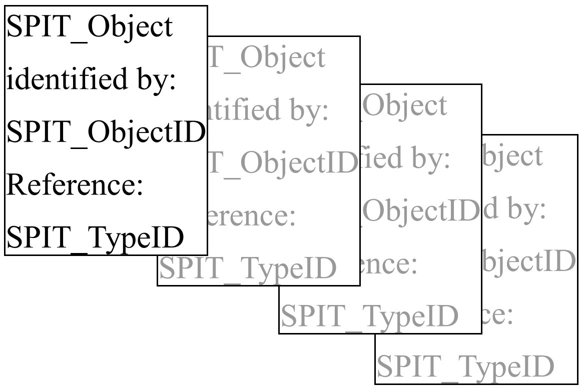
The SPIT server can query from the SPIT client all SPIT types and SPIT objects created in it. The information exchange happens with talking descriptions. The server can now trigger an action on the client at any time by sending a play sequence with a reference to a SPIT object to the client. Conversely, the SPIT Server can also create SPIT types and SPIT objects that can later be linked to executable actions in the client. So you can work on a project on both sides.
If a connection is established between a SPIT client and a SPIT server, all SPIT types and SPIT objects can be synchronized. Thus, changes are very easy to carry out and all parties involved always have all the information at their disposal.

Client-Software: SPIT Client Structur

The SPIT client exposes its controllable functionality via SPIT types. This functionality may differ per loaded project. The SPIT types are also made available to other SPIT clients via the SPIT server. The connection to a SPIT server is made possible via the UDP network protocol, each UDP packet contains a SPIT message (see below). Theoretically, it would also be conceivable to set up the connection via another way, e.g. ArtNet, the SPIT messages are in the ArtNet trigger command. When the SPIT client connects to a SPIT server, both exchange their SPIT types and SPIT objects. Regardless of the type of connection, there is a logical connection between the client's SPIT project and the server's corresponding SPIT project. So the SPIT client can certainly be started on another computer and load the client project. In other words, the SPIT client is interchangeable because the logical exchange between the SPIT projects happens.

SPIT-Client (Software)
UDP Datasocket
identified by:
a) IP Address
b) Port number

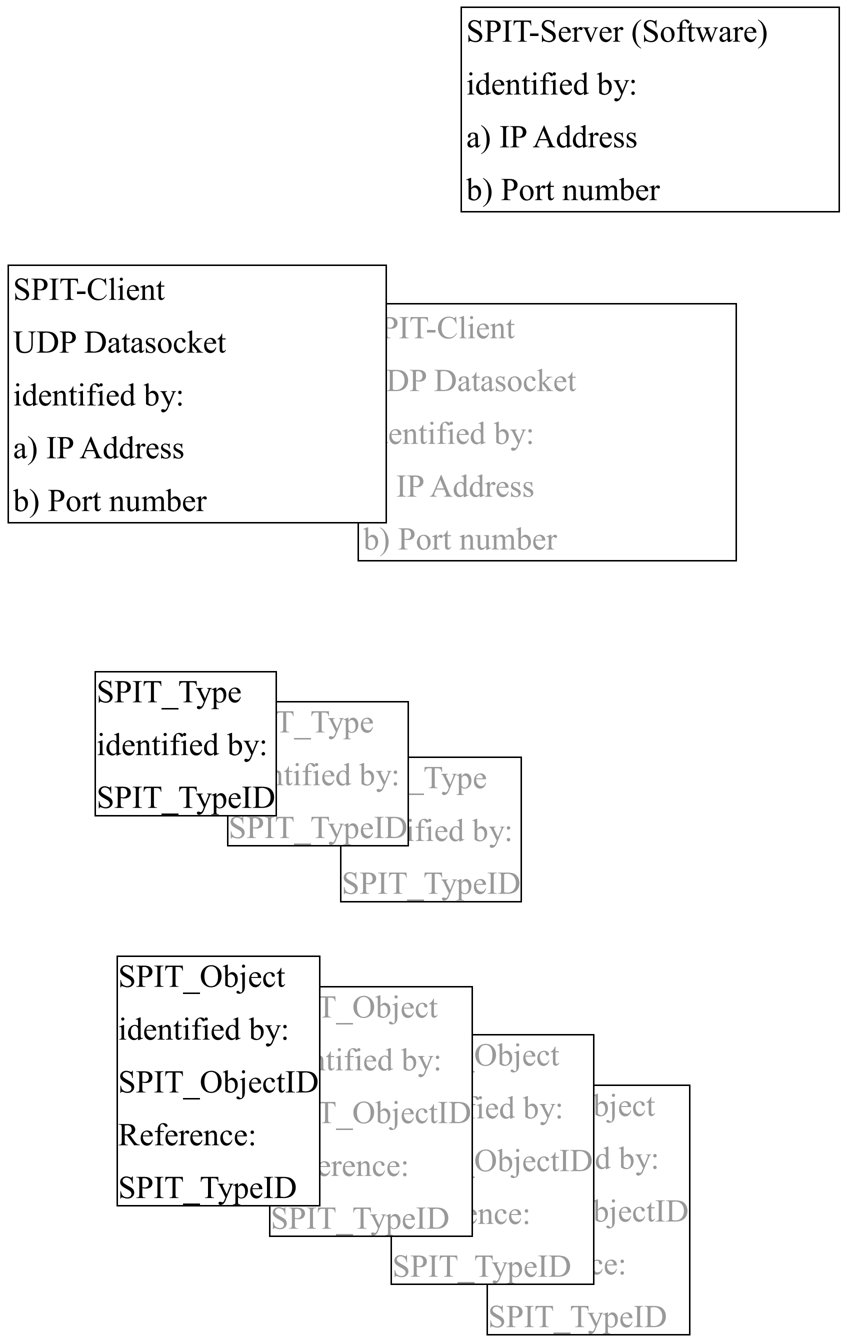




Each SPIT client has its SPIT type / SPIT objects and all other registered clients after connecting to the SPIT server

Server-Software: SPIT Server Struktur

When a SPIT client connects to the SPIT server, the SPIT server queries all SPIT types and all SPIT objects of the SPIT client, these can be synchronized and distributed to all registered SPIT clients. Thus, it is possible to make changes on the SPIT server or the SPIT client without both being connected.
Several SPIT clients can connect to a SPIT server. After synchronization, the SPIT server and all SPIT clients have the same SPIT types and SPIT objects.



The SPIT server collects all SPIT types and SPIT objects from all connected SPIT clients and forwards them to all clients.

SPIT Server - Remote Control

If the SPIT server allows certain functions / actions to be triggered by the SPIT client, it transfers the SPIT types and SPIT objects to the SPIT clients. These SPIT types and SPIT objects are marked so that the SPIT client recognizes them as SPIT server functions - see below (ReportSPITObject - 'ServerRemote').
The (remote) SPIT objects can be sent by the SPIT client in play sequences to the SPIT server, triggering actions on the server.

Transmission paths

A SPIT message can be sent directly over the UDP network protocol. This is also the preferred way to enable the exchange of SPIT messages between the server and client software.

Alternative transmission paths:

- TCP IP sockets are conceivable, but harder to program.
- The SPIT messages do not exceed the length of 512 bytes, so these could be e.g. also be sent via the ArtNet trigger command.

Excamples

The SPIT Server could control:

- mixing consoles
- animations software steuern (laser, pyrotechnics, ..)

Structure of SPIT messages (network protocol)

Each SPIT message consists of a byte sequence which has a header and an operation part.

SPIT message = HEADER + OPPART
The HEADER has a fixed length.
The respective OPPART has a fixed length.

Parameters

1. Strings are coded in ISO 8859-15, they have a fixed length and are null terminated (0x00).
2. Integers are always resolved in 4 Bytes (low byte first, little endian).
3. Longs are always resolved in 8 Bytes (low byte first, little endian)
4. Boolean are always resolved in one Byte (0x00 = false, >0x00 = true).
5. An ID is always 41 bytes long (ISO 8859-15) and null terminated (40 characters + 0x00 maximum), so the ID string can hold a UUID (Universal Unique Identifier). A UUID usually has 36 characters + possibly 2 brackets.
   In the case of remote objects, the ID can be artificially extended by two characters (if a SPIT object has different remote commands, several remote objects can be created, which have the two command characters appended to the ID).
6. Names usually have a length of 64 bytes and are null terminated (max 63 characters + 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. 0x00: message not sorted 0x01 .. 0xFF: When 0xFF is reached, 0x01 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

The SPIT Version

MessageNumber

Depending on the network architecture, UDP packets can overtake each other. To enable the receiver to sort the UDP packets, each message is provided with a MessageNumber. The sender must ensure that the MessageNumber is incremented continuously.

MessageNumber 0x01 - 0xFF:
The SPIT messages are numbered by the sender and when 0xFF is reached, it starts again at 0x01. The receiver can recognize by the MessageNumber if a UDP packet (SPIT message) has overtaken another one and can re-sort them.
Special case MessageNumber 0x00:
If the order of the SPIT messages is not important, the sender can assign the MessageNumber 0x00 to a SPIT message. The receiver processes those SPIT messages without paying attention to an order.

BytesCount

The number of bytes of the entire header

ServerID

The ID of the server, which the client can use to identify the server program.
The ServerID is set by the server when sending messages to the client.
As a rule, a client only connects to one server, in which case the ServerID is redundant.

ClientID

The server uses this ID to identify the sending client. Because multiple clients can connect to the server, each client must generate a unique ID (UUID).
The ClientID is set by the client when sending messages to the server.

OPFlag

The flag of the OPParts – see OPPART.

OPPART

The OPPart consists of the description of a SPIT command. The OPPART can consist of one of the following commands:

OP Connection

There is a possibility for a SPIT client to automatically find all SPIT servers within a network - see Appendix: 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

The name of the client computer (HostName). This information is used by the server for illustrative purposes.

Client Name

The name of the client software. This information is used by the server for illustrative purposes.

User Login Name

Username with which the client logs in to the server.
The server program can be protected against unqualified use by matching a user name and a user password.
If the server is to be open (unprotected), this parameter is ignored.

User Login Password

User password with which the client logs in
If the server is to be open (unprotected), this parameter is ignored.

ServerIP

The IP address of the server through which the client has reached the server.
If ServerAutodetect was used, the client knows the IP address from which it can reach the server based on the received UDP packet.
This information is used by the server for illustrative purposes.

ServerPort

Port number via which the client has reached the server program.
This information is used by the server for illustrative purposes.

<u>Behaviuor:</u>

After the message has been sent:

Client:

1. Wait of ConnectAnswer from the SPIT Servers

After the message has been received :

Server:

1. Sends ConnectAnswer to the SPIT client. The Connected parameter can be set to true or false:
   true = the SPIT server accepts the connection
   false = the SPIT server rejects the connection. (Maybe the login parameters are wrong, or ....)
2. If the connection is accepted, the SPIT server sends ReportSPITProject with the project data of the SPIT server.
   (this is not a SPIT project, but information about the project of the server that can serve the SPIT client to identify the SPIT server)

ConnectAnswer (OPFlag = 2)

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 > 0x00 = true
			137	

Server Computername

Computer name of the server. This information is used by the client for illustrative purposes.

Server Name

Name of the server software. This information is used by the client for illustrative purposes.

Server Step Time:

This is an informal value in which times playback commands (PlaySPITStart, ..., PlaySPITEnd) can be made.

A server could have multiple players (player for playing a timeline, player for playing a jingle, a player for moving a fader, ..). A player can consist of a software thread from a worker thread or simply be a fictitious player that only comes into effect for certain events (fader movement, press of a button, ...).

If the server has a player, e.g. plays a timeline, you can specify here the time in milliseconds between two play sequences (see PlaySPITStart, ..., PlaySPITEnd). This corresponds to the server's sampling rate. Such a player can also be declared as a MAIN player, which can transmit a timecode.

Note that here breaks and jumps in the timeline are allowed. It is also possible that the step time is not maintained continuously.

If the server does not have a player that provides a fixed sequence of playback commands, 0 can be

entered here as a value.
A player does not exist as a standalone object in the SPIT protocol, only a free PlayerID is transmitted to PlaySPITStart and PlaySPITEnd in order to differentiate the play sequences of different 'players'.

Connected

Indicates whether the server has accepted the connection or not.

- 0x00 (false) the server refused the connection
- >0x00 (true) the server has accepted the connection

Behaviour

After sending the message:

Server:

1. Send ReportSPITProject to the client if the connection has been accepted (see Connect)

After receiving the message :

Client:

1. Client is waiting for ReportSPITProject

Disconnect (OPFlag = 3)

Client->Server

Server->Client

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

Behaviour:

After sending the message:

Server and Client:

Close the UDPSockets

After receiving the message:

Server and Client:

Close the corresponding UDPSockets.

Watchdog (OPFlag = 8)

Client->Server

Server->Client

To check whether the other party is still connected, both the server and the client watchdog can send messages. The opposite side should respond immediately with a WatchdogAnswer message. If no WatchdogAnswer arrives after a defined time (watchdog timeout), it can be assumed that the other party is no longer reachable.
The watchdog timeout can be set individually in each software, since there is a rule that always a WatchdogAnswer is sent immediately. It should be noted that the other party may be busy processing another command.
The default timeout is 1000 ms in SPIT.

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

After receiving the message:

Server und Client:

1. Sende sofort eine WatchdogAnswer Nachricht

WatchdogAnswer (OPFlag = 9)

Client->Server

Server->Client

That's the answer that should be sent directly to a watchdog message.

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

After receiving the message

Server und Client:

1. Reset the watchdog wait time

Any reaction (receipt of any message) from the other party can also be used to reset the watchdog waiting time.

OP Projekt

ReportSPITProject (OPFlag = 11)

BytesCount	Bytes count of this OPPart	int	4	Number of bytes of the OP part low byte first
ProjectID	UUID	String iso 8859-15	41	Max 40 characters + Null

(Dateiname des Projektes)				(0x00)
ProjectName		String iso 8859-15	64	Max 63 characters + Null (0x00)
FramesPerSecond	Frames per second	int	4	Low byte first all time information are set in frames This can be used to convert between time and frames
			113	

ProjectID

Often it will happen that an old project will be copied to create a new project. But also a project can be saved under a different name. In the first case, a new project is created, the ProjectID would have to change. In the second case, the project remains, the ProjectID must not change. On the software side, the two cases can not be distinguished. Therefore, it is recommended to use the file name of a project as ID.

ProjectName

A meaningful name for the project. In many cases, this name will also be used for saving projects.

FramePerSecond

All time information is transmitted in the frames unit. Here, the conversion factor frames per second is specified.
It should be noted that usually the server determines this value and the client may have to recalculate all times.

Behaviour:

After receiving the message:

Client:

1. send RequestSPITProject
2. send ReportSPITProject

Server:

1. send 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	

If either the client or server closes a project, the other party must be notified.

Closing a project does not mean that the project is deleted on the opposite side!

CloseSPITProject can occur if on the opposite side:

1. creates a new project
2. loads another project
3. twhen the program is closed

Behaviour:

When receiving this message :

Client:

1. Set the 'Confirmed' flag to false for all SPIT types and SPIT objects.

Server:

1. Remove the client as owner from all SPIT objects

When a project is loaded or a blank project is created:

Client:
1. send ReportSPITProject
2. send RequestProjectTransfer

Server:
1. send ReportSPITProject

RequestProjectTransfer (OPFlag = 15)

Server->Client

Client->Server

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

In order to be able to query the SPIT objects and SPIT objects from the other side, the server or the client must send this message.

After sending the message:

Server and Client:

1. Wait for the sequence StartProjectTransfer, ..., EndProjectTransfer

After receiving the message:

1. send sequenz:
   a) StartProjectTransfer
   b) ReportSPITType message for every SPIT Type...
   c) ReportSPITObject message for every SPIT Object...
   d) EndProjectTransfer

StartProjectTransfer (OPFlag = 16)

Client->Server

Server->Client

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

Informs the other party that all SPIT types and all SPIT objects are now transferred.

EndProjectTransfer (OPFlag = 17)

Client->Server

Server->Client

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

Informs the far site that the complete transfer of the SPIT types and SPIT objects has ended.

SPIT Typ

The SPIT type indicates what type of trigger / action is involved.
In the video software this could e.g. Be the name of a video or a series of background animations.
In a control software this could e.g. the command 'next scene', or the indication of a specific DMX lamp.
For a mixer, this could be the knob of a particular input.

A SPIT type can be used several times in the show process. The SPIT type is not yet a complete trigger, it only classifies the action to be triggered.
The SPIT type has no parameters for start time, length, fadeIn, fadeout, ...
A trigger that can be triggered is the SPIT object, which contains a SPIT type as reference - see SPIT object. A SPIT object has other parameters that are important for its appearance and execution.

ReportSPITType (OPFlag = 21)

BytesCount	Bytes	int	4	Number of bytes of the OP part

Confirmed	Boolean		1	0x00 = false > 0x00 = true
SPIT_TypeID	UUID	String iso 8859-15	41	Max 40 characters + Null (0x00)
Fixed	Boolean		1	0x00 = false >= 0x01 = true
ServerRemote	Boolean	byte	1	>= 0x01 = true the object runs on the server the client can start the object (Remote) 0x00 = false the object runs on the client The server can start the object
ServerProjectInpended	Boolean	byte	1	0x00 = false > 0x00 = true
GroupName	String	String iso 8859-15	39	Max 38 characters + Null (0x00) 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
If either a SPIT type is created or changed by the server or the client, then a ReportSPITType should be sent to the opposite party, in which case the confirmed flag is usually set to 'false'. The far site responds with the same message, but here the confirmed flag is set to true. Thus, it is clear that the other side has received and implemented the message.

If the other party does not allow the editing of SPIT types, then this sends ReportSPITType with the old parameters and the Confirmed flag is set to 'true'.

Conversely, if the other party is not allowed to change the SPIT type, the Confirmed flag will be set to true already when sending ReportSPIT_Type.
See below Parameter Fixed
Thus, the following simple rule applies:
Whenever a ReportSPITType message is received and the Confirmed flag is set to true, the parameters are applied
.

SPIT_TypeID

A true unique identifier for the SPIT type. Based on this ID, a reference can be created in the SPIT object and the SPIT type is recognized on the opposite side.

Fixed

If a client has fixed immutable SPIT types, then true (0x01) must be entered here. This describes actions that are always possible and unchangeable.
If a SPIT type can be changed or deleted, then false (0x00) is entered here.

ServerRemote

IAs a rule, the SPIT types belong to a client, they describe the possible actions of a client. But the server could also provide actions to the client to eventually allow remote control by the client.
ServerRemote indicates who provides the action

0x00 (false): The client has the SPIT type, it can be triggered by the server via a SPIT object
The action is triggered on the client.
>= **0x01 (true)**: The server has the SPIT type, it can be triggered by the client via a SPIT object (remote). The action is triggered on the server.

ServerProjectIndepended

Normally the SPIT types are generated within a project. However, in the case of Remote SPIT types (and all SPIT objects that reference these SPIT types), there may also be types / objects that are independent of a project, e.g. Main volume, jump to the next scene, ...
0x00 false = the SPIT type depends on the loaded project of the server. Remote SPIT type and its SPIT objeke should be deleted when the server closes the project.
0x01 true = the SPIT type is independent of the loaded project of the server. Remote SPIT type and its SPIT objects can be preserved when the server closes the project.

GroupName

The group name can be used for the classification / grouping of SPIT types.

Name

A talking name for the type, the triggering action.

Default Delay

In some applications, starting an action is not always synonymous with the visual impact. This parameter specifies the delay in frames between the start of an action and the visible impact of the action. The value is a default value, it can be overwritten in the SPIT object.

In video software, this could be the time until a video is loaded.

Default Duration

Some actions have a length of time. This value is specified in frames and can be overridden in the SPIT object.

A video may e.g. have a fixed length.

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	

If a SPIT type is deleted, it must be reported to the other party.
If the far site does not allow the editing of SPIT types or the SPIT type is defined as fixed, the far site should not execute this message and send a ReportSPITType in response to this message, so that the SPIT type is recreated.
The server could have UNDO functionality, so the client should not completely remove its settings for the SPIT type, but cache them with the associated SPIT type ID. If the SPIT type is restored on the server and the server sends a ReportSPITType message, then the client can reproduce the corresponding setting from the clipboard again. When the client loads a saved project, the entire cache can be emptied.

SPIT Objekt

A SPIT object has a reference to a SPIT type (SPIT_TypeID) and other parameters that may be important to the execution of an action. The extent to which these parameters are used or interpreted is up to the respective software.
A SPIT object can only exist once. However, there may be multiple SPIT objects that have a reference to the same SPIT type.
In order to trigger actions on the other side, a sequence is sent to the opposite side:

1. PlaySPITStart
2. sequence of PlaySPITObjekten, each with a reference to a SPIT object ...
3. PlaySPITEnd

This is explained in more detail below – PlaySequence.

The triggering of an action is therefore not the SPIT object itself, but a PlaySPITObject message with the reference to a SPIT object.
Since the other party knows the SPIT object with its parameters, it ensures that all information for executing the action exists.

Times are given in frames - see ReportSPITProject.FramesPerSecond for the conversion.

ReportSPITObject (OPFlag = 31)

Server -> Client

Client -> Server

BytesCount	Bytes	int	4	Number of bytes of the OP part
Confirmed	Boolean		1	0x00 = false > 0x00 = true
SPIT_ObjectID	UUID	String iso 8859-15	41	Max 40 characters + Null (0x00)

Fixed	Boolean		1	0x00 = false > 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 0x01 = Value in Range (0 – 1000000 ppm) 0x02 = Value can be overridden
Track/Priority	Number	int	4	Layer level
Remark		String iso 8859-15	64	Max 63 characters + Null (0x00) HTML tags can be used e.g. new Line
Delay	Frames	long	8	
Duration	Frames	long	8	
ActionAssigned	Boolean		1	0x00 = false > 0x00 = true
			339	

Confirmed
If either a SPIT object is created or changed by the server or the client, then a ReportSPIT object should be sent to the remote site, in which case the confirmed flag is usually set to 'false'. The far site responds with the same message, but here the confirmed flag is set to true. Thus, it is clear that

the other side has received and implemented the message.
If the other party does not allow the editing of SPIT objects, then this ReportSPIT object sends the old parameters and the Confirmed flag is set to 'true'.
Conversely, if the other party is not allowed to change the SPIT object, the Confirmed flag will be set to true already when sending ReportSPIT_Object.

Thus, the following simple rule applies:
Whenever a ReportSPITObject message is received and the Confirmed flag is set to true, the parameters are applied.

SPIT_ObjectID

A true unique ID for the SPIT object. This ID identifies the SPIT object on the opposite side.

Fixed

If Fixed is set to true, the SPIT object is fixed, it can not be changed by the other party, even if the other party is allowed to edit SPIT objects.

ServerRemote

This is already determined by the SPIT type and is taken over here redundantly to allow a somewhat easier programming of a client.
As a rule, the SPIT objects trigger actions on the SPIT client.
But the SPIT server could also provide SPIT objects to allow remote control by the client. In this case, the client sends a play sequence and initiates actions on the server (remote).

ServerRemote indicates who provides the object / action

0x00 (false): The SPIT object / action is done on the client. The server triggers this action via a play sequence.
0X01 (true): The SPIT object / action is done on the server. The client triggers this action via a play sequence (remote).

ServerParams

Here the server can supply additional parameters as a string, which are required for processing / displaying the SPIT object. The clients DO NOT change this! The ServerParams should be stored by the clients and supplied with the ReportSPITObject.

FrameStart

The start time of a SPIT object in frames. If the server uses the SPIT object in a timeline / fixed sequence, this parameter makes sense. In all other cases, if e.g. the SPIT object is just a fader event, 0 can be entered here.

FrameLength

The length of a SPIT object in frames. A SPIT object can be displayed with a certain length. Often this will be the duration of the action (see below - Duration), but this is not mandatory.

FrameFadeInLength

SPIT objects can be faded in and out If the action is e.g. from the playback of a movie, it can be

faded in and out in its brightness.
Time for fading in frames.

FrameFadeOutLength

Time to fade out in frames - see FadeInTime

ValueFactor

Here, a factor can be specified in PartsPerMillion (ppm), which can be used to multiply the 'Value' from PlaySPITObject to determine the current value of the object.
'ValueFactor' may exceed 1000000, e.g. for raising or lowering the volume / brightness of an action in general.
The value should not be confused with the value of the PlaySPITObject message. When playing an object, Value indicates the current value, e.g. when an object is inserted or faded out.
Calculation of the value of an object:
'Value' * ((float) 'ValueFactor' / 1000000) - see PlaySPITObject.

ValueUse

Some SPIT clients do not use the value specification for the SPIT object. A jump to a video sequence will not take into account fade-in or fade-out values. Software that can show or hide animations will use the value specification.

Basically, the server sends the value and the client decides whether he uses the value or not.

ValueUse is especially important for the remote actions of the SPIT server. Here, the server informs the SPIT client whether the SPIT object is just a simple control command (e.g., next scene), or if it is a command that sets values (e.g., a lamp fader).

- ValueUse = 0x00 the value is not used
  The SPIT client can display the SPIT object as a button.
- ValueUse = 0x01 the value is used in the limits 0-100% (0 – 1000000 ppm)
  The SPIT client can display the SPIT object as a slider / fader.
- ValueUse = 0x02 the value can be overridden
  The SPIT client can display the SPIT object as a scroll wheel and in PlaySPITObject 'ValueFlag' should be set with (0x01 increase or 0x02 decrease) and 'Value' with the value change.
  The value changes must be sent, not the absolute value.

Track/Priority

If the server represents the SPIT object in a multi-track timeline, the track number could be entered here. Or the action consists of the playback of artwork that is to be displayed at a certain layer level (Z-plane).

Remark

Here further information / comments can be transmitted

.

Delay

In some applications, starting an action is not always synonymous with the visual impact. This

parameter specifies the delay in frames between the start of an action and the visible impact of the action. The value can overwrite the default value of the SPIT type.
In video software, this could be the time until the video assigned in the client is loaded.
Delay has no direct effects on the server, the value is only shown.

Duration

Some actions have a length of time. This value is specified in frames and overwrites the default value of the SPIT type.
A video may e.g. have a fixed length.
Duration has no direct effects on the server, the value is only displayed.

ActionAssigned

If a client assigns an action to a SPIT object, ActionAssigned is set to true (> = 0x01).

The server can use this information to prevent another client from deleting this SPIT object or its SPIT_type.

ActionAssigned may only be evaluated on the server . The client must always override this, depending on whether the action is assigned to it or not.
If an action is assigned or removed while the client is connected to the server, the client must send a ReportSPITObject message to the server.
ATTENTION: the server always sets 'ActionAssigned' in ReportSPITObject to 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	

If a SPIT object is deleted, this must be communicated to the other party.
If the far site does not allow the editing of SPIT objects or the SPIT object is defined as fixed, the remote site should not execute this message and send a ReportSPITObject in response to this message so that the SPIT object can be recreated.
The server could have UNDO functionality, so the client should not completely remove its settings for the SPIT object, but cache them with the associated SPIT object ID. If the SPIT object is restored to the server and the server sends a ReportSPITObject message, then the client can reproduce the corresponding setting from the clipboard again. When the client loads a saved project, the entire cache can be emptied.

PlaySPIT-Sequenz - The triggering of actions

Actions with their most important playback parameters are classified with the SPIT types and SPIT objects.

To trigger an action on the far site, the following sequence is sent:

1. PlaySPITStart
2. sequence of PlaySPITObject messages, jin each case with a reference to a SPIT object
3. PlaySPITEnd

All SPIT objects referenced in the PlaySPITObject messages play their assigned actions - the status of the SPIT object is changed or an action is simply triggered.

There may be no PlaySPITObject messages between PlaySPITStart and PlaySPITEnd.

There will be several instances in a server or client that are responsible for triggering an action on the opposite side, these instances are called players here. Players can be tangible objects / threads in software or they can only exist fictively, e.g. if events are handled by an event queue - see player concept below.

In the LiveShow software there are e.g. one player each for the timeline and each jingle.

Times are given in frames - see ReportSPITProject.FramesPerSecond for the conversion.

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 the player is the main player – the 'Time' can be used for time code 0x01 = additional continious the player is an addtional continious player 0x00 = additional discrete for events like fader movement, button click, .. >0x02 reserved
PlayerIsRunning		Boolean	1	0x00 = false the playcursor maybe only set once > 0x00 = true the playcursor is running
Time	Frames	long	8	Time inside of the project in frames

			119	

PlayerID

The ID of the player - see player concept below.

There can be several players (one for the timeline, one for jingles, one for fader movements, ..) and they can work at the same time. This means that sequences (PlaySPITStart, PlaySPITObject, ..., PlaySPITEnd) can get mixed up.
Each player has its own ID, so the PlaySPITObject messages can be assigned.

PlayerName

The name of the player, this can be used for presentations and simple troubleshooting.

PlayerType

The type of player - see player concept below.

1. 0x02= Main Player
   The player runs continuously and provides the temporal position in the overall process.
   This would be e.g. a player of a timeline.
   -> the Time parameter can be used as TimeCode.
2. 0x01 = additional continuous player
   The player is running continuously, but the player can start at some point in the process.
   This would be a player, e.g. Jingles is playing.
3. 0x00 = additional discrete player
   The player only appears on certain events (such as fader movements, mouse clicks, ...).

In the LiveShow software, a main player (0x02) for the timeline and for each jingle a continuous player (0x01).

PlayerIsRunning

0x00 The player has just been set to a certain position and will not continue.
> 0x00 The player continues to run

Time

This value indicates the time position in the process and can be used as time code if the player is a main player.
For all other player types this value can be neglected.
When playing a timeline, it is quite possible to stop playing, to jump in the timeline or to control certain positions in the timeline. The time code does not necessarily provide continuously progressive values, but always the current score of the 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) 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 normal playing of object > 0x00 = true the object is played from an old position to fade out
Value	ppm	int	4	fade status, a value of an event, .. or increase actual value or decrease actual value see - ValueFlag
ValueFlag			1	0x00 = direct 0x01 = increase 0x02 = decrease ox03 – 0xFF reserverd
			100	

PlayerID

The ID of the player.
As a reminder, multiple players can trigger actions simultaneously so that PlaySPITObject messages can arrive mixed. Therefore, each player should receive a unique ID (software-wide). This can be a fixed ID or a UUID.

SPIT_ObjectID

The ID of the SPIT object

FrameInsideObject

The current time in frames within the SPIT object.
The parameters FrameLength, FrameFadeInLength, FrameFadeOutLength of the SPIT object can be used to calculate the fade factor. However, this can already be taken into account in the 'Value' parameter (see below).

PreviousPosition

If jumps are done in a timeline, then fade transition could be defined for these jumps. This can mean that an object is hidden from its previous position, even though the current position in the timeline is already in a different place.

1. 0x00 = The object is at the current position in the timeline and plays normally.
2. > 0x00 = A jump / transition is made, the object is faded out from its previous position.

Value (in ppm)

For SPIT objects that can be faded in or faded out, this is the product of the object volume and the fade value.
For players that do not work continuously but only respond to events, this can be e.g. be the control value of a fader.
'ValueFlag' (ReportSPITObject) indicates whether 'Value' is taken over directly, 'Value' is added to the current value or subtracted from the current value.
The actual value of the object is calculated from:

'actual value' * ((float)'ValueFactor' / 1000000),

ValueFlag

The ValueFlag indicates whether the 'Value'

- is taken directly (0x00 direct)
- is added to the current value (0x01 increase)
- deducted from the current value (0x02 decrease)

If ValueFlag has a value other than 0x00, 0x01 or 0x02 then this is interpreted as 0x00.

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

The ID of the player

After receiving PlaySPITEnd all currently affected SPIT objects (PlaySPITObject - SPITObjectID) were transmitted. The actions of these SPIT objects can be executed / updated.

Case 1: The player is a continuous player (PlayerType 0x02 or 0x01 in PlaySPITStart)
Possibly still SPIT objects from the previous play sequence are active. If these SPIT objects are no longer contained in the current playback sequence, their actions must be ended.

Case 2: The player is a discrete (event player) (PlayerType 0x00 in PlaySPITStart)
Only the actions of the contained SPIT objects are updated. All other actions retain their current status.

To find out if an object has just been started or stopped, you can remember the IDs of the last played objects (from the last play sequence).

- If there is an object in the current play sequence that was not included in the last play sequence, the object has just been started.
- If there was an object in the last play sequence that is not included in the current play sequence, the object has just been ended.

ReportPlayStatus (OPFlag = 40)

Server->Client (REMOTE)
(Remote server sends the current status of the SPIT object to the client,
vice versa to PlaySPITObject, here the client sends the status to be changed to the server)
This can be used, e.g. in the server changed fader settings to the client to submit.

Client-> server ??? not provided

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

The ID of the SPIT Objektes

FrameInsideObject

The current time in frames within the SPIT object.

Value (in ppm)

Value, e.g. fader position.

Different SPIT objects are possible:

1. The SPIT object simply triggers only one action, e.g. Jump to the next scene. (ValueUse in ReportSPITObject is set to 0x00)
2. The SPIT_object is e.g. intended to fade in or fade out animations, pictures etc. (ValueUse in ReportSPITObject is set to 0x01)
   The value 'Value' will then be e.g. interpreted as fading in or out.
3. The SPIT object controls a controllable function, such as faders, movements, etc. (ValueUse in ReportSPITObject is set to 0x01 or 0x02)
   The value 'Value' will then be e.g. interpreted as a controller position.

In case 1. 'Value' is ignored, in cases 2 and 3 'Value' is taken into account. The SPIT object saves here the value ('Value' and / or 'FrameInsideObject') as state (Playstatus / Actionstatus).

In the case of the SPIT client remotely controlling the SPIT server, the SPIT client sends a PlaySPIT sequence to the server. If a e.g. If a controller position is remotely controlled by the client, then in PlaySPITObject the controller position or the change of the controller position in 'Value' is sent to the server.
Conversely, the server can send its current controller position (the play or action status) to the client - this is done with ReportPlayStatus. The client could then adjust the slider in its graphical interface.

The Player Concept

A server that can trigger SPIT actions on a client will trigger actions from different situations, possibly simultaneously from multiple situations. Conversely, this may also apply to a client that invokes remote server functions.

1. The server has a kind of timeline that plays continuously.
2. The server plays certain sequences manually controlled, e.g. jingles
3. The server only transmits the value of an event, e.g. a fader movement, a mouse click, ...

There will be instances in the server software that handle these tasks. For SPIT clients it is important to know from which instance (of which kind of instance) SPIT play sequences are sent. Therefore, the fictitious term 'player' is introduced in the SPIT protocol.

Fall 1.): Timeline or continuous project flow

In this case, the server software will have a (mostly only) instance that polls and submits the state of the timeline at regular intervals. In principle, this instance will always know the current position in the overall process. Such an instance could also be called a main instance.
Such an instance can convey a kind of timecode - the current position in the overall process.
In the SPIT protocol, this instance appears as a main player. Whereby the main player is not an object of its own, but whenever SPIT play sequences are sent to the other party, an ID for the (main) instance is entered in the PlaySPITxxx messages and in the PlaySPITStart message the parameter PlayType is set to 0x02 (main) set.

Fall 2.) Continuous playback of sequences that can start at different times

This case occurs when the server is e.g. Jingles that can be manually triggered at any time. These actions are not related to the overall process, but may take some time.
Again, there will be instances in the server software that query jingles and transmit their state.
In the SPIT protocol, these instances emerge as continuous players. These players are not their own objects, but whenever SPIT play sequences are sent to the other party, an ID for the respective instance is entered in the PlaySPITxxx messages and in the PlaySPITStart message the parameter PlayType is set to 0x01 (continious).

Fall 3.) A single event is transmitted (state of a fader, start / stop command, ..)

Here, the server will respond to an event of its event queue (or event queues). These actions have no length, but convey a value of an event.
In the SPIT protocol these cases appear as discrete players. These players are not their own objects, but whenever SPIT play sequences are sent to the other party, an ID for the respective Eventqueue

(Event) is entered in the PlaySPITxxx messages and in the PlaySPITStart message the parameter PlayType is set to 0x00 (discrete).

Processing in case 1.) und 2.)
Since a timeline or a jingle is queried at regular intervals (sampling rate) and only the respective state of the sampling step is transmitted, there is no STOP for a SPIT object.

Example:
Suppose there are two SPIT objects (Object1 and Object2) that are offset one above the other.
In the scanning step (x) both objects are detected, the following play sequence is sent to the opposite side:
PlaySPITStart, PlaySPITObject (Object1), PlaySPITObject (Object2), PlaySPITEnd.

Suppose that in the next sampling step (x + 1) we are behind the object1, but still in the middle of object2. The following play sequence is sent to the far site:
PlaySPITStart, PlaySPITObject (Object2), PlaySPITEnd

The object1 is no longer there, but may still have state values from step (x). Since it no longer plays along in step (x + 1), the state values should be zeroed.
This means that the receiver of play sequences must always remember (per player) the objects from the previous play sequence and must set objects that are no longer present in the current play sequence to a zero state.

Processing in case 3.)

In this case, only the changes of a certain value are transmitted, which should also be kept until it is changed again.
Here, the (changed) state of a SPIT object is retained, even if this is no longer contained in the current playback sequence.

SPIT Client Protocol expiration

SPIT_Client	
Receive from the server	**Send to the server**
Connection to the server	
	Connect
ConnectAnswer	
(Connected = true->)	**----**
(Connected = false->)	----
Server reports its project (after the connection or server has loaded a new project)	
ReportSPITProject (set for all SPIT types and SPIT objects Confirmed = false)	
->	**RequestProjectTransfer**
	ReportSPITProject
Server reports that he has closed his project	
CloseSPITProject (set for all SPIT types and SPIT objects Confirmed = false) The server will send the corresponding RemoveSPITObject / Type messages for all existing SPIT objects in the current server project and all SPIT types.	
Client öffnet ein neues Projekt	
(old project is closed)	**CloseSPITProject**
(when a new client project is loaded, set for all SPIT types and SPIT objects Confirmed = false)	**ReportSPITProject** **RequestSPITProject**
Server sendet Projekt-Transfer	
StartProjectTransfer	
ReportSPITType **when Source = 0x01 (Remote):** Add the SPIT type to the remotel list or change the SPIT type in the Remotel list **when Source = 0x00:** (SPIT type does not exist -> add SPIT type -> set Confirmed = true) (SPIT type exists:	

Parameters are the same -> set Confirmed = true Parameters different -> set Confirmed = false)	(Server lists the SPIT Type as a conflict)
ReportSPITObject **when Source = 0x01 (Remote):** Add the SPIT object to the remotel list or change SPIT_ Object in the remote list **Wenn Source = 0x00:** (SPIT object does not exist -> add SPIT object, set Confirmed = true) (SPIT object exists: Parameters are the same -> set Confirmed = true Parameters different -> set Confirmed = false)	(Server lists the SPIT Object as a conflict)
EndProjectTransfer	
Projekt Transfer (Request comes from the server**)**	
RequestProjectTransfer	
	StartProjectTransfer
(for all SPIT Typen)	**ReportSPITType**
(for all SPIT Objekte)	**ReportSPITObject** (If an action is assigned to the SPIT object, then ActionAssigned> = 0x01, otherwise ActionAssigned = 0x00 must be set)
	EndProjectTransfer
Server added / changed a SPIT type / SPIT object	
ReportSPITType **when Source = 0x01 (Remote):** Add the SPIT type to the remotel list or change the SPIT type in the remote list **Wenn Source = 0x00:** (SPIT type exists - change SPIT type set Confirmed = true) (does not exist - add SPIT type, set Confirmed = true)	
ReportSPITObject **If in the referenced SPIT Type Source = 0x01 (Remote):** Add the SPIT object to the remotel list or change SPIT_ Object in the remote list **If in the referenced SPIT Type Source = 0x00:** (SPIT object exists - change SPIT_ Object, set Confirmed	

= true) (SPIT object does not exist - add SPIT_ Object, set Confirmed = true)	
Server deleted a SPIT type / SPIT object (possibly by closing a project)	
RemoveSPITType (Delete SPIT type, possibly delete all associated SPIT objects, but this is usually done by the server) (ATTENTION the server could have an UNDO function, the client settings for the SPIT type could be cached with the associated SPIT type ID in a kind of recycle bin to be reproduced if necessary)	
RemoveSPITObject (Delete SPIT object) (NOTICE the server could have an UNDO function, the client settings for the SPIT type could be cached with the associated SPIT object ID in a kind of recycle bin to be reproduced if necessary)	
Client adds a new SPIT type or SPIT object or change a SPIT type or a SPIT object	
(new SPIT type or SPIT type changed, set Confirmed = false) (Server will respond with ReportSPITType -> set Confirmed = true)	**ReportSPITType**
(changed new SPIT object or SPIT object, set Confirmed = false) (Server will respond with ReportSPITObject -> set Confirmed = true)	**ReportSPITObject** (If an action is assigned to the SPIT object, then ActionAssigned> = 0x01, otherwise ActionAssigned = 0x00 must be set)
Client deletes a SPIT type or a SPIT object	
(SPIT Type deleted) (several RemoveSPITObject messages can come from the server)	**RemoveSPITType**
(SPIT Object deleted)	**RemoveSPITObject**
Client assigns an action to a SPIT object or removes the assigned action	
(set ActionAssigned> = 0x01, set Confirmed = false) or (set ActionAssigned = 0x00, set Confirmed = false) (Server responds with ReportSPITObject.	**ReportSPITObject**

Attention: the server always sets ActionAssigned = 0x00)	
Connection is terminated / client is terminated	
Disconnect (set for all SPIT types and SPIT objects Confirmed = false) This can happen when closing the server.	

SPIT Client save project

Since the SPIT_Client also receives SPIT types and SPIT objects from other SPIT clients via the SPIT server, which may not be used by it, unused SPIT types and SPIT objects could accumulate.

Careful recommendation:
The SPIT client should only save the following:

- SPIT types generated by the client itself
- SPIT objects that the client has created itself
  and the referenced SPIT types
  (It may be that a client creates a SPIT object that refers to a foreign SPIT type)
- SPIT objects to which an action is assigned and their referenced SPIT types
  (It may be that an action has been assigned to a foreign SPIT object

All other SPIT types or SPIT objects are retransmitted when connected to the server.

SPIT Server Protocol expiration

SPIT_Server	
received	**send**
Client tries to connect	
Connect	
(Login failed ->)	**ConnectAnswer (Connected = true)** (only to the sending client)
(Login ok ->)	**ConnectAnswer (Connected = false)** (only to the sending client) **ReportSPITProject** (only to the sending client) *Client Reaction: Client sends RequestProjectTransfer und ReportSPITProject*
Client reports his project	
ReportSPITProject	
	RequestProjectTransfer (only to the sending client)
Client reports that he has closed his project	
CloseSPITProject (remove the client as owner from all SPIT objects)	
Server loads a new project	
The server should first perform a security query to save the client projects. (old project is closed)	**CloseSPITProject** (to all clients) for each SPIT Objekt **RemoveSPITObject** for each SPIT Typ **RemoveSPITType**
(When the new project is loaded)	**ReportSPITProject** (to all clients) *Client Reaktion: Client sends RequestProjectTransfer und ReportSPITProject*
Client sends project transfer	
StartProjectTransfer	
ReportSPITType (If the SPIT type does not exist or the parameters differ, list the SPIT type as a conflict)	--

(If the SPIT type exists and the parameters are equal, set Confirmed = true)	**ReportSPITType** **(**Confirmed always = true)
ReportSPITObject (If the SPITObject does not exist or the parameters are different, list the SPIT object as a conflict) (If the SPITObject exists and the parameters are the same, set Confirmed = true If AssignedAction> = 0x01, assign the sending client as the owner of the SPIT object)	--- **ReportSPITObject** (ActionAssigned always = 0x00, Confirmed always = true)
EndProjectTransfer	
Projekt Transfer (Request comes from the client **)**	
	StartProjectTransfer (only to sending client)
(for each SPITType) (for each Remote SPIT Type of the Server)	**ReportSPITType** (only to sending client) (Confirmed immer = true, SourceFlag immer = 0x00) **ReportSPITType** (only to sending client) (Confirmed always = true, SourceFlag always = 0x01)
(for each SPIT Object) (für alle Remote SPIT Objekte des Servers)	**ReportSPITObject** (only to sending client) (AssignedAction always = 0x00, Confirmed always = true, SourceFlag always = 0x00) **ReportSPITObject** (only to sending client) (AssignedAction always = 0x00, Confirmed always = true, SourceFlag always = 0x01)
	EndProjectTransfer (only to sending client)
Client added / changed a SPIT type / SPIT object	
ReportSPITType (If the SPIT type occurs in the conflicts> delete conflict) (If the SPIT type exists and the parameters are the same) (If the SPITType exists and the parameters are different, change SPIT type) (If the SPIT type does not exist, add SPIT type)	--- **ReportSPITType** (only to sending client) (Confirmed always = true) **ReportSPITType** (to all clients) (Confirmed always = true) **ReportSPITType** (to all clients) (Confirmed always = true)
ReportSPITObject	

(If the SPIT object occurs in the conflicts> delete conflict) (If the SPIT object exists and the parameters are the same) (If the SPIT object exists and the parameters are different, change the SPIT object) (If the SPIT object does not exist, add SPIT object) CAUTION: was AssignedAction> = 0x01, the sending client is added as the owner AssignedAction = 0x00, the sending client is removed as the owner	--- **ReportSPITObject** (only to sending client) (AssignedAction always = 0x00, Confirmed always = true) **ReportSPITObject** (to all clients) (AssignedAction always = 0x00, Confirmed always = true) **ReportSPITObject** (to all clients) (AssignedAction always = 0x00, Confirmed always = true)
Client deleted a SPIT type / SPIT object	
RemoveSPITType (Delete all SPIT objects that reference this SPIT type Delete SPIT type)	**RemoveSPITObject** (to all clients) (for all affected SPIT objects) **RemoveSPITType** (to all clients)
RemoveSPITObject (delete SPIT Objekt)	**RemoveSPITObject** (to all clients)
Server adds a new SPIT type or SPIT object or change a SPIT type or a SPIT object	
(new SPIT type or SPIT type changed, set Confirmed = true)	**ReportSPITType** (to all clients) (Confirmed always = true) (If the SPIT Type is of the server's remote control SourceFlag = 0x01, otherwise SourceFlag = 0x00)
(changed new SPIT object or SPIT object, set Confirmed = true)	**ReportSPITObject** (to all clients) (AssignedAction always = 0x00, Confirmed always = true) (If the SPIT Object is of the server's remote control SourceFlag = 0x01, otherwise SourceFlag = 0x00)
Server deletes a SPIT type or a SPIT object	
(Delete all SPIT objects that reference this SPIT type, Delete SPIT type)	**RemoveSPITObject** (to all clients) (for each affected SPIT Object) **RemoveSPITType** (to all clients)
(SPIT object deleted)	**RemoveSPITObject** (to all clients)
Client terminates his connection	

Disconnect (remove from the conflicts all conflicts affecting this client) (remove the client as owner from all SPIT objects)	
Server is terminated	
	Disconnect No CloseSPITProject is sent so that the clients keep their current state.

SPIT Server save project

The SPIT server saves all existing SPIT types and SPIT objects. The user may need to clean up before saving.

Appendix: SAD Server Auto Detect
(SPIT Server Auto Detect)

(all characters are ISO 8859-15 encoded)

Client
The client periodically sends the following content via UDP:
#SAD##,##spit#

each to:
all IP4 broadcast addresses of the computer
und

to the multicast IP6 address "ff02 :: 1"

Content:
#SAD# is a Precode

#,# is the LineSeparator

#spit# is an identifier for SPIT server

Server

The server joins the multicast group "ff02 :: 1".

Once a UDP packet is received by a SPIT client:

1. the server recognizes the IP address and the port from the UDP packet (UDP protocol)
2. The server sends a response with the following content to the IPAddress / Port of the client:

#SAD##,##spit##,#s*erver task name*#,#s*erverid*#,#*computername*#,#s*erverport*#,#*clientIP*

Content:

#SAD# is a Precode

#,# is the LineSeparator

#spit# is an identifier for SPIT server

server task name is the talking name für #spit#

serverid is an ID for the scope of the server (usually: #spit#)

computername is the computer name the server is running on

serverport is the port through which the client can reach the server

clientIP is the IP address of the client through which the client has reached the server

Annotation:
The liveShow SPIT server uses the port: 15120

Content