# Dxd File Format

Beschreibung des Szene/Daten (*.dxd) Formates zum Abspielen von
DMX Daten des Interface **SOUNDLIGHT USBDMX-TWO** V1.3 ab Firmware 1.3.3325

## Inhaltsverzeichnis

**INHALTSVERZEICHNIS**
**INTRO**
**FILE HEADER**
**CHUNK**
**CHUNK „SCENE“**
**CHUNK „DATA“**
**CHUNK „CHANNEL MASK**
**CHUNK „APPDATA**
**ANMERKUNGEN**

## Intro

Das Format einer Datendatei für das Interface USBDMX-TWO besteht aus einem Dateiheader und darauf folgenden Bereichen (sog. „Chunks“), die in einer beliebigen Reihenfolge angeordnet sein können. Pro dxd-File muß ein Scene-Chunk und ein Data-Chunk enthalten sein.

Jeder Type eines Chunks darf, mit Ausnahme von AppData, maximal einmal pro dxd-File vorkommen. Wobei AppData & ChannelMask nicht vorkommen müssen. Der AppData Chunk und der ChannelMask Chunk sind also optional.

Der AppData Chunk ist dafür gedacht, Daten, die zum Erstellen des dxd-Files für die erstellende Software notwendig sind, mit im dxd file zu hinterlegen, um später beim Ändern/Bearbeiten des dxd-Files diese Daten wieder zu Rate zu ziehen. Dort könnte z.B. ein Setup oder Patch mit hinterlegt werden. Wie diese Daten interpretiert werden, ist der jeweiligen erstellenden Software überlassen. Empfohlen ist aber das Descriptionfeld mit den Namen der erstellenden Software zu starten, und danach den Identifier für die Daten (z.B. PATCH, oder SETUP) anzufügen.

Der ChannelMask Chunk ist dazu gedacht, ein Mask zu dem in DatenChunk hinterlegten DMX-Daten zur Verfügung zu stellen, die das abspielende Gerät auswertet und z.B. beim ändern des Masters verwendet, um nur Farbe & Dimmer-Wert zu verändern.

Der Scene Chunk ist dazu gedacht, bestimmte Informationen, die die DMX-Daten betreffen, zu hinterlegen. Dazu zählen z.B. Autor, Erstellungsdatum…usw. Sie sollen das spätere Identifizieren der dxd-Datei einfacher gestalten.

Der DataChunk enthält Informationen zu den DMX Daten, sowie alle DMX-Daten, die in DataFrames verpackt sind. Im DataChunk werden nur Dataframes eines Typs. verwendet.

## File Header

| Byte | Size | Field Name | Field Description | Field Content |
|---|---|---|---|---|
| 1 - 8 | 8 | File Identifier | Id to recognize a Plexus data file (“DMXDATA\0”) | 0x44 0x4D 0x58 0x44<br>0x41 0x54 0x41 0x00 |
| 9, 10 | 2 | File Version | Version number | 0x01 0x00 (v1.0) |

## Chunk Header

Jeder Chunk beginnt mit einem Chunk-Header, der wie folgt aufgebaut ist:

| Byte | Size | Field Name | Field Description | Field Content |
|---|---|---|---|---|
| 1, 2 | 2 | Chunk Header Length | Length of the chunk header | Low byte first<br>Inclusive ChunkHeaderLen<br>0x16, 0x00 |
| 3 -10 | 8 | Chunk Lenght | Length of the whole chunk<br><br>ChunkLength | Low byte first<br>Inclusive ChunkHeaderLen and |
| 11 ,12 | 2 | Chunk Type | Identifier for type of chunk | 0x00,0x00=CHUNK_TYPE_IGNORE<br>0x01,0x00=CHUNK_TYPE_SCENE<br>0x02,0x00=CHUNK_TYPE_<br>CHANNELMASK<br>0x04,0x00=CHUNK_TYPE_DATA<br>0x08,0x00=CHUNK_TYPE_APPDATA |
| 13 | 1 | Chunk version Major | Major Version number of chunk type | Major ( v**1**.0)<br>0x01 |
| 14 | 1 | Chunk version Minor | Minor Version number of chunk type | Minor ( v1.**0**)<br>0x00 |
| 15-22 | 8 | Future extension | This is space for future extensions | 0x00, 0x00, 0x00, 0x00<br>0x00, 0x00, 0x00, 0x00 |

## Chunk „Scene“

Der Chunk “Scene” beginnt mit dem Chunk Header, gefolgt von den Szenendaten. Die Versionsnummer für das Datenformat ist im Header definiert.

| Byte | Size | Field Name | Field Description | Field Content |
|---|---|---|---|---|
| 1, 2 | 2 | Chunk Header Length | Length of the chunk header | 0x16 0x00 |
| 3-10 | 8 | Chunk Lenght | Length of the whole chunk | 0x1E 0x01 0x00 0x00 0x00 0x00 0x00 0x00 |
| 11,12 | 2 | Chunk Type | Identifier for type of chunk | CHUNK_TYPE_SCENE 0x01,0x00 |
| 13,14 | 2 | Chunk version | Version number of chunk type | (V1.0) 0x01 0x00 |
| 15-22 | 8 | Future extension | This is space for future extensions | 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 |

## V1.0

So sehen die Szenedaten der Version 1.0 aus:

| Byte | Size | Field Name | Field Description | Field Content |
|---|---|---|---|---|
| 1-128 | 128 | Scene Name | In wide char format 16 bit chars Name for the Scene | WideChar [64] incl. end 0x00 0x00 |
| 129-192 | 64 | Generator | In wide char format 16 bit chars Name of software which creates this file | WideChar [32] incl. end 0x00 0x00 example “SCANSHOW_5_2\0” |
| 193-256 | 64 | Author | In wide char format 16 bit chars Name of the author | WideChar [32] incl. end 0x00 0x00 example “Username\0” |
| 257,258 | 2 | Date Year | Date of generation | Year, low Byte first 0xDB,0x07 (2011) |
| 259 | 1 | Date Month | Date of generation | Month 0x08 (August) |
| 260 | 1 | Date Day | Date of generation | Day 0x05 (5.) |
| 261 | 1 | Time Hour | Time of generation at Date | Hour |
| 262 | 1 | Time Minutes | Time of generation at Date | Minutes |
| 263 | 1 | Time Seconds | Time of generation at Date | Seconds |
| 264 | 1 | Reserved T | his is space for future extensions | 0x00 |

## Chunk „Data“

Auch hier kommt zuerst der Header mit dem Datentyp und der Versionsnummer.

| Byte | Size | Field Name | Field Description | Field Content |
|---|---|---|---|---|
| 1, 2 | 2 | Chunk Header Length | Length of the chunk header | 0x16 0x00 |
| 3-10 | 8 | Chunk Lenght | Length of the whole chunk | Low byte first |
| 11,12 | 2 | Chunk Type | Identifier for type of chunk | CHUNK_TYPE_DATA = 0x04 0x00 |
| 13,14 | 2 | Chunk Version | Version number of chunk type | 0x01 0x00 ( v1.0) |
| 15-22 | 8 | Future Extension | This is space for future extensions | 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 |

### V1.0

Zum besseren Verständnis sind die DMX Daten noch mal in Daten Frames gekapselt. Zuerst kommen die allgemeinen Formatbeschreibungen und dann die ein oder mehrere Daten Frames, die auch die DMX Daten enthalten

| Byte | Size | Field Name | Field Description | Field Content |
|---|---|---|---|---|
| 1,2 | 2 | Data Format | Identifier for the chunk intern | UNKNOWN = 0<br>0x00, 0x00<br>ONE_UNIVERSE = 1<br>0x01, 0x00<br>TWO_UNIVERSE= 2<br>0x02, 0x00 |
| 3, 4 | 2 | FrameRate InMiliseconds | Frame Rate in Milliseconds<br>Mit welcher Framerate getriggert werden sollte<br>Alle in FrameTime angegebenen Werte sollten Teiler dieses Wertes sein.<br>Dieser Wert ist nur zur Information, nicht für Berechnungen zu nutzen.<br>Wenn FrameTime kein Teiler ist, dann ist FrameTime zu benutzen | FrameRateInMiliseconds<br>Default ist 23ms= 43,5Hz<br>0x17, 0x00 |
| 5-12 | 8 | FrameCount | Anzahl aller Frames im DataFrame. Dieser Wert ist nur zur Information zu benutzen, z.B. um einen Überblick über die Dateiinformation anzuzeigen | Low byte first |
| 20-27 | 8 | FrameTime | Summe der Gesamtspielzeit aller Frames, wenn mit vorgegebener Framerate abgespielt wird. Dieser Wert ist nur zur Information zu benutzen, z.B. um einen Überblick über die Dateiinformation anzuzeigen | Low byte first |

| | | | | |
|---|---|---|---|---|
| 28,29 | 2 | ReverseStop | Stop length for Reverse reading | 0x00, 0x00 |
| 30-n | n | DataFrames | The data block, in a structure which is defined thru data format.<br>The length of the block can be calculated with the position and the whole chunk length | See DataFrame<br>ONE_UNIVERSE<br>or<br>TWO_UNIVERSE |
| 30+n+1<br>30+n+2 | 2 | ForwardStop | Stop len for Forward reading | 0x00, 0x00 |

Die Daten Frames haben folgende Struktur. Sie dient zum Einlesen eines Frames.
Die Datenaufteilung innerhalb eines Frames wird durch das Data Format aus dem Data Chunk bestimmt. Der Block kann auch verschlüsselt sein, was ebenfalls der Formattyp angibt.
Es können mehrere Data Frames hintereinander folgen. Um Frames zu überspringen, sollte DataFramesLength ausgelesen werden und dann um diesen Wert nach vorn gesprungen werden.
Dort kann dann die DataFramesLength des folgenden Frames ausgelesen werden. Falls dieser Wert 0 sein sollte, handelt es sich schon um den Wert von ForwardStop, der das Ende des einzulesenden Chunks angibt. Das gleiche funktioniert auch rückwärts, nur daß dann ReverseStop das Ende der Data Frames angibt und die Frames Größe aus DataFramesLengthBefore gelesen werden kann.

DataFrame DataFormat=ONE_UNIVERSE

| Byte | Size | Field Name | Field Description | Field Content |
|---|---|---|---|---|
| 1, 2 | 2 | DataFrameLength | Length of the DataFrame | Inclusive DataFrameLength Duration,StartChannel, ChannelCount,DataDMX, DataFramesLengthBefore<br>If use 512 DMX value then length=528=0x10, 0x02 |
| 3-6 | 4 | Duration | Time the frame is visible until next frame in ms | Duration in ms<br>Ein Mehrfaches von FrameRateInMiliseconds von Chunk data.<br>Werte kleiner FrameRateInMiliseconds sowie Wert 0 sind illegal |
| 7-10 | 4 | FadeTime | Time to fade from frame to frame, in ms | FadeOut Zeit startet nach abgelaufener Duration. |
| 11,12 | 2 | Startchannel | | Start channel of Universe<br>Value 0 = DMX slot #1<br>Startchannel = 0…511 |
| 13,14 | 2 | ChannelCount | X | Count of DMX values for Universe |
| 15-15+n | n | DmxData | Length of data = channelcount | |
| n+16, n+17 | 2 | DataFrameLength Before | Length of The DataFrame | The same value as DataFramesLength<br>Use it to reverse play frames |

DataFrame DataFormat=TWO_UNIVERSE

| Byte | Size | Field Name | Field Description | Field Content |
|---|---|---|---|---|
| 1,2 | 2 | DataFrameLength | Length of the DataFrame | Inclusive DataFrameLength Duration,StartChannel1&2, ChannelCount1&2,DataDMX, DataFramesLengthBefore If use 2x512 dmx value then length=1044=0x14, 0x04 |
| 3–6 | 4 | Duration | Time the frame is visible until next frame in ms | Duration in ms Ein mehrfaches von FrameRateInMiliseconds von Chunk data. Werte kleiner FrameRateInMiliseconds sowie Wert 0 sind illegal |
| 7-10 | 4 | FadeTime | Time to fade from frame to frame in ms | FadeOut Zeit startet nach abgelaufener Duration. |
| 11,12 | 2 | StartchannelU1 | | Start channel of Universe 1 Value 0 = DMX slot #1 StartchannelU1 = 0…511 |
| 13,14 | 2 | ChannelcountU1 | n | Count of DMX values for Universe 1 |
| 15,16 | 2 | StartchannelU2 | | Start channel of Universe 2 Value 0 = DMX slot #1 StartchannelU2 = 0…511 |
| 17,18 | 2 | ChannelcountU2 | m | Count of DMX values for Universe 2 |
| 19 – n+m+19 | n+m | DmxData | Length of data = channelcountU1 + channelcountU2 | |
| n+m+20, n+m+21 | 2 | DataBlockLength Before | Length of the Data block | The same value as DataFramesLength, use it to reverse play frames |

## Chunk „Channel Mask“

In dem “Channel Mask” Chunk werden die Kanal-Maskierungen für z.B. den Masterdimmer gespeichert. Auch hier zuerst der Chunk-Header und danach die Maskierungsdaten. Da in einem Channel-Mask-Chunk nur ein Universum abgebildet wird, kann dieser in einer Datei mehrmals vorkommen (mit unterschiedlichem Universe-Bezug).

| Byte | Size | Field Name | Field Description | Field Content |
|---|---|---|---|---|
| 1-2 | 2 | Chunk Header Length | Length of the chunk header | 0x16 0x00 |
| 3-10 | 8 | Chunk Lenght | Length of the whole chunk | 0x18, 0x04, 0x00, 0x00<br>0x00, 0x00, 0x00, 0x00 |
| 11,12 | 2 | Chunk Type | Identifier for type of chunk | CHUNK_TYPE_CHANNELMASK<br>0x02, 0x00 |
| 13,14 | 2 | Chunk version | Version number of chunk type | 0x01, 0x01 ( v2.0) |
| 15-22 | 8 | Future extension | This is space for future extensions | 0x00, 0x00, 0x00, 0x00<br>0x00, 0x00, 0x00, 0x00 |

Die Maskierungsdaten sind aufgeteilt in ein Channel Format, Flags und zusätzliche Data, welche schon mal für spätere Verwendungen vorgesehen sind.

| Byte | Size | Field Name | Field Description | Field Content |
|---|---|---|---|---|
| 1,2 | 2 | Universe | Universe of the mask data | 0x00, 0x00or 0x01, 0x00 |
| 3-514 | 512 | Channel Format | | 0x00 - NONE<br>0x01 - ITENSITY<br>0x02 - FOCUS // V2.0 not used<br>0x03 - COLOR // V2.0 not used<br>0x04 - BEAM // V2.0 not used |
| 515-1026 | 512 | Channel Flags | | 0x00 - NONE<br>0x01 - GROUP // V2.0 supported<br>0x02 - HTP // V2.0 not used<br>0x04 - INVERT // V2.0 not used<br>0x08 - CROSSFADE // V2.0 supported |
| 1027-1539 | 512 | Channel Data | If Channel Flag = GROUP then interprete this as group number. combination of groups is possible, otherwise ignore this data and set to 0x00 | If Channel Flag = GROUP<br>then interprete this as group number.<br>Group assignments:<br>0x00 =GROUP_NONE<br>0x01 =GROUP 1<br>0x02 =GROUP 2<br>etc until<br>0x80 =GROUP 8<br>combination of groups is possible |

## Chunk „AppData “

In dem “AppData” Chunk können Daten zur Applikation hinterlegt werden. Diese werden einfach als RAW-Daten hinterlegt. Das DMX Interface sollte diesen Chunk einfach überspringen. Dieser Chunk ist zur Informationswiedergewinnung gedacht. Eine Software kann hiermit zusätzliche Daten, wie PATCH oder SETUP hinterlegen, um diese später wieder auslesen zu können. Um zu unterscheiden, welche Daten hinterlegt wurden, sollte das Description-Feld eindeutig ausgefüllt werden.

z.B. „SLH_SETUP“, oder „SLH_PATCH“

| Byte | Size | Field Name | Field Description | Field Content |
|---|---|---|---|---|
| 1-2 | 2 | Chunk Header Length | Length of the chunk header | 0x16, 0x00 |
| 3-10 | 8 | Chunk Lenght | Length of the whole chunk | 0x?? 0x?? 0x?? 0x??<br>0x?? 0x?? 0x?? 0x?? |
| 11,12 | 2 | Chunk Type | Identifier for type of chunk | CHUNK_TYPE_APPDATA<br>0x08, 0x00 |
| 13,14 | 2 | Chunk version | Version number of chunk type | 0x01, 0x00 ( v1.0) |
| 15-22 | 8 | Future extension | This is space for future extensions | 0x00, 0x00, 0x00, 0x00<br>0x00, 0x00, 0x00, 0x00 |

### V1.0

Die Application Daten werden als RAW Daten geschrieben.

| Byte | Size | Field Name | Field Description | Field Content |
|---|---|---|---|---|
| 1-128 | 128 | Description | In wide char format 16 bit characters<br>Description of follow data | WideChar [64] incl. end 0x00, 0x00 example<br>“SLH_PATCH\0”<br>“SLH_SETUP\0” |
| 129-136 | 8 | DataLength | | Length of follow data<br>Low byte first |
| 137 – DL+137 | DL | Data | | Application data in RAW format |

## Anmerkungen:

pro File ist genau 1 FileHeader enthalten am Anfang der Datei
pro File ist 1 DataChunk enthalten
pro File ist ein oder keine Chunk Scene enthalten
pro File ist keine, eine, oder mehrere (pro Universum) MaskChunk enthalten
pro File ist kein, ein, oder mehrere AppData enthalten