1. CITP CAEX Layer Specification

1.1 History

Date	*Version*	*Brief info*
2013-06-15	PA1	First prerelease of A draft 1, featuring Live View and Cue Recording messages.
2013-07-13	PA2	Added reply message clarification to the GetLiveViewImage message.
2013-08-03	PA3	Removed trailing copy & paste remark for the RecordCue message.
2013-10-15	PA4	Added new Cue Recording messages for clearing.
2013-11-09	A	Accepted as version A with the release of Capture Polar 2.11.
2014-10-23	PB1	Added new Show Synchronization messages.
2015-01-21	PB2	Polished Show Synchronization messages.
2015-05-07	B	Accepted as version B with the release of Capture Argo 21.0.
2015-10-25	PC1	First prerelease of C draft 1, featuring laser feeds.
2015-11-02	PC2	Changed laser feed frames to be sent over UDP and added data packing.
2015-11-06	C	Accepted as version C with the release of Capture Argo 21.1.20.
2017-05-15	PD1	Added the FixtureIdentify message.
2017-06-05	D	Accepted as version D with the release of Capture Nexum 23.1.
2018-07-27	E	Clarifications with regard to the guid type and format of anglular information. Types of RDM identification clarified and extended.

1.2 Introduction

The CITP/CAEX "Capture Extensions" layer is a set of networking messages implemented as a private CITP (http://www.citp-protocol.org) layer. This document could be interpreted as an addendum to the CITP protocol specification.

1.3 Data types

The following data types are used in CITP/CAEX, in addition to those defined in CITP.

```
guid      // 128-bit UUID encoded using Microsoft COM/OLE mixed-endian format.
          // "00112233-4455-6677-8899-aabbccddeeff" is encoded as
          // the bytes "33 22 11 00 55 44 77 66 88 99 aa bb cc dd ee ff".
```

2. Header and NACK message

Most messages in CAEX are of request/reply nature, although some messages may be sent unsoliticed. Capture will honor the CITP header RequestIndex and InResponseTo fields if they are set, allowing the peer to correlate replies to responses.

2.1 The CAEX header

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

```
CITP_CAEX_Header
{
    CITP_Header         CITPHeader           // CITP header. CITP ContentType is "CAEX".
    uint32              ContentCode          // Number identifying which CAEX message it is.
}
```

Unlike other CITP messages, CAEX doesn't use ASCII-form 32-bit integers to identify messages, but rather standard numeric 32-bit integers.

2.2 CAEX / NACK message

This message must be sent by Capture or a peer in response to any unknown message or any request that prompts a reply which cannot be served. The CITP header RequestIndex and InResponseTo fields must be honored when sending this message.

```
CITP_CAEX_NACK
{
    CITP_CAEX_Header    CITPCAEXHeader       // CITP CAEX header, ContentCode = 0xFFFFFFFF.
    uint8               Reason               // 0x00 = Unknown request.
                                             // 0x01 = Incorrect or malformed request.
                                             // 0x02 = Internal error.
                                             // 0x03 = Request refused.
}
```

3. Live View messages

Capture allows a peer to interact with one of its views in live mode. If no view is in live mode, these features will not be available. If multiple views are in live mode, the first view in live mode will be the one with which the peer can interact.

3.1 CAEX / GetLiveViewStatus message

This message is sent to Capture as a request for a LiveViewStatus reply.

```
CITP_CAEX_GetLiveViewStatus
{
    CITP_CAEX_Header   CITPCAEXHeader       // CITP CAEX header, ContentCode = 0x00000100.
}
```

3.2 CAEX / LiveViewStatus message

This message is sent by Capture in reply to a GetLiveViewStatus message.

```
CITP_CAEX_LiveViewStatus
{
    CITP_CAEX_Header   CITPCAEXHeader       // CITP CAEX header, ContentCode = 0x00000101.
    uint8              Availability         // 0x00 = Not available.
                                            // 0x01 = Alpha view available.
                                            // 0x02 = Beta view available.
                                            // 0x03 = Gamma view available.
    uint16[2]          Size                 // The width and height of the view.
    float[3]           CameraPosition       // The XYZ position of the camera.
    float[3]           CameraFocus          // The XYZ focus of the camera.
}
```

Note: The purpose of sending the camera position and focus is so that the peer can use this as default when requesting live images.

3.3 CAEX / GetLiveViewImage message

This message is sent to Capture as a request for a LiveViewImage reply.

```
CITP_CAEX_GetLiveViewImage
{
    CITP_CAEX_Header   CITPCAEXHeader       // CITP CAEX header, ContentCode = 0x00000200.
    uint8              Format               // 0x01 = JPEG
    uint16[2]          Resolution           // The width and height requested.
    float[3]           CameraPosition       // The position of the camera, as XYZ.
    float[3]           CameraFocus          // The focus of the camera, as XYZ.
}
```

Note: If the peer requests a resolution higher than the current size of the view, Capture will return an image with the current size of the view. If no live view is available, Capture will reply with a 'Refused' NACK.

3.4 CAEX / LiveViewImage message

This message is sent by Capture in reply to a GetLiveViewImage message.

```
CITP_CAEX_LiveViewImage
{
    CITP_CAEX_Header   CITPCAEXHeader       // CITP CAEX header, ContentCode = 0x00000201.
    uint8              Format               // 0x01 = JPEG
    uint32             DataSize             // The number of bytes in the following field.
    uint8[DataSize]    Data                 // Image data.
}
```

4. Cue Recording messages

Capture allows a peer with cue recording capabilities to expose some amount of remote control of its 'recording unit' / 'programmer'.

4.1 CAEX / SetCueRecordingCapabilities message

This message is sent to Capture in order to enable or disable remote cue recording.

```
CITP_CAEX_SetCueRecordingCapabilities
{
   CITP_CAEX_Header   CITPCAEXHeader       // CITP CAEX header, ContentCode = 0x00010100.
   uint8              Availability         // 0x00 = Not available
                                           // 0x01 = Available
   uint8              OptionCount          // The number of options that follow.
   {
      ucs2               Name              // The name of the option.
      ucs2               Choices           // Tab ('\t') separated list of choices.
      ucs2               Help              // An optional explanation/description.
   }[OptionCount]
}
```

Note: The abscence of choices for an option denotes a free text field.

4.2 CAEX / RecordCue message

This message is sent unsolicited by Capture when the user records a cue.

```
CITP_CAEX_RecordCue
{
   CITP_CAEX_Header   CITPCAEXHeader       // CITP CAEX header, ContentCode = 0x00010200.
   uint8              OptionCount          // The number of options that follow.
   {
      ucs2               Name              // The name of the option.
      ucs2               Value             // Value of the option.
   }[OptionCount]
}
```

4.3 CAEX / SetRecorderClearingCapabilities message

This message is sent to Capture in order to enable or disable remote clearing of the recorder.

```
CITP_CAEX_SetRecorderClearingCapabilities
{
   CITP_CAEX_Header   CITPCAEXHeader       // CITP CAEX header, ContentCode = 0x00010300.
   uint8              Availability         // 0x00 = Unsupported (default state)
                                           // 0x01 = Currently unavailable
                                           // 0x02 = Available
}
```

4.4 CAEX / ClearRecorder message

This message is sent unsolicited by Capture when the user wishes to clear the state of the recorder.

```
CITP_CAEX_ClearRecorder
{
   CITP_CAEX_Header   CITPCAEXHeader       // CITP CAEX header, ContentCode = 0x00010400.
}
```

5. Show Synchronization messages

The Show Synchronization messages allow a peer to exchange patch, selection and fixture status information with Capture.

In order for the user experience to be smooth and seamless, it is necessary to communicate "show state" information with Capture. The following are the rules of interaction:

- Capture will send EnterShow and LeaveShow messages as projects are opened and closed, given that the user has enabled the "console link" with the peer. If "console link" is disabled and then reenabled, Capture will act as if the project was closed and opened again. Always keep track of whether Capture is currently in a show or not.
- When opening or creating a new show: send an EnterShow message to Capture.
- When opening or creating a new show and Capture is currently in a show: send a patch information request to Capture.
- When closing a show: send a LeaveShow message to Capture.
- When in a show and Capture enters a show: send a patch information request to Capture.
- If the user chooses to disable synchronization: act as if the user had closed the show.
- If the user chooses to reenable synchronization: act as if the user had just opened the current show.

It is important that the peer, upon receving complete patch information when both the peer and Capture have entered a show, provides the user with the means to determine whether the patch is in sync and/or requires modification, as well as the option to disable the synchronization.

5.1 CAEX / EnterShow message

This message is sent unsolicited by both Capture and the peer when a show/project is opened and/or the user wishes to enable show synchronization.

```
CITP_CAEX_EnterShow
{
    CITP_CAEX_Header   CITPCAEXHeader       // CITP CAEX header, ContentCode = 0x00020100.
    ucs2               Name                 // The name of the show.
}
```

5.2 CAEX / LeaveShow message

This message is sent unsolicited by both Capture and the peer when a show/project is closed or when the user wishes to disable show synchronization.

```
CITP_CAEX_LeaveShow
{
    CITP_CAEX_Header   CITPCAEXHeader       // CITP CAEX header, ContentCode = 0x00020101.
}
```

5.3 CAEX / FixtureListRequest message

This message can be sent unsolicited by Capture or a peer in order to acquire the full patch list from the other side. The expected response is a FixtureList message with Type = 0x00.

```
CITP_CAEX_FixtureListRequest
{
    CITP_CAEX_Header   CITPCAEXHeader       // CITP CAEX header, ContentCode = 0x00020200.
}
```

5.4 CAEX / FixtureList message

This message is sent in response to a FixtureListRequest message (with Type = 0x00) as well as unsolicited by both Capture and the peer (with Type = 0x01 or Type = 0x02). An existing patch fixture list (Type = 0x00) must contain all known fixtures while a new fixture (Type = 0x01) or exchanged fixture (Type = 0x02) message contains only the fixture(s) that were recently added or exchanged for other fixtures.

```
CITP_CAEX_FixtureList
{
    CITP_CAEX_Header   CITPCAEXHeader             // CITP CAEX header, ContentCode = 0x00020201.
    uint8              Type                       // 0x00 = Existing patch list
                                                  // 0x01 = New fixture(s)
                                                  // 0x02 = Exchanged fixture(s)
    uint16             FixtureCount               // Number of fixtures following.
    {
       uint32             FixtureIdentifer        // Console's fixture identifier.
                                                  // Set to 0xffffffff if unknown by Capture.
       ucs2               ManufacturerName        // The name of the fixture's manufacturer.
       ucs2               FixtureName             // The model name of the fixture.
       ucs2               ModeName                // The name of DMX mode.
       uint16             ChannelCount            // The number of channels of the DMX mode.
       uint8              IsDimmer                // A boolean 0x00 or 0x01 indicating whether it's.
```

```
                                                 // a dimmer (only) fixture or not.
       uint8             IdentifierCount         // The number of following identifier blocks.
       {
          uint8             IdentifierType       // 0x05 = RDMManufacturerId (uint16)
                                                 // 0x00 = RDMDeviceModelId (uint16)
                                                 // 0x01 = RDMPersonalityId (uint64 - see note!)
                                                 // 0x02 = AtlaBaseFixtureId (guid)
                                                 // 0x03 = AtlaBaseModeId (guid)
                                                 // 0x04 = CaptureInstanceId (guid)
          uint16            IdentifierDataSize   // The size of the data following.
          uint8[]           IdentifierData       // Identifier type specific data.
       }[IdentifierCount]
       uint8             Patched                 // A boolean 0x00 or 0x01 indicating whether the
                                                 // the fixture is patched or not.
       uint8             Universe                // The (0-based) universe index.
       uint16            UniverseChannel         // The (0-based) DMX channel.
       ucs2              Unit                    // The unit number.
       uint16            Channel                 // The channel number.
       ucs2              Circuit                 // The circuit number.
       ucs2              Note                    // Any notes.
       float[3]          Position                // The 3D position.
       float[3]          Angles                  // The 3D angle.
    }[FixtureCount]
}
```

Note: The RDMPersonalityId IdentifierType is incorrectly uint64 when it should have been uint16. As a result of this, the highest six bytes should be set to zero.

Note: The 3D coordinate system is right-handed with Z being downstage and Y up. Angles are Euler angles in radians, Tait-Bryan X1 Y2 Z3 order.

5.5 CAEX / FixtureIdentify message

This message can be sent to Capture in order to set the fixture identifier of a fixture based on its instance id (the 0x04 identifier type of the FixtureList message). This is only required in the specific scenario of a fixture being added in Capture and then patched on the console. The console can then respond to the addition of the fixture with a FixtureIdentify message before it sends the FixtureModify message (which relies on the identifier having been set).

```
CITP_CAEX_FixtureIdentify
{
    CITP_CAEX_Header   CITPCAEXHeader           // CITP CAEX header, ContentCode = 0x00020204.
    uint16             FixtureCount             // The number of fixture identifications following.
    {
       guid               CaptureInstanceId      // The instance id of the fixture.
       uint32             FixtureIdentifier      // The fixture identifier to set.
    }[FixtureCount]
}
```

5.6 CAEX / FixtureModify message

This message is sent unsolicited by both Capture and the peer whenever a fixture has been modified. All fields must always be present, but it is important that the ChangedFields field indicates which have actually been modified.

```
CITP_CAEX_FixtureModify
{
    CITP_CAEX_Header   CITPCAEXHeader           // CITP CAEX header, ContentCode = 0x00020202.
    uint16             FixtureCount             // The number of fixtures following.
    {
       uint32             FixtureIdentifier     // Console's fixture identifier.
       uint8              ChangedFields         // Bitmask indicating which of the following fields have changed.
       uint8              Patched               // As in FixtureList. (ChangedFields & 0x01).
       uint8              Universe              // As in FixtureList. (ChangedFields & 0x01).
       uint16             UniverseChannel       // As in FixtureList. (ChangedFields & 0x01).
       ucs2               Unit                  // As in FixtureList. (ChangedFields & 0x02).
       uint16             Channel               // As in FixtureList. (ChangedFields & 0x04).
       ucs2               Circuit               // As in FixtureList. (ChangedFields & 0x08).
       ucs2               Note                  // As in FixtureList. (ChangedFields & 0x10).
       float[3]           Position              // As in FixtureList. (ChangedFields & 0x20).
       float[3]           Angles                // As in FixtureList. (ChangedFields & 0x20).
    }[FixtureCount]
}
```

5.7 CAEX / FixtureRemove message

This message is sent unsolicited by both Capture and the peer whenever on or more fixture(s) have been removed.

```
CITP_CAEX_FixtureRemove
{
    CITP_CAEX_Header      CITPCAEXHeader       // CITP CAEX header, ContentCode = 0x00020203.
    uint16                FixtureCount         // The number of fixture identifiers following.
    uint32[FixtureCount]  FixtureIdentifier    // Console's fixture identifiers.
}
```

5.8 CAEX / FixtureSelection message

This message is sent unsolicited by by both Capture and the peer to inform the other side of a new fixture selection. Note that the order of the fixture identifiers should carry the order in which the fixture(s) were selected.

```
CITP_CAEX_FixtureSelection
{
    CITP_CAEX_Header      CITPCAEXHeader       // CITP CAEX header, ContentCode = 0x00020300.
    uint16                FixtureCount         // The number of fixture identifiers following.
    uint32[FixtureCount]  FixtureIdentifier    // Console's fixture identifiers.
}
```

5.9 CAEX / FixtureConsoleStatus message

This message is sent unsolicited by the peer to Capture in order to convey "live information" data that can be displayed by Capture.

```
CITP_CAEX_FixtureConsoleStatus
{
    CITP_CAEX_Header   CITPCAEXHeader          // CITP CAEX header, ContentCode = 0x00020400.
    uint16             FixtureCount            // The number of fixtures following.
    {
       uint32             FixtureIdentifier    // Console's fixture identifiers.
       uint8              Locked               // The fixture has been locked from manipulation.
       uint8              Clearable            // The fixture has a clearable programmer state.
    }[FixtureCount]
}
```

6. Laser feed messages

A peer may serve laser feeds to Capture. Information to Capture about which feeds are available and information from Capture about which feeds to transmit is sent over the TCP based CITP session. Actual feed frame data is transmitted to the UDP based CITP multicast address.

In order for Capture to be able to correlate the feed frames with the appropriate session, a process instance unique and random "source key" is to be generated by the laser controller.

6.1 CAEX / GetLaserFeedList message

This message is sent by Capture upon connection to determine what laser feeds are available. Receving this message is an indication of Capture's ability to understand CAEX laser feeds.

```
CITP_CAEX_GetLaserFeedList
{
    CITP_CAEX_Header   CITPCAEXHeader       // CITP CAEX header, ContentCode = 0x00030100.
}
```

6.2 CAEX / LaserFeedList message

This message can be sent to Capture both in response to a GetLaserFeedList message as well as unsolicited if the list of available laser feeds has changed.

```
CITP_CAEX_LaserFeedList
{
    CITP_CAEX_Header   CITPCAEXHeader       // CITP CAEX header, ContentCode = 0x00030101.
    uint32             SourceKey            // The source key used in frame messages.
    uint8              FeedCount            // The number of laser feed listings that follow.
    {
       ucs2               Name              // The name of the feed.
    }[FeedCount]
}
```

6.3 CAEX / LaserFeedControl message

This message is sent by Capture to indicate whether it wishes a laser feed to be transmitted or not. The frame rate can be seen as an indication of the maximum frame rate meaningful to Capture.

```
CITP_CAEX_LaserFeedControl
{
    CITP_CAEX_Header   CITPCAEXHeader       // CITP CAEX header, ContentCode = 0x00030102.
    uint8              FeedIndex            // The 0-based index of the feed.
    uint8              FrameRate            // The frame rate requested, 0 to disable transmission.
}
```

6.4 CAEX / LaserFeedFrame message

This message is sent unsolicited to Capture, carrying feed frame data.

```
CITP_CAEX_LaserFeedFrame
{
    CITP_CAEX_Header   CITPCAEXHeader       // CITP CAEX header, ContentCode = 0x00030200.
    uint32             SourceKey            // The source key as in the LaserFeedList message.
    uint8              FeedIndex            // The 0-based index of the feed.
    uint32             FrameSequenceNo      // A 0-based sequence number for out of order data detection.
    uint16             PointCount           // The number of points that follow.
    {
       uint8              XLowByte          // The low byte of the x coordinate.
       uint8              YLowByte          // The low byte of the y coordinate.
       uint8              XYHighNibbles     // The high nibbles of the x and y coordinates.
       uint16             Color             // The colour packed as R5 G6 B5.
    }[PointCount]
}

Such that:
    Point.X [0, 4093] = Point.XLowByte + (Point.XYHighNibbles & 0x0f) << 8
    Point.Y [0, 4093] = Point.YLowByte + (Point.XYHighNibbles & 0xf0) << 4
    Point.R [0, 31] = Point.Color & 0x001f
    Point.G [0, 63] = (Point.Color & 0x07e0) >> 5
    Point.B [0, 31] = (Point.Color & 0xf800) >> 11
```