New LXF Format

As seen in a Hex Editor, this is what the new LXF format looks like, as seen in the HD and Preditor product lines of Nexio. The Notes section displays notations that match descriptions from our own LXF interface document and the packet header source code file. Each header has a link to source code that is able to read this file format. You will find those pieces of source code immediately following this table.

Follow this link to view the OldLxf format.

Byte Count	Bytes	Hex	Notes
		Packet Header	
00 - 07	8	4C 45 49 54 43 48 00 00	[1]"LEITCH" in ASCII, padded with two 00's
08 - 11	4	01 00 00 00	[2]Version of this structure: 1=advanced (the old LXF format = 0)
12 - 15	4	48 00 00 00	[3]Size of header: 72 Dec (48 Hex)
16 - 19	4	?? 00 00 00	[4]Packet type: video=0, audio=1, or segment=2
20 - 23	4	00 00 00 00	[5]Unique stream identifier: currently always 00
24 - 31	8	Variable	[6]Start field number: 8-byte _int64 units represented in 720 kHz units (NTSC frame = 24024, PAL frame = 28800, Audio = Number of samples * 15)
32 - 39	8	Variable	[7]Duration in _int64 units represented in 720 kHz units: Segment Packets display the entire clip length, Video Packets are always one frame's worth of material (24024 NTSC, 28800 PAL), and Audio Packets will typically be 10 frames of material for NTSC and 2 frames of material for PAL.
		Video Packet Header	
			4 bytes representing 32 bits of value (see below). Take Byte values, enter in Little Endian order in calculator, convert to Binary, read in Little Endian order last setting

40 -43	4	variable	first. Example: 12 C8 03 00, entered in calculator as 03 C8 12, produces 00000000 00000011 11001000 00010010, which gets split up like 00000000 00 00001111 001 0000001 0010
	4b	From 0000 - 1001	[v1]Video format: JPEG = 0, MPEG1 = 1, Mpeg2 [4:2:0] = 2, Mpeg2 [4:2:2] = 3, Dv25 [4:1:1] or [4:2:0] = 4, DVCPRO = 5, DVCPRO50 = 6, uncompressed KRGB 8 bits = 7, uncompressed K 16 bits = 8, Mpeg2 [4:2:2] Constrained Bytes per GOP = 9
	7b	From 0000001 - 0001111	[v2]N Value of GOP
	3b	From 001 - 011	[v3]M Value of GOP
	8b	From 00000100 - 00110010	[v4]Video Bit Rate
	2b	From 00 - 11	[v5]Picture Type of current frame: I-frame (closed) = 0, I-frame (open) = 1, P-frame = 2, B-frame = 3
	8b	00000000	[v6]Reserved
44 - 47	4	Variable	**[V7]**Bytes of video data to follow header
48 - 51	4	00 00 00 00	**[V8]**00's
52 - 55	4	Variable	**[V9]**Bytes of VBI data to follow header, preceding video data
56 - 59	4	00 00 00 00	**[V10]**00's
60 - 63	4	00 00 00 00	**[V11]**Bytes of metadata to follow header, preceding video & VBI data
64 - 67	4	Variable	Checksum values
68 - 71	4	00 00 00 00	4 nulls to provide 8-byte alignment
		Audio Packet Header	
40 - 43	4		4 bytes representing 32 bits of value (see below). Take Byte values, enter in Little Endian order in calculator, convert to Binary, read in Little Endian order last setting

			first.
	6b	010000, 010100, 011000, or 100000	[A3]Sample size, in bits: 16, 20, 24, or 32
	6b	Variable	[A4]Sample precision, in bits, in the case that the audio samples are aligned to other variables; for example, 20 bits stuffed into 3 bytes.
	20b	0	[A5]Reserved
44 - 47	4	Variable	[A6]Track mask: a bitmap indicating which of 32 possible tracks are included after this header. (Tracks are included from low to high.) For example, 0F=1111= audio channels 1-4 enabled
48 - 51	4	Variable	[A7]Track size: in bytes, the stride between the starts of two consecutive audio tracks in the data area. This number may exceed the active space taken up by the samples due to alignment considerations. Total data that follows this header should equal track size * number of bits set to one in track mask.
52 - 63	12	00's	00's
64 - 67	4	Variable	Checksum values
68 - 71	4	00 00 00 00	4 nulls to provide 8-byte alignment
		Segment Packet Header	
40 - 43	4	01 00 00 00	[S1]Segment Packet Format = 1 always
44 - 47	4	78 00 00 00	[S2]Data size = 120 in Dec
48 - 51	4	Variable	[S3]Extended fields size: current bytes of data stored
52 - 63	12	00's	00's
64 - 67	4	Variable	Checksum values
68 - 71	4	00 00 00 00	Null padding to fill out header to 8-byte alignment
		Disk Segment	
72 - 75	4	Variable	[D1]Previous of the Back & Forth links

76 - 79	4	Variable	[D2]Next of the Back & Forth links
80 - 83	4	Variable	[D3]Video clusters - usually small values
84 - 87	4	Variable	[D4]Audio clusters - usually smaller values (1 or 2)
88 - 95	8	Variable	[D5]8-byte ID
96 - 99	4	Variable	[D6]Min Frame: lower limit for the material, used in conjunction with Start to fully define the first playable frame
100 - 103	4	Variable	[D7]Start: beginning time code
104 - 107	4	Variable	[D8]Duration: stated in frames
108 - 111	4	Variable?	[D9]Timecode Offset
112-115	4	variable 0 1 00011110 001 0000001 0011	[D10]4 bytes representing 32 bits of value (see below). Take Byte values, enter in Little Endian order in calculator, convert to Binary, read in Little Endian order last setting first. Example: 12 08 43 00, entered in calculator as 43 08 12, produces 00000000 01000011 00001000 00010010, which gets split up like 000000000 1 00001100 001 0000001 0010
	4b	From 0000 - 1111	[D11]Video format: JPEG = 0, MPEG1 = 1, Mpeg2 [4:2:0] = 2, Mpeg2 [4:2:2] = 3, Dv25 [4:1:1] or [4:2:0] = 4, DVCPRO = 5, DVCPRO50 = 6, uncompressed KRGB 8 bits = 7, uncompressed K 16 bits = 8, Mpeg2 [4:2:2] Constrained Bytes per GOP (IMX) = 9, None (Audio only) = 15
	7b	From 0000001 - 0010000	[D12]N Value of GOP
	3b	From 001 - 011	[D13]M Value of GOP
	8b	From 00000100 - 00110010	[D14]Video Bit Rate
	1b	From 0 - 1	[D15]VBI present
	9b	000000000	[D16]Reserved

116-119	4	Variable	[D17]Base segment of the alias group
120-123	4	Variable	[D18]Previous segment of the alias group
124-127	4	Variable	[D19]Next segment of the alias group
128-129	2	Variable	[D20]Record date stored in 2-byte structure
130-131	2	Variable	[D21]Kill date stored in 2-byte structure
132	1	??	[D22]Timecode type - 0 & 1 are NTSC, 2 = PAL
133	1	??	[D23]Status
134	1	??	[D24]Disk
135-160	26	Variable	[D25]Description: 25-byte field plus 1
161-176	16	Variable	[D26]Agency: 15-byte field plus 1
177-182	6	Variable	[D27]Type: 5-byte field plus 1
183	1		[D28]Video Gain
184	1		[D29]Video Setup
185	1		[D30]Chroma Gain
186	1		[D31]Hue LSB
187	1	00	Reserved
188-191	4	variable	4 bytes representing 32 bits of value (see below). Take Byte values, enter in Little Endian order in calculator, convert to Binary, read in Little Endian order last setting first. Example: 92 00 00 00, entered in calculator as 00 00 00 92, produces 00000000 00000000 00000000 10010100, which gets split up like 00000000000 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0100 00
	2b	00 - 11	[D32]Hue MSB (previously videoContent(=-2); a value <2 indicates that the procamps

			are valid)
	4b	0000 - 1000	D33Number of Audio Tracks (0-8)
	1b	0 or 1	D342Write Protected bit (Delete Protected)
	1b	0 or 1	D35Allocated bit (belongs to the segment list)
	1b	0 or 1	D36Sliding bit (Looping record)
	1b	0 or 1	D37tcTranslate bit
	1b	0 or 1	D38Invisible bit (Hidden)
	1b	0 or 1	D39Macro bit (Media ID or Consolidated clip)
	1b	0 or 1	D40Alpha bit (Still, roll or crawl)
	1b	0 or 1	D41Project bit (NewsFlash file)
	1b	0 or 1	D42Purged bit (Consolidated clip)
	1b	0 or 1	D43Reference bit
	1b	0 or 1	D44Looping bit
	1b	0 or 1	D45Not Ready To Play bit
	1b	0 or 1	D46Not Ready To Transfer bit
	1b	0 or 1	D47Not Ready To Archive bit
	1b	0 or 1	D48Transfer In Progress bit
	11b	000 00000000	D49Reserved
		Extended Fields	
192	1	08	
193 - 200	8	Variable	Record time-date stamp
201	1	??	Size of next field

202 - Next field

Packet Header

```
class PACKET_HEADER
{
public:
// Identifies this structure as an LXF header. Should contain "LEITCH"
// followed by zero padding.
char logo[8]; [1]

// Version of this structure. The version number will be incremented
// with subsequent versions:
// - 0 = current version, start and duration are int's, in fields.
// - 1 = advanced version, start and duration will be __int64's, in (1/27 MHz) units.
int version;[2]

// Size of this header in bytes.
ULONG size;[3]

// Specifies what type of payload is carried in the packet (i.e. audio,
// video, segment). This must be one of:
//
// PACKET_TYPE_VIDEO
// PACKET_TYPE_AUDIO
// PACKET_TYPE_SEGMENT

PACKET_TYPE type;[4]
```

```
// All LXF packets associated with a particular audio or video stream
// must have a unique stream identifier. For the current VR file system must be 0 because only one video
// and one audio stream per file are supported. Multiple audio services can be supported by embedding them
// in different AES tracks (32 available).

int stream;[5]

// Currently the field number at which presentation of the LXF packet
// begins. For now these values increase by 2 for each subsequent video
// frame, 10 for each audio frame.
//
// This will be changing in the near future.
int start;[6]

// Presentation duration of the current LXF packet in fields. Currently,
// this field is set to 2 for video (normal interlaced), set to 10 for
// audio (8008 samples at 48khz = 5 interlaced frames = 10 fields).
//
// This will be changing in the near future.
int duration;[7]
```

Next sections depend on if the the packet type is video, audio or segment.

Video Packet: always contains one frame or two fields of data

```
// For type == PACKET_TYPE_VIDEO
//
// The header will be immediately followed in this order by:
//
```

```
// info.meta.pictureSize bytes of metadata.
// info.vb.pictureSize bytes of vertical blanking data
// info.active.pictureSize bytes of coded video data
//
struct
{
struct
{
// Specifies video frame format. Must be one of the
// enum VIDEO_FORMAT constants. For transmission applications
ULONG format:4;V1

// These fields (N, M) indicate GOP structure. We have been
// recommended to code them as 15, 3 respectively. My understanding
// is that they are used as hints, but do not need to accurately
// specify what is actually coded.
//
// GOP length (distance between I frames, not GOP headers)
ULONG N:7;V2 // GOP size

// Distance between anchor (reference) pictures.
ULONG M:3;V3 // reference picture period

// Average Bitrate of the coded video in Mbits/s=1000000's of bits/s
ULONG bitRate:8;V4

// MPEG picture_coding_type with added value of zero to specify
// an I-frame at a closed GOP.
ULONG pictureType:2;V5 // 0 - I (closed) , 1 - I (open), 2 - P , 3 - B

ULONG reserved:8;V6
```

```
// Amount of coded video data (in bytes) immediately following this
// LXF header.
ULONG pictureSize;V7
} active;

struct
{
ULONG format;V8 // 0

ULONG pictureSize;V9 // Amount of VBI data (in bytes) immediately preceding the coded data, use 0 for ATSC.
} vb;

struct
{
ULONG format;V10 // 0

ULONG pictureSize;V11 // Amount of metadata (in bytes) immediately preceding the VBI data, use 0 for ATSC.
} meta;

} video;
```

Audio Packet: always contains five frames (8008 samples X number of channels) of data

```
//
// For type == PACKET_TYPE_AUDIO
//
```

```
struct
{
int firstField;
```

Segment Packet

```
// For type == PACKET_TYPE_SEGMENT
//
struct
{
ULONG format;[S1]   // for ATSC use SEGMENT_PACKET_FORMAT_1
ULONG dataSize;[S2]
// end of SEGMENT_PACKET_FORMAT_0
ULONG extendedFieldsSize;[S3]   // for ATSC use 0
// end of SEGMENT_PACKET_FORMAT_1

// The data that follows consists of a DISK_SEGMENT structure followed in the case of SEGMENT_PACKET_FORMAT_1
// by the extended fields area (extendedFieldsSize bytes). The extended fields area is a concatenation of
// all the extended fields (including the empty ones). Each extended field is represented as a 1-byte byte
// count (0-255) followed by the actual data bytes.
} segment;

} info;
```

Next section is the DISK_SEGMENT referred to in PACKET_TYPE_SEGMENT

```
struct DISK_SEGMENT
{
int prev;D1   // back and forth links: for better symmetry in the routines
int next;D2   // the in use segments on the disk are linked in a ring whose
// "initial" point is the ROOT_SEGMENT, at index=0

int videoClusters;D3  //number of video clusters associated with the segment
int audioClusters;D4  //number of audio clusters associated with the segment

ID id;D5   // The VR ID structure - See separate definition below for ID Structure

// playback procamp values
CHAR videoGain; D28
CHAR videoSetup; D29
CHAR chromaGain; D30// previously resolution LSB
UCHAR hueLSB; D31// previously resolution MSB
UCHAR reserved0;
UINT hueMSB : 2; D32// previously videoContent(=-2); a value <2
// indicates that the procamps are valid
UINT audioTracks:4;D33
UINT writeProtected : 1 ;D34
UINT allocated : 1 ; D35// belongs to the segment list
UINT sliding : 1 ;D36
UINT tcTranslate : 1 ;D37
UINT invisible : 1 ;D38
UINT macro : 1 ;D39
UINT alpha : 1 ;D40
UINT project : 1 ;D41
```

```
UINT purged : 1 ;D42
UINT reference : 1 ;D43
UINT looping : 1 ;D44
UINT notReadyToPlay : 1 ;D45
UINT notReadyToTransfer : 1 ;D46
UINT notReadyToArchive : 1 ;D47
UINT transferInProgress : 1 ;D48
UINT reserved:11;D49
};

class DiskSegment?
{
public:

};
```

Next section is the ID Structure referred to in DISK_SEGMENT

```
typedef struct _ID
{
UCHAR code[ID_CODE_SIZE]; D5// 8 format determined by controller
ULONG minFrame; D6// lower limit for the material, used in
// conjunction with start to fully define the
// 1st playable frame
int start; D7// 4 beginning <new> time code ( inclusive )
```

```
ULONG duration; D8// 4 length of recorded material
int tcOffset; D9
VIDEO_PARAMS video; D10 //see below for Video Params
int base; D17// base segment of the alias group
int prev;D18 // prev segment in the alias group
int next; D19// next segment in the alias group
VRDATE recordDate; D20// 2 date id was recorded on vr - see below for 2 byte structure of the date field
VRDATE killDate;D21 // 2 date id expires
UCHAR tc_type;D22 // 1 timecode type
UCHAR status; D23// 1 status of id
UCHAR disk; D24// 1 disk ( index to Disk array )
char description[ID_DESCRIPTION_SIZE+1]; D25// 26 description
char agency[ID_AGENCY_SIZE+1]; D26 // 16 material source agency
char type[ID_TYPE_SIZE+1]; D27// 6 commercial/PSA/etc.
} ID;
```

Next section is the Video Parameters referred to in ID Structure

```
struct VIDEO_PARAMS
{
ULONG format:4;
ULONG N:7; // GOP size
ULONG M:3; // reference picture period
ULONG bitRate:8;
ULONG vbiPresent:1;
```

```
ULONG aspectRatio:1;
ULONG reserved:8;
};
```

Next section is the VR Date Structure referred to in ID Structure

```
typedef struct _VRDATE
{
USHORT year:7;
USHORT month:4;
USHORT day:5;
} VRDATE;
```