1. CITP protocol suite specification

1.1 History

2007-09-16	Revised documentation into a single document.
2007-09-28	Added first comments for MSEX revision, highlighted in red.
2008-01-25	Cleaned up MSEX 1.1 changes for element libraries.
2008-05-28	Minor corrections and clarifications in MSEX.
2008-08-21	Completed MSEX element types 4 - 8, accompanied by the Generic Element Information packet.
2008-10-11	Added BSR E1.31 to the DMX connection strings table.
2008-11-08	Added first OMEX packet suggestions.
2009-02-14	OMEX packet suggestion update and general revision of the introduction section.
2009-02-18	Removed deprecation note of PINF/PNam as it does have its use (with clarifying comments).
2009-05-17	Added note regarding problems with MSEX/GLEI message.
2009-06-23	Clarified the note regarding contiguous element identifiers.
2009-11-27	Added first draft of MSEX 1.2 extensions.
2010-06-12	MSEX 1.2 finalized.
2010-08-29	Clarified the role of the PNam message.
2011-07-20	Added FINF SPos and Posi message suggestions.
2012-05-02	Clarified MSEX image formats.
2012-07-25	Corrected MSEX ELTh and EThn ThumbnailBuffer type missing []. Minor style fixes.
2012-08-20	Added new 3D mesh MSEX element type and fixed minor typos.

1.2 Introduction

The CITP (Controller Interface Transport Protocol) is a dual layer protocol suite that has been designed for communication between lighting consoles, media servers and visualizers. This document describes how it is used on top of an IP stack, but the packets could easily be used over other media as well, such as USB links.

The top layer, CITP, consists of a single message header with content information and support for fragmentation and stream synchronization. This message header is used in the beginning of all CITP protocol suite packets.

The second layer of CITP consists of the PINF, SDMX, FPTC, FSEL, FINF, MSEX and OMEX protocols. Each of these have been designed for a specific purpose, but some of them are closely related (such as FPTC, FSEL and FINF that all operate on a given set of lighting fixtures). Any manufacturer can extend the CITP protocol at the second layer level using a non-reserved layer identifier.

1.3 Lighting console behaviour

Datagram (UDP) socket, port 4809, joined to multicast address 224.0.0.180:
- Regularly send a CITP/PINF/PLoc message with no listening port.
- Receive CITP/PINF/PLoc messages to be aware of available visualizers and media servers.
- Connect either automatically or on user demand to an available visualizer and/or media server.
- Receive CITP/MSEX/StFr Stream Frame video content from media server video subscriptions.

For all TCP connections to a media server:
- Send CITP/PINF/PNam message immediately after connecting.
- Send CITP/MSEX/CInf Client Information message immediately after connecting.
- Receive CITP/MSEX/SInf Server Information and CITP/MSEX/LSta layer status messages.

- Send CITP/MSEX/GELI Get Element Library Information message(s) and initiate element library update. Request all libraries of relevant type to the media server in question (as identified by the CITP/PINF/PLoc Name field).
- Send CITP/MSEX/GVsr Get Video Sources message to retrieve information about available video feeds.

For all TCP connections to a visualizer:
- Send CITP/PINF/PNam message immediately after connecting.
- Send a CITP/SDMX/UNam Universe Name for each DMX universe controlled to provide display names.
- Either Send CITP/SDMX/ChBk Channel Block messages with DMX data,
- or Send a CITP/SDMX/SXSr Set External Source message to specify an alternative DMX transfer method.
- Receive CITP/SDMX/ChBk messages for "autofocus" purposes.
- Send and receive CITP/FPTC, CITP/FSEL and CITP/FINF messages when fit.

1.4 Media server behaviour

TCP listening socket on any (known) port:
- Accept incoming connections from any lighting console or visualizer. If the media server can only handle a limited number of simultaneous connections then it should actively refuse any further connection attempts.

Datagram (UDP) socket, port 4809, joined to multicast address 224.0.0.180:
- Regularly send a CITP/PINF/PLoc message containing the port on which the listening socket is listening.

For all accepted incoming TCP connections from a lighting console or visualizer: *Mbox does not send PNam*
- Send CITP/PINF/PNam message immediately after connecting.
- Send a CITP/MSEX/SInf Server Information message (MSEX 1.0 or MSEX 1.1).
- Receive CITP/MSEX/CInf Client Information message from lighting console and respond with a CITP/MSEX/SInf Server Information message (MSEX 1.2 or later).
- Regularly send a CITP/MSEX/LSta Layer Status message.
- Receive and respond to CITP/MSEX element library browsing messages.
- Send CITP/MSEX element library information messages on library changes.
- Receive and respond to CITP/MSEX video stream browsing and subscription messages.

1.5 Visualizer behaviour

TCP listening socket on any (known) port:
- Accept incoming connections from any lighting console.

Datagram (UDP) socket, port 4809, joined to multicast address 224.0.0.180:
- Regularly send a CITP/PINF/PLoc message containing the port on which listening socket is listening.
- Receive CITP/PINF/PLoc message to be aware of available media servers.
- Connect either automatically or on user demand to an available media server.
- Receive CITP/MSEX/StFr Stream Frame video content from media server video subscriptions.

For all accepted incoming TCP connections from a lighting console:
- Send CITP/PINF/PNam message immediately after connecting.
- Receive CITP/SDMX/UNam Universe Name messages.
- Receive CITP/SDMX/ChBk messages with DMX data.
- Optionally support CITP/SDMX/SXSr messages and receive DMX data over other protocols.
- Send CITP/SDMX/ChBk messages for "autofocus" purposes.
- Send and receive CITP/FPTC, CITP/FSEL and CITP/FINF messages when fit.

For all TCP connections to a media server:
- Send CITP/PINF/PNam message immediately after connecting.
- Send CITP/MSEX/CInf Client Information message immediately after connecting.
- Receive CITP/MSEX/SInf Server Information and CITP/MSEX/LSta layer status messages.
- Send CITP/MSEX/GVSr Get Video Sources message to retrieve information about available video feeds.

1.6 Device status / Operations management servers

Work in progress.

1.7 General IP notes and hints

PC based applications must choose listening ports and set socket address reusability flags as necessary to avoid blocking eachother when run on the same network interface. Achieve this by calling listen() for port 0 and

retrieving the port chosen by the operating system with getsockname(), and by setting the SO_REUSEADDR (and possibly also SO_REUSEPORT) option on the multicast socket before joining the multicast address.

To join a multicast address, use setsockopt() with IPPROTO_IP and IP_ADD_MEMBERSHIP.

2. Definitions

These specifications target lighting software developers. It contains C style types and annotation, although mostly on a pseduo-code level.

2.1 Data types

All structures and variables of CITP use little endian byte order (least significant byte first, "PC standard") and 1-byte packing of C-structures.

```
int8, int16, int32, int64      // 8-bit, 16-bit, 32-bit and 64-bit signed integers
uint8, uint16, uint32, uint64  // 8-bit, 16-bit, 32-bit and 64-bit unsigned integers
ucs1, ucs2                     // 8-bit and 16-bit unicode characters (character types
                                  correspond to uint8 and uint16)
float32                        // 32-bit IEEE floating point (8-bit exp., 23-bit mant.)
float64                        // 64-bit IEEE floating point (11-bit exp., 52-bit mant.)
```

Open arrays of ucs1 or ucs2 are null terminated strings.

2.2 Cookies

The Cookie (and ContentType) fields can be found in CITP headers in both layers. The constant values of these fields are documented using string notation, for instance "CITP" for the CITP header Cookie field. This should be interpreted as sending 'C','I','T','P' over the network.

2.3 DMX Connection Strings

Instead of defining constants and fixed field for various DMX source protocols, a connection string approach is used instead. The following table illustrates well-defined DMX connection strings in CITP:

Protocol	Format	Examples
ArtNet	"ArtNet/<net>/<universe> /<channel>"	"ArtNet/0/0/1" - The first channel of the first universe on the first network.
Avab IPX	"AvabIPX/<net>/<universe> /<channel>"	"AvabIPX/0/0/1" - The first channel of the first universe on the first network.
BSR E1.31	"BSRE1.31/<universe>/<channel>"	"BSRE1.31/0/1" - The first channel of the first universe.
ETC Net2	"ETCNet2/<channel>"	"ETCNet2/1" - The first ETCNet2 channel.

3. CITP, base layer

The base layer as such does not define any packages, it merely adds a header that encapsulate all messages.

3.1 Header definitions

3.1.1 The CITP header

The CITP layer provides a standard, single, header used at the start of all CITP packets:

```
struct CITP_Header
{
    uint32      Cookie;                      // Set to "CITP".
    uint8       VersionMajor;                // Set to 1.
    uint8       VersionMinor;                // Set to 0.
    union
    {
        uint16          RequestIndex;        // See below
        uint16          InResponseTo;        // See below
    };
    uint32      MessageSize;                 // The size of the entire message, including
                                             //  this header.
    uint16      MessagePartCount;            // Number of message fragments.
    uint16      MessagePart;                 // Index of this message fragment (0-based).
    uint32      ContentType;                 // Cookie identifying the type of contents
                                             //  (the name of the second layer).
};
```

RequestIndex/InResponseTo: These allow request/response message pairs to be better associated and is particularly useful for debugging purposes. A node that sends request messages (such as a Lighting Console requesting info from a Media Server) should maintain a request counter, and increment this with every request message sent. When the other side sends a response to a specific request message, it should set this field to the same value as was found in the corresponding request message. The value of 0 is taken to mean 'ignored', so proper RequestIndex values should start at 1 (and wrap back around to 1, avoiding the 0 'ignored' value). This was introduced for MSEX 1.2 and was previously a reserved 2-byte alignment field.

Note: Receipt of any unrecognised or unsupported messages must not be treated as an error condition.

4. CITP/PINF, Peer Information layer

The Peer Information layer is used to exchange peer information, both when connected and when locating peers on the network.

The PINF/PNam message was previousley broadcasted on UDP port 4810, but that behaviour has now been deprecated. Instead, the PINF/PLoc message is multicasted on address 224.0.0.180, port 4809. Do note that it is a good idea to send a PINF/PName message as a first over any established connection!

4.1 Header definitions

4.1.1 The PINF header

The PINF layer provides a standard, single, header used at the start of all PINF packets:

```
struct CITP_PINF_Header
{
    CITP_Header CITPHeader;      // The CITP header. CITP ContentType is "PINF".
    uint32      ContentType;     // A cookie defining which PINF message it is.
};
```

4.2 Message definitions

4.2.1 PINF / PNam - Peer Name message

The PeerName message provides the receiver with a display name of the peer. In early implementations of CITP, the PNam message was broadcasted as a means of locating peers - now the PLoc message is multicasted instead. The PNam message is still useful though, as a message transferred from a peer connected to a listening peer.

```
struct CITP_PINF_PNam
{
    CITP_PINF_Header  CITPPINFHeader;   // The CITP PINF header. PINF ContentType is "PNam".
    ucs1              Name[];           // The display name of the peer (null terminated).
                                           This could be anything from a
                                        // user defined alias for the peer of the name of the
                                           product, or a combination.
};
```

4.2.2 PINF / PLoc - Peer Location message

The PeerLocation message provides the receiver with connectivity information. If the ListeningTCPPort field is non-null, it may be possible to connect to the peer on that port using TCP. If the peer can only handle a limited number of simultaneous connections, then additional connections should be actively refused. The Type field instructs the receiver what kind of peer it is and the Name and State fields provide display name and information.

```
struct CITP_PINF_PLoc
{
    CITP_PINF_Header  CITPPINFHeader;    // The CITP PINF header. PINF ContentType is "PLoc".
    uint16            ListeningTCPPort;  // The port on which the peer is listening for
                                            incoming TCP connections. 0 if not listening.
    ucs1              Type[];            // Can be "LightingConsole", "MediaServer",
                                            "Visualizer" or "OperationHub".
    ucs1              Name[];            // The display name of the peer. Correspons to the
                                            PINF/PNam/Name field.
    ucs1              State[];           // The display state of the peer. This can be any
                                            descriptive string presentable to the user such
                                            as "Idle", "Running" etc.
};
```

5. CITP/SDMX, Send DMX layer

The SDMX layer is used to transmit DMX information. CITP supports transmitting a single - wide - universe of DMX channels with at most 65536 channels. It also supports designating an alternative DMX source such as ArtNet or ETCNet2 (see "connection strings" in the Definitions section).

5.1 Header definitions

5.1.1 The SDMX header

The SDMX layer provides a standard, single, header used at the start of all SDMX packets:

```
struct CITP_SDMX_Header
{
    CITP_Header CITPHeader;      // CITP header. CITP ContentType is "SDMX".
    uint32      ContentType;     // Cookie defining which SDMX message it is.
};
```

5.2 Message definitions: Transfer of DMX channel levels

5.2.1 SDMX / EnId - Encryption Identifier message

The EncryptionIdentifier message is used to agree on encryption schemes when transferring DMX channels. The usage of this message depends completely on the peers communicating it; the contents and results of this message is not part of the CITP specification - it must be agreed upon a priori.

```
struct CITP_SDMX_EnId
{
    CITP_SDMX_Header CITPSDMXHeader;   // CITP SDMX header. SDMX ContentType is "EnId".
    ucs1             Identifier[];     // Encryption scheme identifier.
};
```

5.2.2 SDMX / UNam - Universe Name message

The Universe Name message can be sent by a DMX transmitting peer in order to provide the other end with a displayable name of a universe.

```
struct CITP_SDMX_UNam
{
    CITP_SDMX_Header CITPSDMXHeader;   // CITP SDMX header. SDMX ContentType is "UNam".
    uint8            UniverseIndex;    // 0-based index of the universe.
    ucs1             UniverseName[];   // Name of the universe.
};
```

5.2.3 SDMX / ChBk - Channel Block message

The Channel Block message transmits raw DMX levels to the recipient. How to handle Blind DMX levels is up to the recipient, but the recommended procedure for a visualizer is to switch over to blind DMX whenever such is present and to revert back after some short timeout when it is no longer transmitted.

```
struct CITP_SDMX_ChBk
{
    CITP_SDMX_Header CITPSDMXHeader;   // CITP SDMX header. SDMX ContentType is "ChBk".
    uint8            Blind;            // Set to 1 for blind preview dmx, 0 otherwise.
    uint8            UniverseIndex;    // 0-based index of the universe.
    uint16           FirstChannel;     // 0-based index of first channel in the universe.
    uint16           ChannelCount;     // Number of channels.
    uint8            ChannelLevels[];  // Raw channel levels.
};
```

5.3 Message definitions: Alternate DMX source management

5.3.1 SDMX / SXSr - Set External Source message

The Set External Source message can be sent as an alternative to the ChBk message above, when DMX should be tapped from another protocol on the other end. In the event of handling multiple universes, the external source specified should be treated as the base universe of a consecutive series.

```
struct CITP_SDMX_SXSr
{
    CITP_SDMX_Header    CITPSDMXHeader;         // CITP SDMX header. SDMX ContentType is "SXSr".
    ucs1                ConnectionString[];     // DMX-source connection string. See DMX
                                                   Connection Strings in Definitions.
};
```

6. CITP/FPTC, Fixture patch layer

The Fixture Patch layer is used to communicate fixture existence and patch information. Fixtures are identified by 16-bit unsigned integeres with a range of valid values between 1 and 65535. In most consoles this value maps directly to a "Channel", "Unit" or "Device".

The FPTC layer is built on the following design decisions:

- Unpatched fixtures do not exist from the FPTC layers's point of view. When a fixture is unpatched using the UnPatch message, it is deleted and seizes to exist. However, the fixture may continue to live in the visualizer or the console, without association to a universe. Whenever the fixture is associated with a universe again, it is reintroduced through the Patch message.
- When a fixture is repatched (ie moved to another channel or universe) it does not pass through an unpatched state.
- In the visualizer, it may possible to change the mode of a fixture. Different modes for one fixture usually use different amounts of channels, however sometimes a different mode only changes the interpretation of one or more control channels. When a mode is changed in the visualizer, an unpatch message is not sent, only a new patch message. If the new mode consumes a different amount of channels, this can be told by the ChannelCount field of the patch message. If it does not, there is no way of telling.
- A fixture can change its patch and mode, but never its make or name. The visualizer attempts to map the fixture make and name against its library.
- Fixture identifiers must be persistent. When both the visualizer and the console have reloaded a pair of matching projects, the fixture identifiers must still be the same.
- When a project is closed on either side, fixtures are not unpatched. The same applies to when a universe in the visualizer is deleted or unassociated with a console.
- No synchronisation mechanism exists in CITP, which communicates project closing/opening information. This must be handled by the user by opening and closing matching projects simultaneously.
- When the visualizer or console takes automatic actions as a result of incoming patch messages, it must not result in an echo.

6.1 Header definitions

6.1.1 The FPTC header

The FPTC layer provides a standard, single, header used at the start of all FPTC packets:

```
struct CITP_FPTC_Header
{
    CITP_Header CITPHeader;     // The CITP header. CITP ContentType is "FPTC".
    uint32      ContentType;    // A cookie defining which FSEL message it is.
    uint32      ContentHint;    // Content hint flags.
                                // 0x00000001   Message part of a sequence of messages.
                                // 0x00000002   Message part of and ends a sequence of
                                //                  messages.
};
```

6.2 Message definitions

6.2.1 FPTC / Ptch - Patch message

Patch messages are sent when fixtures are introduced or repatched. The patch message contains the identifier of the fixture added, the sender fixture (library) type make and name of the fixture added and the patching information..

```
struct CITP_FPTC_Ptch
{
    CITP_FPTC_Header CITPFPTCHeader;    // The CITP FPTC header. FPTC ContentType
                                        //   is "Ptch".
    uint16           FixtureIdentifier; // Fixture identifier.
    uint8            Universe;          // Patch universe (0-based).
    uint8            Reserved[1];       // 4-byte alignment.
    uint16           Channel;           // Patch channel (0-based).
    uint16           ChannelCount;      // Patch channel count (1-512).
    ucs1             FixtureMake[];      // Fixture make (only null if omitted).
    ucs1             FixtureName[];      // Fixture name (never omitted).
};
```

6.2.2 FPTC / UPtc - Unpatch message

Unpatch messages are sent when fixtures are deleted or unpatched. The unpatch message only contains the identifiers of the fixtures removed. An empty fixture identifier array indicates complete unpatching..

```
struct CITP_FPTC_UPtc
{
    CITP_FPTC_Header  CITPFPTCHeader;         // The CITP FPTC header. FPTC ContentType
                                              //  is "UPtc".
    uint16            FixtureCount;           // Fixture count (0 to unpatch all).
    uint16            FixtureIdentifiers[];   // Fixture identifiers
};
```

6.2.3 FPTC / SPtc - SendPatch message

The SendPatch message instructs the receiver to send Patch messages in response, one for each fixture specified in the FixtureIdentifiers array. If no fixture identifiers are specified, the entire Patch should be transferred in response. This procedure can be used for testing the existence of fixtures on the remote side or to synchronize the entire patch information..

```
struct CITP_FPTC_SPtc
{
    CITP_FPTC_Header  CITPFPTCHeader;         // The CITP FPTC header. FPTC ContentType
                                              //  is "SPtc".
    uint16            FixtureCount;           // Fixture count (0 to request all).
    uint16            FixtureIdentifiers[];   // Fixture identifiers.
};
```

7. CITP/FSEL, Fixture Selection layer

The Fixture Selection layer is used to carry fixture selection information. Fixture identification is discussed in the CITP/FPTC section.

7.1 Header definitions

7.1.1 The FSEL header

The FSEL layer provides a standard, single, header used at the start of all FSEL packets:

```
struct CITP_FSEL_Header
{
    CITP_Header CITPHeader;     // The CITP header. CITP ContentType is "FSEL".
    uint32      ContentType;    // A cookie defining which FSEL message it is.
};
```

7.2 Message definitions

7.2.1 FSEL / Sele - Select message

The Select message instructs the receive to select a number of fixtures. If the Complete field is non-zero, only the fixtures identified in the message should be selected and all others should be deselected, thus achieving a full synchronization.

```
struct CITP_FSEL_Sele
{
    CITP_FSEL_Header  CITPFSELHeader;       // The CITP FSEL header. FSEL ContentType
                                            //  is  "Sele".
    uint8             Complete;             // Set to non-zero for complete selection
    uint8             Reserved[1];          // 4-byte alignment
    uint16            FixtureCount;         // Greater than 0
    uint16            FixtureIdentifiers[]; // Fixture identifiers
};
```

7.2.2 FSEL / DeSe - Deselect message

The Deselect message acts similarly to the Select message. However, a Deselect message deselects the fixture specified, rather than selectin them. A Deselect with no fixture specified should deselect all fixtures.

```
struct CITP_FSEL_DeSe
{
    CITP_FSEL_Header  CITPFSELHeader;       // The CITP FSEL header. FSEL ContentType
                                            //  is  "DeSe".
    uint16            FixtureCount;         // 0 for complete deselection
    uint16            FixtureIdentifiers[]; // Fixture identifiers
};
```

8. CITP/FINF, Fixture Information layer

The Fixture Information layer is used to carry additional fixture information. Fixture identification is discussed in the CITP/FPTC.

8.1 Header definitions

8.1.1 The FINF header

The FINF layer provides a standard, single, header used at the start of all FINF packets:

```
struct CITP_FINF_Header
{
    CITP_Header CITPHeader;    // The CITP header. CITP ContentType is "FINF".
    uint32      ContentType;   // A cookie defining which FINF message it is.
};
```

8.2 Message definitions

8.2.1 FINF / SFra - Send Frames message

This messages informs the receiver to send frame messages for the specified fixtures.

```
struct CITP_FINF_SFra
{
    CITP_FINF_Header  CITPFINFHeader;        // The CITP FINF header. FINF ContentType
                                             //   is "SFra".
    uint16            FixtureCount;          // Fixture count (0 to request all).
    uint16            FixtureIdentifiers[];  // Fixture identifiers.
};
```

8.2.2 FINF / Fram - Frames message

This messages informs the receiver about the filters & gobos of a fixture.

```
struct CITP_FINF_Fram
{
    CITP_FINF_Header  CITPFINFHeader;      // The CITP FINF header. FINF ContentType
                                           //   is "Fram".
    uint16            FixtureIdentifier;   // Fixture identifier.
    uint8             FrameFilterCount;    // Number of filters in the FrameNames field.
    uint8             FrameGoboCount;      // Number of gobos in the FrameNames field.
    ucs1              FrameNames[];        // List of (first) filters and (last) gobos,
                                           //   newline separated (\n) & null terminated.
                                           //   Contains at least the null.
};
```

8.2.3 FINF / SPos - Send Position message   PRELIMINARY

This message informs the receiver to send position messages for the specified fixtures.

```
struct CITP_FINF_SPos
{
    CITP_FINF_Header  CITPFINFHeader;        // The CITP FINF header. FINF ContentType
                                             //   is "SPos".
    uint16            FixtureCount;          // Fixture count (0 to request all).
    uint16            FixtureIdentifiers[];  // Fixture identifiers.
};
```

8.2.4 FINF / Posi - Position message   PRELIMINARY

This message informs the receiver about the position of the specified fixture(s). Coordinates are expressed in metres.

```
struct CITP_FINF_Posi
{
    CITP_FINF_Header  CITPFINFHeader;       // The CITP FINF header. FINF ContentType
                                            //   is "Posi".
    uint16            FixturePositionCount; // The number of FixturePosition blocks.
    struct FixturePosition
    {
```

```
        uint16          FixtureIdentifier;      // Fixture identifier.
        float32         PositionX;              // Position X axis component.
        float32         PositionY;              // Position Y axis component.
        float32         PositionZ;              // Position Z axis component.
    }[];
};
```

8.2.5 FINF / LSta - Live status message   PRELIMINARY

This message can be sent in any direction on a regular basis. The flag mask and flag fields size is dynamic in order to allow future expansion without redifinition.

```
struct CITP_FINF_LSta
{
    CITP_FINF_Header  CITPFINFHeader;       // The CITP FINF header. FINF ContentType
                                            //    is "LSta".
    uint16            LiveStatusCount;      // The number of LiveStatus blocks.
    uint8             FlagSize;             // Number of bytes in the flag field.
    struct LiveStatus
    {
        uint16        FixtureIdentifier;    // Fixture identifier.
        uint8         FlagMask[FlagSize];   // Flag mask.
        uint8         Flags[FlagSize];      // Flags.
                                            //     0x01 TBD
    }[];
};
```

9. CITP/OMEX, Operations Management layer
PRELIMINARY

The Operations Management EXtensions layer is used for metadata communication.

9.1 Header definitions

The OMEX layer provides a standard, single, header used at the start of all OMEX packets:

```
struct CITP_OMEX_Header
{
    CITP_Header        CITPHeader;         // CITP header. CITP ContentType is "OMEX".
    uint8              VersionMajor;       // Set to 1.
    uint8              VersionMinor;       // Set to 0.
    uint32             ContentType;        // Cookie defining which OMEX message it is.
};
```

9.2 Message definitions: DMX device status signalling

Status signalling of DMX devices is ....

9.2.1 OMEX / SDDS - Signal DMX Device Status

Sent to signal a status for one or more devices. A status is identified by a short string which is used again when clearing or updating the status (by sending a new SDDS message). It is typically a short string, such as "Offline", "On fire" or "Lamp fail".

```
struct CITP_OMEX_SDDS
{
    CITP_OMEX_Header   CITPOMEXHeader;     // CITP OMEX header. OMEX ContentType
                                           //   is "SDDS".
    ucs2               StatusIdentifier[]; // Displayable status tag.
    uint8              Severity;           // 50 = Info, 100 = Warning, 150 = Error
    ucs2               Category[];         // Category identifier.
    ucs2               ShortText[];        // Short descriptive text.
    ucs2               LongText[];         // Long descriptive text.
    uint16             DeviceCount;        // The number of following device information
                                           //   blocks for which to set this status.

    struct DeviceInformation
    {
        ucs1           DMXConnectionString; // A DMX connection string.
    };
};
```

9.2.2 OMEX / CDDS - Clear DMX Device Status

Sent to clear a specific status from a set of devices. It is not necessary that the status is cleared from all deviced that have it set, but it is possible. If a status clear is requested for a device that is not known to have status, the request is silently ignored.

```
struct CITP_OMEX_CDDS
{
    CITP_OMEX_Header   CITPOMEXHeader;     // CITP OMEX header. OMEX ContentType
                                           //   is "CDDS".
    ucs2               StatusIdentifier[]; // Displayable status tag.
    uint16             DeviceCount;        // The number of following device information
                                           //   blocks.

    struct DeviceInformation
    {
        ucs1           DMXConnectionString; // A DMX connection string.
    };
};
```

10. CITP/MSEX, Media Server Extensions layer

The Media Server EXtensions layer is used for communication with Media Servers.

For information about how peers find eachother and connect, see the Connectivity section. Typically all packets are sent over a peer-to-peer TCP socket connection, except for the MSEX/StFr message which is sent over the multicast address for all to process.

MSEX Versions

Currently acknowledged versions of MSEX are 1.0, 1.1 and 1.2. During a session, the appropriate MSEX version that is common to both sides must be established and used for all communication - different versions cannot be mixed in a single session. See the MSEX/SInf and MSEX/CInf messages also regarding supported version signalling.

Prior to MSEX 1.2 it was expected that all client and server implementations check the MSEX version of all received messages to ensure that the message format is acceptable. Starting with MSEX 1.2 this is a mandatory requirement.

There is no requirement for an implementation of a specific MSEX version to support any previous MSEX versions, for this reason the version returned by the MSEX/SInf message must be used for all communication by both sides.

Establishing communications

Prior to MSEX 1.2, a media server was expected to send a MSEX/SInf Server Information message immediately after connecting to a lighting console or visualiser. This approach has the drawback that the MSEX/SInf message format has to be fixed since the media server is unaware of what MSEX version(s) the other side supports. Starting with MSEX 1.2, the lighting console or visualiser must send a MSEX/CInf Client Information message to the server immediately after connecting, and the server will respond with a version 1.2 or later MSEX/SInf message.

NB: Although the MSEX/CInf message format must be fixed, provision has been made to allow extra data to be appended as a future-proofing measure.

Highest Common MSEX Version

For MSEX 1.2 and later, the server must establish the Highest Common MSEX Version when a MSEX/CInf is received from a newly connected lighting console or media server. This is the highest MSEX version that is supported on both sides, and must be used for all unsolicited messages, such as MSEX/SInf, MSEX/LSta and MSEX/ELUp. The Highest Common MSEX Version is at least 1.2.

Mandatory messages

Implementations can choose to implement a subset of MSEX messages to suit their needs, but some messages are essential for correct interoperation and are marked as mandatory. The mandatory messages are:

1. CInf - Client Information message
2. SInf - Server Information message
3. LSta - Layer Status message
4. Nack - Negative acknowledge message

Image formats

MSEX supports three image formats for thumbnail and video stream frames;

- RGB8 - a raw array of 8-byte RGB triples (this is **not** BMP). In MSEX 1.0 the byte order was BGR, but from MSEX 1.1 the byte order is RGB.
- PNG - the well known file format.
- JPEG - the well known file format (which does **not** include EXIF).

10.1 Header definitions

10.1.1 The MSEX header

The MSEX layer provides a standard, single, header used at the start of all MSEX packets:

```
struct CITP_MSEX_Header
{
```

```
    CITP_Header         CITPHeader;              // CITP header. CITP ContentType is "MSEX".
    uint8               VersionMajor;            // See below.
    uint8               VersionMinor;            // See below.
    uint32              ContentType;             // Cookie defining which MSEX message it is.
};
```

The ContentType cookie identifies the specific MSEX message type (e.g. "GETh" for Get Element Thumbnail etc.). If an implementation receives a message with an unrecognised cookie it must silently discard the message and not treat this as an error condition. This is to allow the specification to continue to evolve over time.

10.2 Message definitions: Communication establishment

10.2.1 MSEX / CInf - Client Information message

The Client Information message advises the media server of which versions of MSEX are supported by the client. This message is mandatory and must be sent by the client to the media server immediately after establishing a connection. The media server will examine the list of supported versions and establish the Highest Common MSEX Version defined above.

```
struct CITP_MSEX_CInf
{
    CITP_MSEX_Header    CITPMSEXHeader;          // CITP MSEX header. MSEX ContentType is
                                                 //    "CInf". Version is 1.2.
    uint8               SupportedMSEXVersionsCount; // Number of following MSEX version pairs.
    uint16              SupportedMSEXVersions[]; // Each 2 byte value is MSB = major MSEX version,
                                                 //    LSB = minor MSEX version.
    uint                FutureMessageData[];     // A hint that future versions of this message
                                                 //    may contain trailing data.
};
```

Note: The format of this message up to FutureMessageData cannot be changed in future versions of MSEX, since the client does not yet know which versions the media server will understand. Future versions can be defined however, but they must preserve the format of the previous version and only insert new fields immediately before the FutureMessageData field.

10.2.2 MSEX / SInf - Server Information message

The Server Information message provides the receiver with product and layer information. This message is mandatory. If the media server supports MSEX 1.0 or 1.1, it should send the v1.0 SInf message immediately after accepting an incoming connection from a lighting console or visualiser. If the media server supports MSEX 1.2 or later, it must send a SInf message in response to a MSEX/CInf message received from the connected client, and the format of that SInf message must match the Highest Common MSEX Version.

```
struct CITP_MSEX_1.0_SInf
{
    CITP_MSEX_Header    CITPMSEXHeader;          // CITP MSEX header. MSEX ContentType
                                                 //    is "SInf". Version is set to 1.0.
    ucs2                ProductName[];           // Display name of the product.
    uint8               ProductVersionMajor;     // Major version number of the product.
    uint8               ProductVersionMinor;     // Minor version number of the product.
    uint8               LayerCount;              // Number of following layer information blocks.
    struct LayerInformation
    {
        ucs1            DMXSource[];             // DMX-source connection string. See DMX
                                                 //    Connection Strings in Definitions.
    };
};
```

A MSEX 1.2 or later version of the MSEX/SInf message is sent in response to a MSEX/CInf Client Information message received from the lighting console or visualiser. The MSEX version used for this message is the Highest Common MSEX Version (described in under MSEX Versions, above).

```
CITP_MSEX_1.2_SInf
{
    CITP_MSEX_Header    CITPMSEXHeader;          // CITP MSEX header. MSEX ContentType is "SInf".
                                                 //    Version is at least 1.2 and is the highest common
                                                 //    version supported by both server and client.
    ucs1                UUID[36];                // A standard 36 character UUID that uniquely
                                                 //    identifies this media server (see below).
    ucs2                ProductName[];           // Display name of the product.
    uint8               ProductVersionMajor;     // Major version number of the product.
    uint8               ProductVersionMinor;     // Minor version number of the product.
    uint8               ProductVersionBugfix;    // Bugfix version number of the product.
```

```
    uint8              SupportedMSEXVersionsCount;  // Number of following MSEX version pairs.
    uint16             SupportedMSEXVersions[];     // Each 2 byte value is MSB = major MSEX version,
                                                    //   LSB = minor MSEX version (see below)

    uint16             SupportedLibraryTypes;       // Bit-encoded flagword that identifies which library
                                                    //   types are provided by the media server (e.g. this
                                                    //   would be 1 for Media, 2 for Effects, 4 for Cues etc.).
    uint8              ThumbnailFormatsCount;       // Number of following thumbnail format cookies
    uint32             ThumbnailFormats[];          // Must include "RGB8", but can also include "JPEG" and
                                                    //   "PNG " (see below)
    uint8              StreamFormatsCount;          // Number of following stream format cookies
    uint32             StreamFormats[];             // Must include "RGB8", but can also include "JPEG" and
                                                    //   "PNG " (see below)
    uint8              LayerCount;                  // Number of following layer information blocks.
    struct LayerInformation
    {
        ucs1           DMXSource[];                 // DMX-source connection string. See DMX
                                                    //   Connection Strings in Definitions.

    };
};
```

UUID: This is required so that a lighting console can reliably associate cached information (e.g. thumbnails) with a specific Media Server when starting a new session, in the case where there may be more than 1 of a specific type of Media Server. The UUID is a string of 36 hexadecimal characters grouped as "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx", e.g. "550e8400-e29b-41d4-a716-446655440000"

SupportedMSEXVersions: Media Servers that support a specific version of MSEX are not required to support all earlier versions, so this identifies which specific versions are provided.

Format arrays: the order that formats are presented in the ThumbnailFormats and StreamFormats arrays can indicate the Media Server's format preference, the first being the best and the last being the least convenient. Only the "PNG " format can support transparency and it is recommended that all implementations support this format.

10.2.3 MSEX / Nack Negative Acknowledge message

The Negative Acknowledge message is sent in response to any unsupported or unrecognised message received by the Media Server. As with all response messages, the InResponseTo field of the CITP_Header should be set to the same value as the RequestIndex in the corresponding request message. The ReceivedContentType cookie is a copy of the ContentType field in the CITP_MSEX_Header of the corresponding request message. This message is mandatory for MSEX 1.2 and later.

```
struct CITP_MSEX_Nack
{
    CITP_MSEX_Header   CITPMSEXHeader;              // CITP MSEX header. MSEX ContentType
                                                    //   is "Nack" and version is 1.2.
    uint32             ReceivedContentType          // MSEX message type of the message being NACKed
                                                    //   (e.g. "GELT" if the Media Server does not
                                                    //   support library thumbnails)

};
```

10.3 Message definitions: Layer information

10.3.1 MSEX / LSta - Layer Status message

The LayerStatus message is sent at a regular interval (suggestion: 4 times / second) to provide the receiver with live status information. This message is mandatory.

```
struct CITP_MSEX_1.0_LSta
{
    CITP_MSEX_Header   CITPMSEXHeader;          // CITP MSEX header. MSEX ContentType
                                                //   is "LSta" and version is 1.0.
    uint8              LayerCount;              // Number of following layer information
                                                //   blocks.
    struct LayerStatus
    {
        uint8          LayerNumber;             // 0-based layer number, corresponding to
                                                //   the layers reported in the SInf message.
        uint8          PhysicalOutput;          // Current physical video output index,
                                                //   0-based.
        uint8          MediaLibraryNumber;      // Current media library number.
        uint8          MediaNumber;             // Current media number.
        ucs2           MediaName[];             // Current media name.
```

```
        uint32         MediaPosition;          // Current media position (in frames).
        uint32         MediaLength;            // Current media length (in frames).
        uint8          MediaFPS;               // Current media resolution in frames per
                                               //     second.
        uint32         LayerStatusFlags;       // Current layer status flags
                                               //     0x0001 MediaPlaying
    }[];
};

struct CITP_MSEX_1.2_LSta
{
    CITP_MSEX_Header   CITPMSEXHeader;         // CITP MSEX header. MSEX ContentType
                                               //   is "LSta" and version is 1.2.
    uint8              LayerCount;             // Number of following layer information
                                               //   blocks.
    struct LayerStatus
    {
        uint8          LayerNumber;            // 0-based layer number, corresponding to
                                               //   the layers reported in the SInf message.
        uint8          PhysicalOutput;         // Current physical video output index,
                                               //   0-based.
        uint8          MediaLibraryType;       // Library content type.
        MSEXLibraryId  MediaLibraryId;         // Current media library ID. (defined later in
                                               //   this specification)
        uint8          MediaNumber;            // Current media number.
        ucs2           MediaName[];            // Current media name.
        uint32         MediaPosition;          // Current media position (in frames).
        uint32         MediaLength;            // Current media length (in frames).
        uint8          MediaFPS;               // Current media resolution in frames per
                                               //   second.
        uint32         LayerStatusFlags;       // Current layer status flags
                                               //     0x0001 MediaPlaying
                                               //     0x0002 MediaPlaybackReverse
                                               //     0x0004 MediaPlaybackLooping
                                               //     0x0008 MediaPlaybackBouncing
                                               //     0x0010 MediaPlaybackRandom
                                               //     0x0020 MediaPaused
    }[];
};
```

10.4 Message definitions: Element libraries and element information

In MSEX 1.0, there is a finite set of at most 256 libraries, each containing a finite set of at most 256 elements. This is designed to match the common media server layout of 2 dmx channels identifying the library and item respectively.

In MSEX 1.1 however, there is a finite set of at most 3 library levels with at most 256 elements each. Libraries are identified using a library identifier, a 4-byte integer divided into four 1-byte fields. When it's Level byte is set to 0, it is specifying the builtin root level, the parent of all first level libraries.

MSEX 1.0 and 1.1 suffer from a limitation imposed by using a uint8 to represent the LibraryCount and ElementCount values. MSEX 1.2 has removed this limitation by using a uint16 for these numbers, thus allowing library/element counts of up to the prescribed maximum of 256 to be reported.

Beginning with MSEX 1.2, element and library numbers are explicitly defined as being 0-based contiguous index values. E.g. if an element library is reported as containing 10 elements, those element numbers will be 0 thru 9. Prior to MSEX 1.2 the intention was the same, but the specification had been unclear: some implementations of MSEX 1.0 and 1.1 do not honor this pattern and allow for non-continuous library and element identifiers/numbers.

```
struct MSEXLibraryId
{
    uint8          Level;           // 0 - 3
    uint8          Level1;          // Sublevel 1 specifier, when Depth >= 1.
    uint8          Level2;          // Sublevel 2 specifier, when Depth >= 2.
    uint8          Level3;          // Sublevel 3 specifier, when Depth == 3.
};
```

Level1, Level2 and Level3 above are 0-based contiguous indexes for MSEX 1.2.

An attempt to visualize by example the most traditional structure, two levels:

```
/Root Folder (abstract) ID{0,0,0,0}
   /Images ID{1,0,0,0}
      /Primo.gif ID{2,0,0,0}
      /Secundo.gif ID{2,0,1,0}
      /Tertio.gif ID{2,0,2,0}
   /Movies ID{1,1,0,0}
```

```
      /One.mpg ID{2,1,0,0}
      /Two.mpg ID{2,1,1,0}
      /Three.avi ID{2,1,2,0}
   /Empty folder ID{1,2,0,0}
   /Empty folder ID{1,3,0,0}
   /More Movies ID{1,4,0,0}
      /Test.mpg ID{2,4,0,0}
      /Test2.avi ID{2,4,1,0}
```

There are currently eight recognized elements types (a library can only contain elements of one type) and when information about elements is requested, different kinds of Element Information messages (Media, Effect or Generic) are returned:

1. Media (images & video)
2. Effects
3. Cues
4. Crossfades
5. Masks
6. Blend presets
7. Effect presets
8. Image presets
9. 3D meshes

Change Detection

From MSEX 1.2, SerialNumber fields are included in all Element Library Information and Element Information messages. When a Media Server updates an item, that item's SerialNumber is incremented along with the SerialNumber of all parent nodes. E.g. in the above example, if Test2.avi is changed to some different media, the corresponding Media Element Information returned for the new item will have it's SerialNumber incremented, as will the SerialNumber for /More Movies. The Media Server should maintain SerialNumber values between sessions, so that previously connected clients can revalidate their cached information when they re-connect with the Media Server.

DMX Ranges

These value pairs identify the range of values that need to be sent over the corresponding DMX channel in order to select the relevant library or element. If a library contains the maximum 256 elements or sub-libraries, then each element will contain (0,0), (1,1), (2,2) etc. Some Media Servers may choose to distribute fewer elements over the available value range to make selection via an encoder wheel or fader easier. E.g. if a Media Server's media library contains only 10 subfolders, these might be assigned DMX ranges of (0,25), (26,50), (51,75) etc. which would evenly distribute the 10 folders across the full range.

10.4.1 MSEX / GELI - Get Element Library Information message

The GetElementLibraryInfo message is sent to a media server in order to request information about an element library, or all available element libraries.

```
struct CITP_MSEX_1.0_GELI
{
    CITP_MSEX_Header    CITPMSEXHeader;          // CITP MSEX header. MSEX ContentType
                                                 // is "GELI" and version is 1.0.
    uint8               LibraryType;             // Content type requested.
    uint8               LibraryCount;            // Number of libraries requested, set to
                                                 // 0 when requesting all available.
    uint8               LibraryNumbers[];        // Requested library numbers, none if
                                                 // LibraryCount is 0.
};

struct CITP_MSEX_1.1_GELI
{
    CITP_MSEX_Header    CITPMSEXHeader;          // CITP MSEX header. MSEX ContentType
                                                 // is "GELI" and version is 1.1.
    uint8               LibraryType;             // Content type requested.
    MSEXLibraryId       LibraryParentId;         // Parent library id.
    uint8               LibraryCount;            // Number of libraries requested, set to
                                                 // 0 when requesting all available.
    uint8               LibraryNumbers[];        // Requested library numbers, none if
                                                 // LibraryCount is 0.
};
```

The MSEX 1.2 version of this message uses a uint16 for LibraryCount to avoid the limitation described in "Message Definitions: Element libraries and element information":

```
struct CITP_MSEX_1.2_GELI
{
    CITP_MSEX_Header    CITPMSEXHeader;         // CITP MSEX header. MSEX ContentType
                                                //   is "GELI" and version is 1.2.
    uint8               LibraryType;            // Content type requested.
    MSEXLibraryId       LibraryParentId;        // Parent library id.
    uint16              LibraryCount;           // Number of libraries requested, set to
                                                //   0 when requesting all available.
    uint8               LibraryNumbers[];       // Requested library numbers, none if
                                                //   LibraryCount is 0.
};
```

Example 1: two DMX channel media selection media server. A GELI message with LibraryParentId set to {0, 0, 0, 0} is sent to retrieve all libraries on the folder selection channel. This generates a response with an ELIn message with at most 256 items with LibraryId values of {1, 0-255, 0, 0}.

Example 2: three DMX channel media selection media server. First the procedure in Example 1 is executed to collect all Level 1 libraries (none of these will contain any elements, but up to 256 sub libraries). For each N of these (up to 256) libraries, an additional GELI message is sent with the LibraryParentId set to {1, N, 0, 0}. This will trigger a response with an ELin message with at mosts 256 items with LibraryId values of {2, N, 0-255, 0}.

Note: Prior to MSEX 1.2 there is a limitation caused by the use of a uint8 to represent the library/element count, in which case the above examples can report at most 255 libraries and 255 elements within a library. See "Message definitions: Element libraries and element information", above

10.4.2 MSEX / ELIn - Element Library Information message

The ElementLibraryInfo message is sent in response to the GetElementLibraryInfo message. It should contain individual element library information for the *entire contents* of the requested element library.

```
struct CITP_MSEX_1.0_ELIn
{
    CITP_MSEX_Header    CITPMSEXHeader;         // CITP MSEX header. MSEX ContentType
                                                //   is "ELIn" and version is 1.0.
    uint8               LibraryType;            // Content type requested.
    uint8               LibraryCount;           // Number of following element library
                                                //   information blocks.

    struct ElementLibraryInformation
    {
        uint8           Number;                 // 0-based library number.
        uint8           DMXRangeMin;            // DMX range start value.
        uint8           DMXRangeMax;            // DMX range end value.
        ucs2            Name[];                 // Library name.

        uint8           ElementCount;           // Number of elements in the library.
    }[];
};

struct CITP_MSEX_1.1_ELIn
{
    CITP_MSEX_Header    CITPMSEXHeader;         // CITP MSEX header. MSEX ContentType
                                                //   is "ELIn" and version is 1.1.
    uint8               LibraryType;            // Content type requested.
    uint8               LibraryCount;           // Number of following element library
                                                //   information blocks.

    struct ElementLibraryInformation
    {
        MSEXLibraryId   Id;                     // Library id.
        uint8           DMXRangeMin;            // DMX range start value.
        uint8           DMXRangeMax;            // DMX range end value.
        ucs2            Name[];                 // Library name.

        uint8           LibraryCount;           // Number of sub libraries
                                                //   in the library.
        uint8           ElementCount;           // Number of elements in the library.
    }[];
};
```

The MSEX 1.2 version of this message uses a uint16 for LibraryCount & ElementCount to avoid the limitation described in "Message Definitions: Element libraries and element information":

```
struct CITP_MSEX_1.2_ELIn
{
    CITP_MSEX_Header    CITPMSEXHeader;         // CITP MSEX header. MSEX ContentType
                                                //   is "ELIn" and version is 1.2.
    uint8               LibraryType;            // Content type requested.
    uint16              LibraryCount;           // Number of following element library
                                                //   information blocks.
```

```
struct ElementLibraryInformation
{
    MSEXLibraryId   Id;                    // Library id.
    uint32          SerialNumber;          // See below
    uint8           DMXRangeMin;           // DMX range start value.
    uint8           DMXRangeMax;           // DMX range end value.
    ucs2            Name[];                // Library name.

    uint16          LibraryCount;          // Number of sub libraries in the library (0-256).
    uint16          ElementCount;          // Number of elements in the library (0-256).
}[];
};
```

SerialNumber: this field is used to detect changes to an element library. See Change Detection above.

10.4.3 MSEX / ELUp - Element Library Updated message

The ElementLibraryUpdated message is sent by a media server to notify a console or visualizer about updated media library contents.

```
struct CITP_MSEX_1.0_ELUp
{
    CITP_MSEX_Header   CITPMSEXHeader;     // CITP MSEX header. MSEX ContentType
                                           //    is "ELUp" and version is 1.0.
    uint8              LibraryType;        // Content type of updated library.
    uint8              LibraryNumber;      // Library that has been updated.
    uint8              UpdateFlags;        // Additional information flags.
                                           //     0x01   Existing elements have been
                                           //                updated
                                           //     0x02   Elements have been added or
                                           //                removed
                                           // 0x04 Sub libraries have been updated
                                           // 0x08 Sub libraries have been added or removed
};
```

```
struct CITP_MSEX_1.1_ELUp
{
    CITP_MSEX_Header   CITPMSEXHeader;     // CITP MSEX header. MSEX ContentType
                                           //    is "ELUp" and version is 1.1.
    uint8              LibraryType;        // Content type of updated library.
    MSEXLibraryId      LibraryId;          // Library that has been updated.
    uint8              UpdateFlags;        // Additional information flags.
                                           //     0x01   Existing elements have been
                                           //                updated
                                           //     0x02   Elements have been added or
                                           //                removed
                                           // 0x04 Sub libraries have been updated
                                           // 0x08 Sub libraries have been added or removed
};
```

```
struct CITP_MSEX_1.2_ELUp
{
    CITP_MSEX_Header   CITPMSEXHeader;     // CITP MSEX header. MSEX ContentType
                                           //    is "ELUp" and version is 1.2.
    uint8              LibraryType;        // Content type of updated library.
    MSEXLibraryId      LibraryId;          // Library that has been updated.
    uint8              UpdateFlags;        // Additional information flags.
                                           // 0x01 Existing elements have been updated
                                           // 0x02 Elements have been added or removed
                                           // 0x04 Sub libraries have been updated
                                           // 0x08 Sub libraries have been added or removed
                                           // 0x10 All elements have been affected
                                           //         (ignore AffectedElements)
                                           // 0x20 All sub libraries have been affected
                                           //         (ignore AffectedLibraries)
    AffectedItems      AffectedElements;   // Which elements have been affected
    AffectedItems      AffectedLibraries;  // Which sub-libraries have been affected
};
```

The MSEX 1.2 (and later) version of ELUp contains extra detail to identify which elements and/or sublibraries have been changed.

```
struct AffectedItems
{
    uint8              ItemSet[32];        // A set of 256 bits used to indicate which item
                                           //    numbers have been changed
};
```

E.g. the following test will be true if the element or library indexed by ItemIndex has changed:

```
ItemSet[ItemIndex / 8] & (1 << (ItemIndex % 8))
```

10.4.4 MSEX / GEIn - Get Element Information message

The GetElementInformation message is sent by a console or visualizer to a media server in order to request information about individual elements.

```
struct CITP_MSEX_1.0_GEIn
{
    CITP_MSEX_Header    CITPMSEXHeader;         // CITP MSEX header. MSEX ContentType
                                                //   is "GEIn" and version is 1.0.
    uint8               LibraryType;            // Content type requested.
    uint8               LibraryNumber;          // Library for which to retrieve element info.
    uint8               ElementCount;           // Number of elements for which information
                                                //   is requested, set to 0 when requesting
                                                //   all available.
    uint8               ElementNumbers[];       // Numbers of the elements for which
                                                //   information is requested.
};

struct CITP_MSEX_1.1_GEIn
{
    CITP_MSEX_Header    CITPMSEXHeader;         // CITP MSEX header. MSEX ContentType
                                                //   is "GEIn" and version is 1.1.
    uint8               LibraryType;            // Content type requested.
    MSEXLibraryId       LibraryId;              // Library for which to retrieve elements
    uint8               ElementCount;           // Number of elements for which information
                                                //   is requested, set to 0 when requesting
                                                //   all available.
    uint8               ElementNumbers[];       // Numbers of the elements for which
                                                //   information is requested.
};
```

The MSEX 1.2 version of this message uses a uint16 for ElementCount to avoid the limitation described in "Message Definitions: Element libraries and element information":

```
struct CITP_MSEX_1.2_GEIn
{
    CITP_MSEX_Header    CITPMSEXHeader;         // CITP MSEX header. MSEX ContentType
                                                //   is "GEIn" and version is 1.1.
    uint8               LibraryType;            // Content type requested.
    MSEXLibraryId       LibraryId;              // Library for which to retrieve elements
    uint16              ElementCount;           // Number of elements for which information
                                                //   is requested, set to 0 when requesting
                                                //   all available.
    uint8               ElementNumbers[];       // Numbers of the elements for which
                                                //   information is requested.
};
```

10.4.5 MSEX / MEIn - Media Element Information message

The MediaElementInformation message is sent in response to the GetElementInformation message for element type 1. It should contain individual media element information for *all* elements requested.

```
struct CITP_MSEX_1.0_MEIn
{
    CITP_MSEX_Header    CITPMSEXHeader;         // CITP MSEX header. MSEX ContentType
                                                //   is "MEIn" and version is 1.0.
    uint8               LibraryNumber;          // Library containing the media elements.
    uint8               ElementCount;           // Number of following (media) information
                                                //   blocks.
    struct MediaInformation
    {
        uint8           Number;                 // 0-based number of the media.
        uint8           DMXRangeMin;            // DMX range start value.
        uint8           DMXRangeMax;            // DMX range end value.
        ucs2            MediaName[];            // Media name.
        uint64          MediaVersionTimestamp;  // Media version in seconds since
                                                //   1st January 1970.
        uint16          MediaWidth;             // Media width.
        uint16          MediaHeight;            // Media height.
        uint32          MediaLength;            // Media length (in frames).
        uint8           MediaFPS;               // Media resolution (in frames per second).
    }[];
};

struct CITP_MSEX_1.1_MEIn
{
    CITP_MSEX_Header    CITPMSEXHeader;         // CITP MSEX header. MSEX ContentType
```

```
                                                   is "MEIn" and version is 1.1.
    MSEXLibraryId        LibraryId;            // Library containing the media elements.
    uint8                ElementCount;         // Number of following (media) information
                                               //    blocks.
    struct MediaInformation
    {
        uint8            Number;               // 0-based number of the media.
        uint8            DMXRangeMin;          // DMX range start value.
        uint8            DMXRangeMax;          // DMX range end value.
        ucs2             MediaName[];          // Media name.
        uint64           MediaVersionTimestamp; // Media version in seconds since
                                               //    1st January 1970.
        uint16           MediaWidth;           // Media width.
        uint16           MediaHeight;          // Media height.
        uint32           MediaLength;          // Media length (in frames).
        uint8            MediaFPS;             // Media resolution (in frames per second).
    }[];
};
```

The MSEX 1.2 version of this message uses a uint16 for ElementCount to avoid the limitation described in "Message Definitions: Element libraries and element information":

```
struct CITP_MSEX_1.2_MEIn
{
    CITP_MSEX_Header     CITPMSEXHeader;       // CITP MSEX header. MSEX ContentType
                                               //    is "MEIn" and version is 1.1.
    MSEXLibraryId        LibraryId;            // Library containing the media elements.
    uint16               ElementCount;         // Number of following (media) information
                                               //    blocks.
    struct MediaInformation
    {
        uint8            Number;               // 0-based contiguous index of the media.
        uint32           SerialNumber;         // See below
        uint8            DMXRangeMin;          // DMX range start value.
        uint8            DMXRangeMax;          // DMX range end value.
        ucs2             MediaName[];          // Media name.
        uint64           MediaVersionTimestamp; // Media version in seconds since
                                               //    1st January 1970.
        uint16           MediaWidth;           // Media width.
        uint16           MediaHeight;          // Media height.
        uint32           MediaLength;          // Media length (in frames).
        uint8            MediaFPS;             // Media resolution (in frames per second).
    }[];
};
```

SerialNumber: this field is used to detect changes to an element within a library. See Change Detection above.

10.4.6 MSEX / EEIn - Effect Element Information message

The EffectElementInformation message is sent in response to the GetElementInformation message for element type 2. It contains individual effect element information for *all* elements requested.

```
struct CITP_MSEX_1.0_EEIn
{
    CITP_MSEX_Header     CITPMSEXHeader;       // CITP MSEX header. MSEX ContentType
                                               //    is "EEIn" and version is 1.0.
    uint8                LibraryNumber;        // Library containing the effect elements.
    uint8                ElementCount;         // Number of following (effect) information
                                               //    blocks.
    struct EffectInformation
    {
        uint8            ElementNumber;        // 0-based number of the effect.
        uint8            DMXRangeMin;          // DMX range start value.
        uint8            DMXRangeMax;          // DMX range end value.
        ucs2             EffectName[];         // Effect name.
        uint8            EffectParameterCount; // Number of following effect
                                               // parameter names.
        ucs2             EffectParameterNames[][]; // List of effect parameter names.
    }[];
};

struct CITP_MSEX_1.1_EEIn
{
    CITP_MSEX_Header     CITPMSEXHeader;       // CITP MSEX header. MSEX ContentType
                                               //    is "EEIn" and version is 1.1.
    MSEXLibraryId        LibraryId;            // Library containing the effect elements.
    uint8                ElementCount;         // Number of following (effect) information
                                               //    blocks.
    struct EffectInformation
    {
```

```
        uint8           ElementNumber;          // 0-based number of the effect.
        uint8           DMXRangeMin;            // DMX range start value.
        uint8           DMXRangeMax;            // DMX range end value.
        ucs2            EffectName[];           // Effect name.
        uint8           EffectParameterCount;   // Number of following effect
                                                   parameter names.
        ucs2            EffectParameterNames[][]; // List of effect parameter names.
    }[];
};
```

The MSEX 1.2 version of this message uses a uint16 for ElementCount to avoid the limitation described in "Message Definitions: Element libraries and element information":

```
struct CITP_MSEX_1.2_EEIn
{
    CITP_MSEX_Header    CITPMSEXHeader;         // CITP MSEX header. MSEX ContentType
                                                   is "EEIn" and version is 1.1.
    MSEXLibraryId       LibraryId;              // Library containing the effect elements.
    uint16              ElementCount;           // Number of following (effect) information
                                                   blocks.

    struct EffectInformation
    {
        uint8           ElementNumber;          // 0-based contiguous index of the effect.
        uint32          SerialNumber;           // See below
        uint8           DMXRangeMin;            // DMX range start value.
        uint8           DMXRangeMax;            // DMX range end value.
        ucs2            EffectName[];           // Effect name.
        uint8           EffectParameterCount;   // Number of following effect
                                                   parameter names.
        ucs2            EffectParameterNames[][]; // List of effect parameter names.
    }[];
};
```

SerialNumber: this field is used to detect changes to an element within a library. See Change Detection above.

10.4.7 MSEX / GLEI - Generic Element Information message

The GenericElementInformation message is sent in response to the GetElementInformation message for element types 3 through 8. It contains individual element information for *all* elements requested.

```
struct CITP_MSEX_1.1_GLEI
{
    CITP_MSEX_Header    CITPMSEXHeader;         // CITP MSEX header. MSEX ContentType
                                                   is "GLEI" and version is 1.1.
    MSEXLibraryId       LibraryId;              // Library containing the elements.
    uint8               ElementCount;           // Number of following information
                                                   blocks.

    struct GenericInformation
    {
        uint8           ElementNumber;          // 0-based number of the element.
        uint8           DMXRangeMin;            // DMX range start value.
        uint8           DMXRangeMax;            // DMX range end value.
        ucs2            Name[];                 // Element name.
        uint64          VersionTimestamp;       // Element version in
                                                   seconds since 1st January 1970.
    }[];
};
```

Note: The MSEX 1.1 version of this message lacks a field indicating which library type the contained information belongs to (which is not necessary with the MEIn and EEIn messages since each is for a particular library type). The MSEX 1.2 version of this message defined below corrects this problem, as well as ElementCount limitation described in "Message Definitions: Element libraries and element information":

```
struct CITP_MSEX_1.2_GLEI
{
    CITP_MSEX_Header    CITPMSEXHeader;         // CITP MSEX header. MSEX ContentType
                                                   is "GLEI" and version is 1.1.
    uint8               LibraryType;            // Library content type.
    MSEXLibraryId       LibraryId;              // Library containing the elements.
    uint16              ElementCount;           // Number of following information
                                                   blocks.

    struct GenericInformation
    {
        uint8           ElementNumber;          // 0-based contiguous index of the element.
        uint32          SerialNumber;           // See below
        uint8           DMXRangeMin;            // DMX range start value.
        uint8           DMXRangeMax;            // DMX range end value.
        ucs2            Name[];                 // Element name.
        uint64          VersionTimestamp;       // Element version in
```

```
    }[];
};
```

SerialNumber: this field is used to detect changes to an element within a library. See Change Detection above.

10.5 Message definitions: Thumbnail information

10.5.1 MSEX / GELT - Get Element Library Thumbnail message

The GetElementLibraryThumbnail message is sent to a media server in order to retrieve a thumbnail of an element library, or of all available element libraries.

```
struct CITP_MSEX_1.0_GELT
{
    CITP_MSEX_Header    CITPMSEXHeader;        // CITP MSEX header. MSEX ContentType
                                               //   is "GELT" and version is 1.0.
    uint32              ThumbnailFormat;       // Format of the thumbnail.
                                               //   Can be "RGB8" or "JPEG" (or "PNG " for MSEX 1.2 and up).
    uint16              ThumbnailWidth;        // Preferred thumbnail image width.
    uint16              ThumbnailHeight;       // Preferred thumbnail image height.
    uint8               ThumbnailFlags         // Additional information flags.
                                               //     0x01   Preserve aspect ratio
                                               //   of image (use width and height as maximum)
    uint8               LibraryType;           // 1 for Media, 2 for Effects.
    uint8               LibraryCount;          // Number of libraries requested, set to 0
                                               //   when requesting all available.
    uint8               LibraryNumbers[];      // Numbers of the libraries requested, not present
                                               //   if LibraryCount is 0.
};

struct CITP_MSEX_1.1_GELT
{
    CITP_MSEX_Header    CITPMSEXHeader;        // CITP MSEX header. MSEX ContentType
                                               //   is "GELT" and version is 1.1.
    uint32              ThumbnailFormat;       // Format of the thumbnail.
                                               //   Can be "RGB8" or "JPEG" (or "PNG " for MSEX 1.2 and up).
    uint16              ThumbnailWidth;        // Preferred thumbnail image width.
    uint16              ThumbnailHeight;       // Preferred thumbnail image height.
    uint8               ThumbnailFlags         // Additional information flags.
                                               //     0x01   Preserve aspect ratio
                                               //   of image (use width and height as maximum)
    uint8               LibraryType;           // 1 for Media, 2 for Effects.
    uint8               LibraryCount;          // Number of libraries requested, set to 0
                                               //   when requesting all available.
    MSEXLibraryId       LibraryIds[];          // Ids of the libraries requested, not present if
                                               //   LibraryCount is 0.
};
```

The MSEX 1.2 version of this message uses a uint16 for LibraryCount to avoid the limitation described in "Message Definitions: Element libraries and element information":

```
struct CITP_MSEX_1.2_GELT
{
    CITP_MSEX_Header    CITPMSEXHeader;        // CITP MSEX header. MSEX ContentType
                                               //   is "GELT" and version is 1.2.
    uint32              ThumbnailFormat;       // Format of the thumbnail.
                                               //   Can be "RGB8" or "JPEG" (or "PNG " for MSEX 1.2 and up).
    uint16              ThumbnailWidth;        // Preferred thumbnail image width.
    uint16              ThumbnailHeight;       // Preferred thumbnail image height.
    uint8               ThumbnailFlags         // Additional information flags.
                                               //     0x01   Preserve aspect ratio
                                               //   of image (use width and height as maximum)
    uint8               LibraryType;           // 1 for Media, 2 for Effects.
    uint16              LibraryCount;          // Number of libraries requested, set to 0
                                               //   when requesting all available.
    MSEXLibraryId       LibraryIds[];          // Ids of the libraries requested, not present if
                                               //   LibraryCount is 0.
};
```

10.5.2 MSEX / ELTh - Element Library Thumbnail message

The ElementLibraryThumbnail message is sent in response to the GetElementLibraryThumbnail message.

```
struct CITP_MSEX_1.0_ELTh
{
    CITP_MSEX_Header    CITPMSEXHeader;        // CITP MSEX header. MSEX ContentType
                                               //   is "ELTh" and version is 1.0.
```

```
    uint8               LibraryType;            // 1 for Media, 2 for Effects.
    uint8               LibraryNumber;          // Number of the library that
                                                        the thumbnail belongs to.
    uint32              ThumbnailFormat;        // Format of the thumbnail.
                                                   Can be "RGB8" or "JPEG" (or "PNG " for MSEX 1.2 and up).
    uint16              ThumbnailWidth;         // Thumbnail width.
    uint16              ThumbnailHeight;        // Thumbnail height.
    uint16              ThumbnailBufferSize;    // Size of the thumbnail buffer.
    uint8               ThumbnailBuffer[];      // Thumbnail image buffer.
};

struct CITP_MSEX_1.1_ELTh
{
    CITP_MSEX_Header    CITPMSEXHeader;         // CITP MSEX header. MSEX ContentType
                                                   is "ELTh" and version is 1.1.
    uint8               LibraryType;            // 1 for Media, 2 for Effects.
    MSEXLibraryId       LibraryId;              // Id of the library that the thumbnail
                                                   belongs to.
    uint32              ThumbnailFormat;        // Format of the thumbnail.
                                                   Can be "RGB8" or "JPEG" (or "PNG " for MSEX 1.2 and up).
    uint16              ThumbnailWidth;         // Thumbnail width.
    uint16              ThumbnailHeight;        // Thumbnail height.
    uint16              ThumbnailBufferSize;    // Size of the thumbnail buffer.
    uint8               ThumbnailBuffer[];      // Thumbnail image buffer.
};
```

10.5.3 MSEX / GETh - Get Element Thumbnail message

The GetElementThumbnail message is sent to a media server in order to retrieve a thumbnail of one or many library elements..

```
struct CITP_MSEX_1.0_GETh
{
    CITP_MSEX_Header    CITPMSEXHeader;         // CITP MSEX header. MSEX ContentType
                                                   is "GETh" and version is 1.0.
    uint32              ThumbnailFormat;        // Format of the thumbnail.
                                                   Can be "RGB8" or "JPEG" (or "PNG " for MSEX 1.2 and up).
    uint16              ThumbnailWidth;         // Preferred thumbnail image width.
    uint16              ThumbnailHeight;        // Preferred thumbnail image height.
    uint8               ThumbnailFlags          // Additional information flags.
                                                // 0x01   Preserve aspect ratio of image
                                                            (use width and height as maximum)
    uint8               LibraryType;            // 1 for Media, 2 for Effects.
    uint8               LibraryNumber;          // Number of the media's library.
    uint8               ElementCount;           // Number of medias for which information
                                                   is requested, set to 0 when requesting
                                                   all available.
    uint8               ElementNumbers[];       // The numbers of the requested elements.
                                                   Not present if ElementCount is 0.
};

struct CITP_MSEX_1.1_GETh
{
    CITP_MSEX_Header    CITPMSEXHeader;         // CITP MSEX header. MSEX ContentType
                                                   is "GETh" and version is 1.1.
    uint32              ThumbnailFormat;        // Format of the thumbnail.
                                                   Can be "RGB8" or "JPEG" (or "PNG " for MSEX 1.2 and up).
    uint16              ThumbnailWidth;         // Preferred thumbnail image width.
    uint16              ThumbnailHeight;        // Preferred thumbnail image height.
    uint8               ThumbnailFlags          // Additional information flags.
                                                // 0x01   Preserve aspect ratio of image
                                                            (use width and height as maximum)
    uint8               LibraryType;            // 1 for Media, 2 for Effects.
    MSEXLibraryId       LibraryId;              // Id of the media's library.
    uint8               ElementCount;           // Number of medias for which information
                                                   is requested, set to 0 when requesting
                                                   all available.
    uint8               ElementNumbers[];       // The numbers of the requested elements.
                                                   Not present if ElementCount = 0. For MSEX 1.2 these are
                                                   0-based contiguous index values.
};
```

The MSEX 1.2 version of this message uses a uint16 for ElementCount to avoid the limitation described in "Message Definitions: Element libraries and element information":

```
struct CITP_MSEX_1.2_GETh
{
    CITP_MSEX_Header    CITPMSEXHeader;         // CITP MSEX header. MSEX ContentType
                                                   is "GETh" and version is 1.2.
    uint32              ThumbnailFormat;        // Format of the thumbnail.
                                                   Can be "RGB8" or "JPEG" (or "PNG " for MSEX 1.2 and up).
```

```
    uint16           ThumbnailWidth;      // Preferred thumbnail image width.
    uint16           ThumbnailHeight;     // Preferred thumbnail image height.
    uint8            ThumbnailFlags       // Additional information flags.
                                          // 0x01   Preserve aspect ratio of image
                                          //        (use width and height as maximum)
    uint8            LibraryType;         // 1 for Media, 2 for Effects.
    MSEXLibraryId    LibraryId;           // Id of the media's library.
    uint16           ElementCount;        // Number of medias for which information
                                          // is requested, set to 0 when requesting
                                          // all available.
    uint8            ElementNumbers[];    // The numbers of the requested elements.
                                          // Not present if ElementCount = 0. For MSEX 1.2 these are
                                          // 0-based contiguous index values.
};
```

10.5.4 MSEX / EThn - Element Thumbnail message

The ElementLibraryThumbnail message is sent in response to the GetElementLibraryThumbnail message.

```
struct CITP_MSEX_1.0_EThn
{
    CITP_MSEX_Header  CITPMSEXHeader;     // CITP MSEX header. MSEX ContentType
                                          // is "EThn" and version is 1.0.
    uint8             LibraryType;         // 1 for Media, 2 for Effects.
    uint8             LibraryNumber;       // Number of the element's library.
    uint8             ElementNumber;       // Number of the element.
    uint32            ThumbnailFormat;     // Format of the thumbnail.
                                           // Can be "RGB8" or "JPEG" (or "PNG " for MSEX 1.2 and up).
    uint16            ThumbnailWidth;      // Thumbnail width.
    uint16            ThumbnailHeight;     // Thumbnail height.
    uint16            ThumbnailBufferSize; // Size of the thumbnail buffer.
    uint8             ThumbnailBuffer[];   // Thumbnail image buffer.
};

struct CITP_MSEX_1.1_EThn
{
    CITP_MSEX_Header  CITPMSEXHeader;     // CITP MSEX header. MSEX ContentType
                                          // is "EThn" and version is 1.1.
    uint8             LibraryType;         // 1 for Media, 2 for Effects.
    MSEXLibraryId     LibraryId;           // Id of the element's library.
    uint8             ElementNumber;       // Number of the element (For MSEX 1.2 this
                                           // is a 0-based contiguous index value).
    uint32            ThumbnailFormat;     // Format of the thumbnail.
                                           // Can be "RGB8" or "JPEG" (or "PNG " for MSEX 1.2 and up).
    uint16            ThumbnailWidth;      // Thumbnail width.
    uint16            ThumbnailHeight;     // Thumbnail height.
    uint16            ThumbnailBufferSize; // Size of the thumbnail buffer.
    uint8             ThumbnailBuffer[];   // Thumbnail image buffer.
};
```

10.6 Message definitions: Streams

10.6.1 MSEX / GVSr - GetVideoSources

The GetVideoSources message is sent to a media server in order to receive all available video source feeds.

```
struct CITP_MSEX_GVSr
{
    CITP_MSEX_Header  CITPMSEXHeader;     // CITP MSEX header. MSEX ContentType
                                          // is "GVSr".
};
```

10.6.2 MSEX / VSrc - Video Sources

The VideoSources message is sent in response to a GetVideoSources message. The PhysicalOutput and LayerNumber fields can be used for automatic connection to outputs and individual layers (for instance the video of output 1 would have PhysicalOutput = 0 and LayerNumber = 0xFF).

```
struct CITP_MSEX_VSrc
{
    CITP_MSEX_Header  CITPMSEXHeader;     // CITP MSEX header. MSEX ContentType
                                          // is "VSrc".
    uint16            SourceCount;         // Number of following source information
                                           // blocks.
    struct SourceInformation
    {
        uint16        SourceIdentifier;    // Source identifier.
```

```
        ucs2            SourceName[];           // Display name of the source (ie "Output 1",
                                                   "Layer 2", "Camera 1" etc).
        uint8           PhysicalOutput;         // If applicable, 0-based index designating
                                                   the physical video output index.
                                                   Otherwise 0xFF.
        uint8           LayerNumber;            // If applicable, 0-based layer number,
                                                   corresponding to the layers reported in
                                                   the SInf message. Otherwise 0xFF.
        uint16          Flags;                  // Information flags.
                                                //    0x0001 Without effects
        uint16          Width;                  // Full width.
        uint16          Height;                 // Full height.
    };
};
```

10.6.3 MSEX / RqSt - Request Stream message

The RequestStream message is sent by a console or visualizer to a media server in order to create a time limited subscription of a video source. The media server will not provide multiple resolutions and frame rates of a single source, but it may provide a feed for each requested format. If different resolutions are requested by multiple peers, the Media Server should only supply the higher resolution to all peers (any peer should be prepared to downscale). It is up to the peer to regularly request a stream, based on its timeout parameter, if it wishes receive a continuous feed. High values of the timeout field is of course discouraged.

```
struct CITP_MSEX_RqSt
{
    CITP_MSEX_Header    CITPMSEXHeader;         // CITP MSEX header. MSEX ContentType
                                                   is "RqSt".
    uint16              SourceIdentifier;       // Identifier of the source requested.
    uint32              FrameFormat;            // Requested frame format.
                                                   Can be "RGB8" or "JPEG" (or "PNG " for MSEX 1.2 and up).
    uint16              FrameWidth;             // Preferred minimum frame width.
    uint16              FrameHeight;            // Preferred minimum frame height.
    uint8               FPS;                    // Preferred minimum frames per second.
    uint8               Timeout;                // Timeout in seconds (for instance 5
                                                   seconds, 0 to ask for only one frame).
};
```

10.6.4 MSEX / StFr - Stream Frame message

The StreamFrame message is multicasted regularly from a media server. The resolutions, formats and FPS are determine by the current set of subscribing peers.

```
struct CITP_MSEX_1.0_StFr
{
    CITP_MSEX_Header    CITPMSEXHeader;         // The CITP MSEX header. MSEX ContentType
                                                   is "StFr".
    uint16              SourceIdentifier;       // Identifier of the frame's source.
    uint32              FrameFormat;            // Requested frame format.
                                                   Can be "RGB8" or "JPEG" (or "PNG " for MSEX 1.2 and up).
    uint16              FrameWidth;             // Preferred minimum frame width.
    uint16              FrameHeight;            // Preferred minimum frame height.
    uint16              FrameBufferSize;        // Size of the frame image buffer.
    uint8               FrameBuffer[];          // Frame image buffer.
};
```

Prior to version 1.1 of MSEX, RGB8 data was transmitted as BGR rather then RGB. As of version 1.1, stream frames are to be transmitted over the multicast channel only (sames as used by PINF) and never over the TCP connection.

```
struct CITP_MSEX_1.2_StFr
{
    CITP_MSEX_Header    CITPMSEXHeader;         // The CITP MSEX header. MSEX ContentType
                                                   is "StFr".
    ucs1                MediaServerUUID[36];    // Source media server UUID, see below.
    uint16              SourceIdentifier;       // Identifier of the frame's source.
    uint32              FrameFormat;            // Requested frame format.
                                                   Can be "RGB8" or "JPEG" (or "PNG " for MSEX 1.2 and up).
    uint16              FrameWidth;             // Preferred minimum frame width.
    uint16              FrameHeight;            // Preferred minimum frame height.
    uint16              FrameBufferSize;        // Size of the frame image buffer.
    uint8               FrameBuffer[];          // Frame image buffer.
};
```

As of version 1.2, the source media server UUID was added as a means of distinguishing incoming stream frames from different media servers on the same IP address.