Direct Stream Digital Interchange File Format

DSDIFF

Version 1.5

Philips

Intellectual Property & Standards

PHILIPS

Title:	Direct Stream Digital Interchange File Format, DSDIFF
Version:	1.5
Date:	2004-04-27

DISCLAIMER

Whereas Philips has taken care to ensure that the information contained in this document is accurate, this information is provided on an "as is" basis, without any warranty as to its completeness or accuracy. Royal Philips Electronics shall not be liable in any manner whatsoever for any damages, including direct, indirect or consequential, resulting from the use of this document or reliance on the accuracy of its contents.
Supply of this document does not confer any license under any intellectual property right of Royal Philips Electronics to use any of those rights in any apparatus, system or any components, subassemblies or software for such apparatus or system.

For any further explanation of the contents of this document, or in case of any perceived inconsistency or ambiguity of interpretation, please consult:

Philips Intellectual Property & Standards
System Standards and Technology Optical Storage and DRM / Super Audio CD
Building SFF-8
P.O. Box 80002
5600 JB Eindhoven
The Netherlands

Fax.: +31 40 27 32641

or send an E-mail to:

info.superaudiocd@philips.com

Contents

1 Introduction

1.1 PURPOSE AND SCOPE

Creating a Super Audio CD [ScarletBook] follows similar steps as conventional CD production – recording, editing, sound mastering and authoring. But Super Audio CD is using DSD or DST sound formats instead of PCM. To allow interchange of these sound formats between systems, a new file format has been defined. This document describes the definition of the DSDIFF file format. This file format is intended for storage of DSD or DST material.

1.2 DEFINITIONS, ACRONYMS AND ABBREVIATIONS

AIFF	**A**udio **I**nterchange **F**ile **F**ormat
ASCII	**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
CD	**C**ompact **D**isc
Channel Byte	8 consecutive DSD samples of 1 channel, the most significant bit is the oldest bit of the sequence
Clustered Frame	N Interleaved Channel Bytes, where N is the number of audio channels in the file
CRC	**C**yclic **R**edundancy **C**heck
Silence Pattern	A digitally generated DSD pattern with the following properties: • all Channel Bytes have the same value • each Channel Byte contains 4 bits equal to zero and 4 bits equal to one
DSD	**D**irect **S**tream **D**igital
DSDIFF	**D**irect **S**tream **D**igital **I**nterchange **F**ile **F**ormat
DSD File	DSDIFF file containing DSD audio
DST	**D**irect **S**tream **T**ransfer Format for lossless encoded DSD audio
DST File	DSDIFF file containing DST encoded audio
DST Frame	1/75 second of DST encoded audio The time corresponds to the Super Audio CD Frame time Actual length in bytes is variable
Fs	Sampling frequency for CD, being 44100Hz
IFF	**I**nterchange **F**ile **F**ormat
Mod	modulo operator: a mod b gives the remainder that results when a is divided by b
PCM	**P**ulse **C**ode **M**odulation
Super Audio CD Frame	1/75 second of audio (is equal to CD frame length) At 64×fs a frame covers 37632 DSD samples per channel

1.3 REFERENCES

[EA-IFF 85] Standard for Interchange Format Files
Electronics Arts, Jerry Morrison
January 14, 1985

[AIFF] Audio Interchange File Format
Apple Computer, Inc
Version 1.3, January 4, 1989

[ScarletBook] Super Audio CD Standard (Part 2)
Sony/Philips
Version 1.3, July 2002

[ITU] Multi-channel stereophonic sound system
recommendation : ITU-R BS.775-1 07/1994
International Telecommunication Union

1.4 DOCUMENT HISTORY

Date	Version	Most important updates
2000-02-14	1.0	Initial document
2000-09-05	1.3	Adaptations for Super Audio CD standard 1.1
2002-06-06	1.4	Added: ▪ Chunk overview of a Form DSD Chunk ▪ Undefined channel set-up value for Loudspeaker Configuration ▪ CRC polynomial for checksum on DST frames ▪ Revision information in File History ▪ Definition of TrackFlags in accordance with Super Audio CD standard 1.2 ▪ Requirements for Edited Master data Changed: ▪ Description of Edited Master Information Chunk (formerly referred to as Disc Information Chunk) ▪ Definition of programs and tracks in Marker Chunk ▪ ProgramStart marker replaces Disc Start marker Removed: ▪ Chunk recommendation (former Appendix A), requirements are covered by definition of an Edited master ▪ Description of usage of times codes (former appendix B), marker example illustrates the usage ▪ Definition of Disc End marker, which is not needed for the definition of programs
2002-11-28	1.4 / document revision 1	Added: ▪ The term "Post-roll" has been defined to denote the data between the last TrackStop and the end of the data Changed: ▪ It has been clarified that Comments, Artist and Title chunks are recommended (not required) in an Edited Master Removed: ▪ Recommendations for usage of File History - revision comment (from section 3.6) ▪ Recommendation for the marker offset value (from section 3.7.2)
2003-03-11	1.4 / document revision 2	Changed: ▪ The restrictions on audio contents for unavailable channels (channels for which a TrackFlag has been set) have been changed according to a modification to Super Audio CD standard version 1.3
2004-04-27	1.5	Added: ▪ Manufacturer Specific Chunk.

2 General description

2.1 PERSPECTIVE

In defining a new file format for DSD production tools it is useful to reuse as much as possible of the standard file formats that the audio industry uses today.

A commonly used file format for PCM is the Audio Interchange File Format, [AIFF], which conforms to the Electronics Arts Interchange File Format, [EA-IFF 85]. It enables storage of uncompressed or compressed sampled sound.

DSD requires a specific file format, since

- DSD is a one bit signal at a high data rate.
- a specific lossless compression technique called DST encoding is used for DSD signals.
- the maximum file size must be larger than 2 gigabyte (limit in AIFF).

Therefore the DSD interchange file format has been defined which conforms, as much as possible, to the principles of AIFF and EA-IFF 85 (with the exception that only one FORM chunk is allowed). This makes it possible to re-use large quantities of source code and libraries available today.

2.2 DATA TYPES, CONSTANTS AND NOTATIONS

In this document structures are described in a C-like notation, using the definitions listed below:

Definition	Meaning
char	8 bit signed integer.
uchar	8 bit unsigned integer.
text byte	char with value within the ASCII range of 0x20 through 0x7E.
ushort	16 bit unsigned integer.
ulong	32 bit unsigned integer.
double ulong	64 bit unsigned integer.
ID	32 bit, a concatenation of four printable ASCII characters in the range ' ' (space, 0x20) through '~'(0x7E). Space (0x20) cannot precede printing characters; trailing spaces are allowed. Control characters are forbidden. The ID is case sensitive. Therefore ID's can be compared using a simple 32-bit equality check.
pad byte	char with value 0, used for making chunks an even length.
Struct	Set of variables forming one entity.

All data is stored in Big Endian format, this means that the most significant byte of e.g. a ushort or ulong is stored first.

Decimal values are referred to as a string of digits, for example 123, 0, 100 are all decimal numbers. Hexadecimal values are preceded by 0x, e.g. 0x0A12, 0x1, 0x64.

In addition, a small part of the Backus Naur Format notation is used. The definitions of this notation used in this document are:

<> = known definition
[a|b] = choice between a or b
+ = 1 or more occurrences

Furthermore all defined variable names are written in italic.

2.3 FILE STRUCTURE

The EA-IFF 85 (see [EA-IFF 85]) defines an overall structure for storing data in files. This document recaps those portions of EA-IFF 85 that are germane to DSDIFF.

A file is built up from a number of *chunks* of data. Chunks are the building blocks of DSD files. A chunk consists of some header information followed by data:

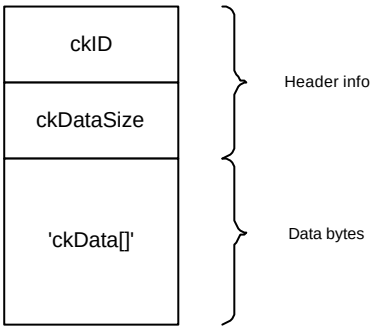

Figure 1: logical representation of a chunk

A chunk can be represented using the C-like language in the following way:

```
typedef struct {
        ID              ckID;           // chunkid
        double ulong    ckDataSize;     // chunk data size, in bytes
        uchar           ckData[];       // data
} Chunk;
```

ckID describes the format of the chunk's *data* portion. An application can determine how to interpret the chunk data by examining *ckID*.

ckDataSize is the size of the *data* portion of the chunk, in bytes. It does not include the 12 bytes used by *ckID* and *ckDataSize*.

ckData[] is the data stored in the chunk. The format of this data is determined by *ckID*. If the data is an odd number of bytes in length, a pad byte must be added at the end. The pad byte is not included in *ckDataSize*.

Note that the *ckDataSize* is **not** 4 bytes in length, as in EA-IFF 85, but 8 bytes. This makes it possible to create chunks larger than 2 GigaByte (up to 2^{64} bytes).

The chunks of DSDIFF are grouped together in a container chunk. EA-IFF 85 defines a number of container chunks. For DSDIFF, an adapted FORM container chunk, called Form DSD Chunk, is used.

A Form DSD Chunk has the following format:

```
typedef struct {
      ID              ckID;              // 'FRM8'
      double ulong    ckDataSize;        // FORM's data size, in bytes
      ID              formType;          // 'DSD '
      Chunk           frm8Chunks[];
} FormDSDChunk;
```

The FORM chunk of *formType* 'DSD ' is called a *Form DSD Chunk*.

ckID is always 'FRM8'. This indicates that this is a Form DSD Chunk. Note that this FORM chunk is slightly different from EA IFF 85 (the *ckDataSize* is not a long but a double ulong). Using the Form DSD Chunk makes it possible to identify that all local chunks have a *ckDataSize* of 8 bytes in length.

ckDataSize contains the size of the data portion of the 'FRM8' chunk. Note that the data portion consists of two parts, *formType* and *frm8Chunks[]*.

formType describes the contents of in the 'FRM8' chunk. For DSDIFF files, *formType* is always 'DSD '. This indicates that the chunks within the FORM pertain to sampled sound according to this DSDIFF standard.

frm8Chunks[] are the chunks within the Form DSD Chunk. These chunks are called *local chunks* since their own *ckID*'s are local to (i.e. specific for) the Form DSD chunk. A Form DSD Chunk together with its local chunks make up a DSDIFF file.

Figure 2 is an example of a simple DSDIFF file. It consists of a single Form DSD Chunk that contains 3 local chunks, a Format Version Chunk, a Property Chunk and a Sound Data Chunk.

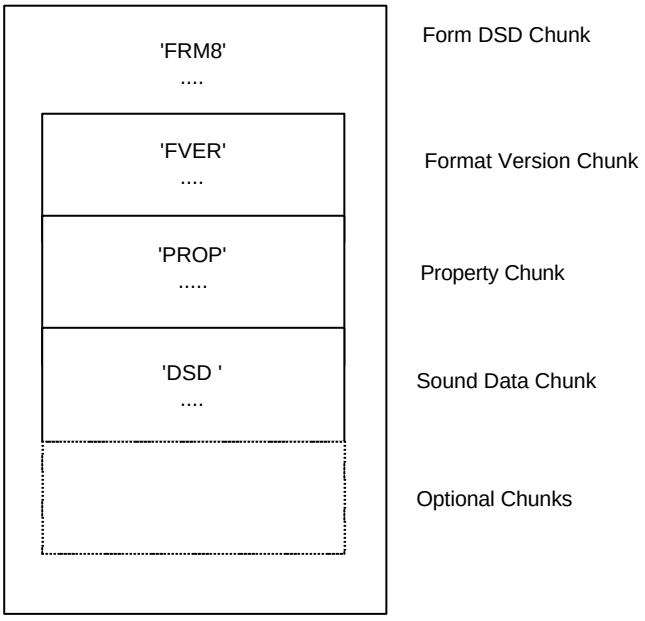

Figure 2: example of a simple DSDIFF file.

The official name for this file format is *Direct Stream Digital Interchange File Format.* Applications that need to present the name of this format to a user can abbreviate it to DSDIFF.
For an operating system that uses filename extensions, such as Windows or UNIX, it is recommended that DSDIFF file names have a ".DFF" extension.

2.4 HANDLING OF UNRECOGNIZED CHUNKS

When an application encounters a local chunk that is not recognised it must discard (skip) it while reading the Form DSD chunk. If an application copies the Form DSD Chunk without edits, it is nicer, but not necessary, to copy unrecognised chunks too. But if an application modifies the data in any way, then it **must** discard all unrecognised chunks: the writing application cannot guarantee that the unrecognised data is still consistent with the modified data.

3 Form DSD Chunk

The Form DSD Chunk is the container chunk that contains all the other (local) chunks. The format for the data within a Form DSD Chunk is shown below:

```
typedef struct {
      ID                ckID;              // 'FRM8'
      double ulong      ckDataSize;        // FORM's data size, in bytes
      ID                formType;          // 'DSD '
      Chunk             frm8Chunks[];      // local chunks
} FormDSDChunk;
```

The *ckID* is always 'FRM8'.

The *ckDataSize* is the summation of the sizes of the local chunks plus the size of *formType*. This is equal to the total file size in bytes minus the length of *ckID* and *ckDataSize* . It is always an even number because all chunks cover an even number of bytes.

The *formType* is always 'DSD '.

frm8Chunks[] are the local chunks. The order of the local chunks is chosen in such a way that streaming of a DSDIFF file is possible. At the definition of each local chunk it is indicated which chunk order should be maintained.
The local chunks are:

- Format Version Chunk (FVER)
- Property Chunk (PROP)
- DSD Sound Data Chunk (DSD)
- DST Sound Data Chunk (DST)
- DST Sound Index Chunk (DSTI)
- Comments Chunk (COMT)
- Edited Master Information Chunk (DIIN)
- Manufacturer Specific Chunk (MANF)

The Format Version Chunk and the Property Chunk are required in a Form DSD Chunk. The first local chunk must be the Format Version Chunk. One of the DSD or DST Sound Data Chunks is required. The Property Chunk must appear before DSD or DST Sound Data Chunk. All applications that use Form DSD Chunk must be able to read the required chunks and may (selectively) ignore the optional chunks.

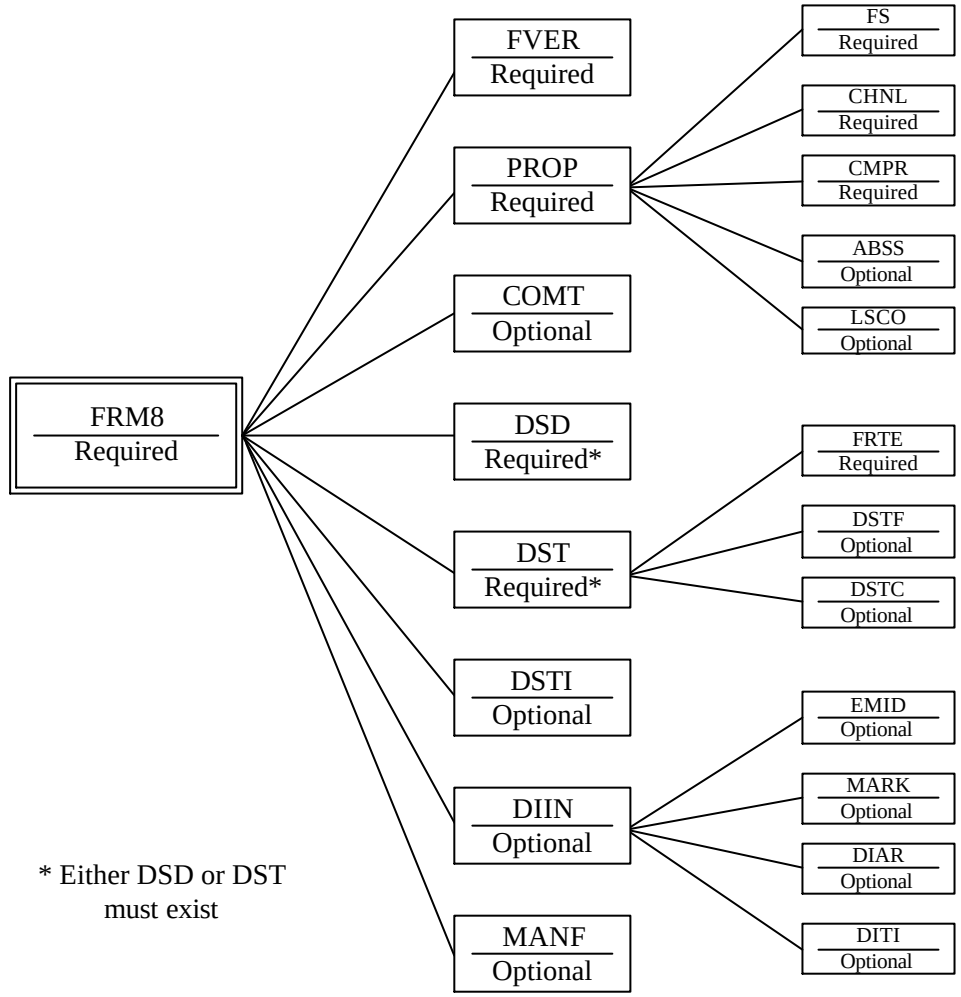

Figure 3: the Chunk Tree of a Form DSD chunk.

The Form DSD Chunk is required. It may appear only once in the file.

3.1 FORMAT VERSION CHUNK

The Format Version Chunk contains a field indicating the format specification version for the DSDIFF file. The format for the data within a Format Version Chunk is shown below:

```
typedef struct {
    ID              ckID;           // 'FVER'
    double ulong    ckDataSize;     // 4
    ulong           version;        // 0x01050000 version 1.5.0.0 DSDIFF
} FormatVersionChunk;
```

ckID is always 'FVER'.

ckDataSize is the size of the data portion of the chunk, in bytes. It does not include the 12 bytes used by *ckID* and *ckDataSize*. For this chunk, *ckDataSize* is 4.

version indicates the version id. It consists of 4 bytes and defines the version number of the file format (version 'byte1'.'byte2'.'byte3'.'byte4' ; from most significant byte to least significant byte).
The first byte (most significant byte) defines the "main version number".
The second byte defines the "addition version number", indicating additions to the description with respect to previous versions of this document. The possible changes are additional definitions and/or chunks.
The last two bytes of *version* are reserved for future use and are set to zero.
A DSDIFF reader with the same "main version number" can still read information but will not recognise additional chunk(s) that are defined in a later version of this document (backwards compatible).

The version number of this description is 1.5.0.0.

The Format Version Chunk can be used to check which chunks of the file are supported.

The Format Version Chunk is required and must be the first chunk in the Form DSD Chunk. It may appear only once in the Form DSD Chunk.

3.2 PROPERTY CHUNK

The Property Chunk is a container chunk, which consists of "local property chunks". These "local property chunks" define fundamental parameters of the defined property type.
The format for the data within a Property Chunk is shown below:

```
typedef struct {
      ID                  ckID;              // 'PROP'
      double ulong        ckDataSize;
      ID                  propType;          // 'SND '
      Chunk               propChunks[];      // local chunks
} PropertyChunk;
```

ckID is always 'PROP'. It indicates that this is the property container chunk (of type *propType*).

ckDataSize is the summation of the sizes of all local property *chunks* plus the size of *propType*. It does not include the 12 bytes used by *ckID* and *ckDataSize*.

propType defines the type to which the "local property chunks" belong. Defined types are:

propType	**Meaning**
'SND '	Sound properties

Other types are reserved for future use and may be defined in future versions.
The Property Chunk must precede the Sound Data Chunk.

propChunks[] are local property chunks. There is no order imposed on these local chunks.
The local chunks are :

- Sample Rate Chunk
- Channels Chunk
- Compression Type Chunk
- Absolute Start Time Chunk
- Loudspeaker Configuration Chunk

The Property Chunk is required and must precede the Sound Data Chunk. It may appear only once in the Form DSD Chunk.

3.2.1 Sample Rate Chunk

The Sample Rate Chunk defines the sample rate at which the sound data has been sampled.

```
typedef struct {
      ID                  ckID;              // 'FS  '
      double ulong        ckDataSize;        // 4
      ulong               sampleRate;        // sample rate in [Hz]
} SampleRateChunk;
```

ckID is always 'FS '.

ckDataSize is the size of the data portion of the chunk, in bytes. It does not include the 12 bytes used by *ckID* and *ckDataSize*. For this chunk, *ckDataSize* is 4.

sampleRate indicates how many samples fit in one second. Because DSD signals are 1 bit wide this number gives the total number of bits in one second per channel.

The Sample Rate Chunk is required and may appear only once in the Property Chunk.

3.2.2 Channels Chunk

The Channels Chunk defines the total number of channels and the channel ID's used in the Sound Data Chunk. The order of the channel ID's also determines the order of the sound data in the file.

```
typedef struct {
      ID                  ckID;              // 'CHNL'
      double ulong        ckDataSize;
      ushort              numChannels;       // number of audio channels
      ID                  chID[];            // channels ID's
} ChannelsChunk;
```

ckID is always 'CHNL'.

ckDataSize is the size of the data portion of the chunk, in bytes. It does not include the 12 bytes used by *ckID* and *ckDataSize*.

numChannels contains the number of audio channels in the file. A value of 1 means one channel, 2 means two channels, etc.. Any number, greater than zero, of audio channels may be represented.

chID[] defines the channel ID for each of the *numChannels* channels. The order in which they are stored in the array is the order in which they are stored in the Sound Data Chunk. There are a number of predefined channel ID's:

'SLFT'	: stereo left
'SRGT'	: stereo right
'MLFT'	: multi-channel left
'MRGT'	: multi-channel right
'LS '	: multi-channel left surround
'RS '	: multi-channel right surround
'C '	: multi-channel center
'LFE '	: multi-channel low frequency enhancement

['C000' .. 'C999']: context specific channel contents

To prevent misinterpretations the following restrictions apply :

- If the file contains 2 channels with channel ID's 'SLFT' and 'SRGT' the order must be: 'SLFT', 'SRGT'.
- If the file contains 5 channels with channel ID's 'MLFT' and 'MRGT' and 'C ' and 'LS ' and 'RS ' the order must be: 'MLFT', 'MRGT', 'C ', 'LS ', 'RS '.
- If the file contains 6 channels with channel ID's 'MLFT' and 'MRGT' and 'C ' and 'LFE ' and 'LS ' and 'RS ' the order must be: 'MLFT', 'MRGT', 'C ', 'LFE ', 'LS ', 'RS '.

The Channels Chunk is required and may appear only once in the Property Chunk.

3.2.3 Compression Type Chunk

The Compression Type Chunk defines the compression/decompression algorithm which is used for compressing sound data.

```
typedef struct {
      ID                  ckID;                  // 'CMPR'
      double ulong        CkDataSize;
      ID                  compressionType;       // compression ID code
      uchar               Count                  // length of the compression name
      text byte           compressionName[];     // human readable type name
} CompressionTypeChunk;
```

ckID is always 'CMPR'.

ckDataSize is the size of the data portion of the chunk, in bytes. It does not include the 12 bytes used by *ckID* and *ckDataSize*.

compressionType is used by applications to identify the compression algorithm, if any, used on the sound data.

count the length in bytes of *compressionName[]*.

compressionName[] can be used by applications to display a message when the needed decompression routine is not available.
Already defined compression types and names:

compressionType	***compressionName***	**Meaning**
'DSD '	"not compressed"	Uncompressed, plain DSD audio data
'DST '	"DST Encoded"	DST Encoded audio data

Other types may be defined in later versions.

The Compression Type Chunk is required and may appear only once in the Property Chunk.

3.2.4 Absolute Start Time Chunk

The Absolute Start Time Chunk defines the point on the time axis at which the sound data starts. The resolution for the Absolute Start Time is determined by the *sampleRate* which is defined in the Sample Rate Chunk.

```
typedef struct {
    ID              ckID;           // 'ABSS'
    double ulong    ckDataSize;
    ushort          hours;          // hours
    uchar           minutes;        // minutes
    uchar           seconds;        // seconds
    ulong           samples;        // samples
} AbsoluteStartTimeChunk;
```

ckID is always 'ABSS'.

ckDataSize is the size of the data portion of the chunk, in bytes. It does not include the 12 bytes used by *ckID* and *ckDataSize*.

hours defines the hours on the time axis. This value is within the range [0..23].

minutes defines the minutes on the time axis. This value is within the range [0..59].

seconds defines the seconds on the time axis. This value is within the range [0..59].

samples defines the samples on the time axis. This value is within the range [0..(*sampleRate*-1)]. *sampleRate* is defined in the Sample Rate Chunk.

If there is no Absolute Start Time Chunk, the start time is 00:00:00:00 [hh:mm:ss:samples].

The Absolute Start Time Chunk is optional but if used it may appear only once in the Property Chunk.

3.2.5 Loudspeaker Configuration Chunk

The Loudspeaker Configuration Chunk defines the set-up of the loudspeakers.

```
typedef struct {
    ID              ckID;           // 'LSCO'
    double ulong    ckDataSize;     // 2
    ushort          lsConfig;       // loudspeaker configuration
} LoudspeakerConfigurationChunk;
```

ckID is always 'LSCO'.

ckDataSize is the size of the data portion of the chunk, in bytes, which is always 2.

The loudspeaker configuration is defined as:

lsConfig	**Meaning**
0	2-channel stereo set-up
1..2	Reserved for future use
3	5-channel set-up according to ITU-R BS.775-1 [ITU]
4	6-channel set-up, 5-channel set-up according to ITU-R BS.775-1 [ITU], plus additional Low Frequency Enhancement (LFE) loudspeaker. Also known as "5.1 configuration"
5..65534	Reserved for future use
65535	Undefined channel set-up.

The Loudspeaker Configuration Chunk is optional but if used it may appear only once in the Property Chunk.

3.3 DSD SOUND DATA CHUNK

The DSD Sound Data Chunk contains the non-compressed sound data. The format for the data within a DSD Sound Data Chunk is shown below:

```
typedef struct {
      ID                ckID;                // 'DSD '
      double ulong      ckDataSize;
      uchar             DSDsoundData[];      // (interleaved) DSD data
} DSDSoundDataChunk;
```

ckID is the same ID used as *compressionType* in the Compression Type Chunk in the Property Chunk.

ckDataSize is the size of the data portion of the chunk in bytes. It does not include the 12 bytes used by ckID and ckDataSize. DSDsoundData[] contains the data that make up the sound.

DSDsoundData[] contains the data that make up the sound. If *DSDsoundData[]* contains an odd number of bytes, a pad byte is added at the end (but not used for playback).

DSD material consists of a sampled signal, where each sample is just one bit. Eight bits (samples) of a channel are clustered together in a Channel Byte (most significant bit is the oldest bit in time). For sound that consists of more than one channel, channel bytes are interleaved in the order as identified in the Channels Chunk (3.2.2). See also section 5.6 of [ScarletBook]. A set of interleaved channel bytes is called a Clustered Frame. This is illustrated below for the 2-channel case:

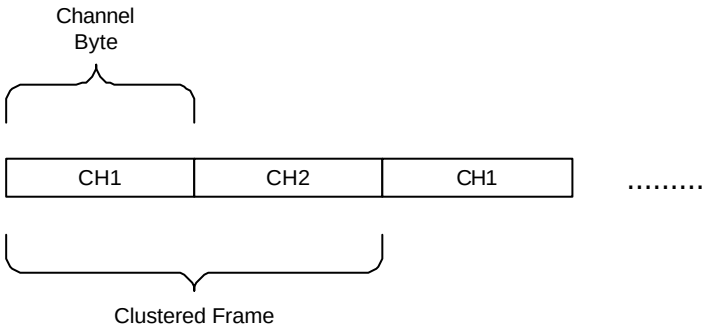

Figure 4: example of a DSD cluster Frame

Note that this implicates that the total number of bits for a channel in a Sound Data Chunk is a multiple of 8 bits. Furthermore there is a restriction that **all** clustered frames are *numChannels* bytes in length preserving the total number of bits per channel is equal for all channels.

Either the DSD or DST Sound Data (described below) chunk is required and may appear only once in the Form DSD Chunk. The chunk must be placed after the Property Chunk.

3.4 DST SOUND DATA CHUNK

The DST Sound Data Chunk contains the DST compressed sampled sound data. The DST Sound Data Chunk contains frames of DST encoded DSD data and the (optional) CRC over the original DSD data. The format for the data within a DST Sound Data Chunk is shown below:

```
typedef struct {
    ID              ckID;           // 'DST '
    double ulong    ckDataSize;
    Chunk           DstChunks[];    // container
} DSTSoundDataChunk;
```

ckID is the same ID used as *compressionType* in the Compression Type Chunk in the Property Chunk.

ckDataSize is the size of the data portion of the chunk in bytes. It does not include the 12 bytes used by *ckID* and *ckDataSize*.

DstChunks[] contain the local chunks.
The local chunks are:

- DST Frame Information Chunk
- DST Frame Data Chunk
- DST Frame CRC Chunk

The DST Frame Data Chunks must be stored in their natural, chronological order.

When the DST Frame CRC Chunks exist, the number o f DST Frame Data Chunks must be the same as the number of DST Frame CRC Chunks. The DST Frame Data Chunks and the DST Frame CRC Chunks will be interleaved, starting with the DST Frame Data Chunk. The CRC, stored in a DST Frame CRC Chunk, must be calculated over the original (non-compressed) DSD data corresponding to the audio of the preceding DST Frame Data Chunk.

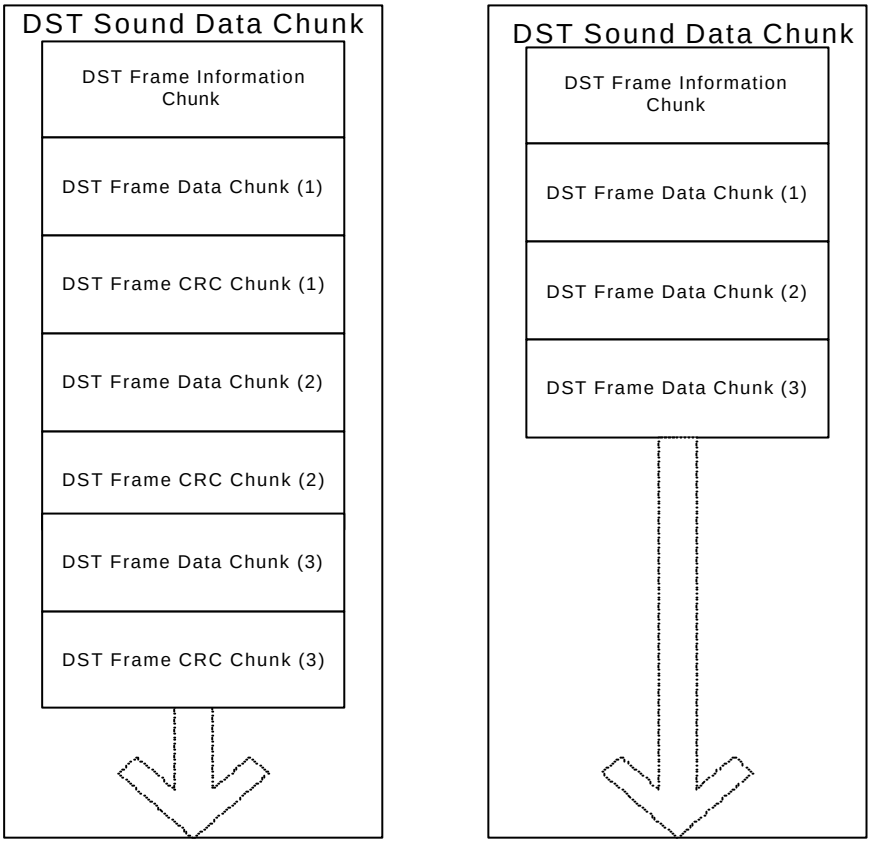

Figure 5: two possible examples of the DST Sound Data Chunks, one with interleaved CRC data and one without.

Either the DSD or DST Sound Data Chunk is required and may appear only once in the Form DSD Chunk. The chunk must be placed after the Property Chunk

3.4.1 DST Frame Information Chunk

The DST Frame Information Chunk contains the actual number of DST frames and the number of DST frames per second (DST frame rate). The format for the data within a DST Frame Information Chunk is shown below:

```
typedef struct {
    ID              ckID;           // 'FRTE'
    double ulong    ckDataSize;
    ulong           numFrames;      // number of DST frames.
    ushort          frameRate;      //  DST frame rate per second
} DSTFrameInformationChunk;
```

ckID is always 'FRTE'.

ckDataSize is the size of the data portion of the chunk in bytes. It does not include the 12 bytes used by *ckID* and *ckDataSize*.

numFrames is the number of DST frames (the number of chunks) in the file.

frameRate is the actual DST frame rate per second. The only valid value is 75.
The DST Frame Information Chunk is required if a DST Sound Data Chunk is used. The DST Frame Information Chunk must be the first chunk within the DST Sound Data Chunk. It may appear only once in a DST Sound Data Chunk.

3.4.2 DST Frame Data Chunk

The DST Frame Data Chunk contains the compressed sound data. The format for the data within a DST Frame Data Chunk is shown below:

```
typedef struct {
    ID              ckID;           // 'DSTF'
    double ulong    ckDataSize;
    uchar           DSTsoundData[]; // The DST data for one frame
} DSTFrameDataChunk;
```

ckID is always 'DSTF'.

ckDataSize is the size of the data portion of the chunk in bytes. It does not include the 12 bytes used by ckID and ckDataSize. DSTsoundData[] contains the data that make up the sound. If the DSTsoundData[] contains an odd number of bytes, a pad byte must be added at the end of the data to preserve an even length for this chunk. This pad byte is not included in ckDataSize and is not used for playback.

DSTsoundData[] contains the data that make up the sound. DST material consists of compressed DSD data (See [ScarletBook]).

The DST Frame Data Chunk is optional, and may appear more than once in the DST Sound Data Chunk.

3.4.3 DST Frame CRC Chunk

The DST Frame Data Chunk always precedes its corresponding DST Frame CRC Chunk. The format for the CRC data within a DST Frame CRC Chunk is shown below:

```
typedef struct {
    ID              ckID;           // 'DSTC'
    double ulong    ckDataSize;
    uchar           crcData[];      // the value of the CRC
} DSTFrameCrcChunk;
```

ckID is always 'DSTC'.

ckDataSize is the size of the data portion of the chunk in bytes. It does not include the 12 bytes used by ckID and ckDataSize. If the *crcData[]* contains an odd number of bytes, a pad byte is added at the end of the data to preserve an even length for this chunk. This pad byte is not included in ckDataSize.

crcData[] contains the data that make up the CRC value over the original (interleaved) DSD data of the preceding DST Frame Data Chunk.

The specification of the CRC Algorithm :
A four-byte CRC is computed over the *DSDsoundData[]*, per frame.
The First Byte of the *crcData* contains bit 31 till 23, ... etc.

The DSD data, consisting of interleaved channel bytes, is considered as a sequence/stream *I(x)* of single bit fields:
- starting with the most significant bit of the first byte, referred to as bit number (*numChannels * sampleRate / frameRate*) - 1
- ending with the least significant bit of the last byte, referred to as b_0.

The CRC, consisting of bits c_{31} through c_0, is defined as:

$$CRC(x) = \sum_{i=31}^{0} c_i x^i = (x^{32} \cdot I(x)) \bmod 2\text{long} \quad G(x)$$

where :

$$I(x) = \sum_{i=(sampleRate/FrameRate)*numChannels-1}^{0} b_i x^i$$

$$G(x) = x^{32} + x^{31} + x^4 + 1$$

Note :

x = 2

mod2long = long division modulo 2, modulo 2 means no "borrowing"

The algorithm to calculate the CRC for each frame is :
1. Append 32 zero bits to the least significant side of the bitstream *I(x)*.
2. Divide *G(x)* into x^{32} *I(x)* by long division modulo 2.
3. The remainder, a polynomial of degree 31 or less, is the CRC.

The DST Frame CRC Chunk is optional and if used exactly one chunk must be placed after each DST Frame Data Chunk.

3.5 DST SOUND INDEX CHUNK

The DST Sound Index Chunk contains indexes to the DST Frame Data Chunks. The format for the data within a DST Sound Index Chunk is shown below:

```
typedef struct {
    ID                  ckID;               // 'DSTI'
    double ulong        ckDataSize;
    DSTFrameIndex       indexData[];        // array of index structs
} DSTSoundIndexChunk;
```

ckID is always 'DSTI'.

ckDataSize is the size of the data portion of the DST Sound Index Chunk in bytes. It does not include the 12 bytes used by *ckID* and *ckDataSize*. If the *indexData[]* contains an odd number of bytes, a pad byte is added at the end to preserve an even length for this chunk. This pad byte is **not** included in *ckDataSize*.

indexData[] contains the file positions of the DST Frame Data Chunks and their lengths. It is an array of DSTFrameIndex structs.

```
typedef struct {
    double ulong    offset;   // offset in the file [in bytes] of the sound in the DST Sound Data Chunk
    ulong           length;   // length of the sound in bytes
} DSTFrameIndex;
```

The DST Sound Index Chunk is recommended, and may appear only when the compression type is 'DST '. Only one DST Sound Index Chunk is allowed in a Form DSD Chunk.

3.6 COMMENTS CHUNK

The Comments Chunk is used to store comments in DSDIFF. The format for the data within a Comments Chunk is shown below:

```
typedef struct {
    ID              ckID;           // 'COMT'
    double ulong    ckDataSize;
    ushort          numComments;    // number of comments
    Comment         comments[];     // the concatenated comments
} CommentsChunk;
```

ckID is always 'COMT'.

ckDataSize is the size of the data portion of the chunk, in bytes. It does not include the 12 bytes used by *ckID* and *ckDataSize*.

numComments contains the number of comments in the Comments Chunk.

comments[] are the comments themselves. Each Comment consists of an even number of bytes, so no pad bytes are needed within the comment chunks.

The format for describing each of the comments is shown below:

```
typedef struct{
    ushort      timeStampYear;      // creation year
    uchar       TimeStampMonth;     // creation month
    uchar       timeStampDay;       // creation day
    uchar       timeStampHour;      // creation hour
    uchar       timeStampMinutes;   // creation minutes
    ushort      cmtType;            // comment type
    ushort      cmtRef;             // comment reference
    ulong       count;              // string length
    text byte   commentText[];      // text
} Comment;
```

timeStampYear indicates the year of the comment creation. Values are in the range [0..65535].

timeStampMonth indicates the month of the comment creation. Values are in the range [0..12].

timeStampDay indicates the day of the comment creation. Values are in the range [0..31].

timeStampHour indicates the hour of the comment creation. Values are in the range [0..23].

timeStampMinutes indicates the minutes of the comment creation. Values are in the range [0..59].

Applications and or machines without a real time clock must use a time stamp according to yyyy-00-00 00:00, where yyyy is in the range of [0000-1999]. This will allow 2000 comments to be created. The numbering of the comment should start at 0000. Each next comment should increase the year number by one.

cmtType indicates the comment type. The following types are defined:

cmtType	**Meaning**
0	General (album) Comment
1	Channel Comment
2	Sound Source
3	File History
4..65535	Reserved

cmtRef is the comment reference and indicates to which the comment refers.

If the comment type is *General Comment* the comment reference must be 0.

If the comment type is *Channel Comment*, the comment reference defines the channel number to which the comment belongs.

cmtRef	**Meaning**
0	all channels
1	first channel in the file
2	second channel in the file
..	..
numChannels	last channel of the file

The value of *cmtRef* is limited to [0...*numChannels*].

If the comment type is *Sound Source* the comment reference is defined as:

cmtRef	**Meaning**
0	DSD recording
1	Analogue recording
2	PCM recording
3.. 65535	Reserved

The *Sound Source* indicates the original storage format used during the live recording.

If the sound type is *PCM recording* then it is recommended to describe (textually) the bit length and the sample frequency in the *commentText[]*.

If the comment type is *File History* the comment reference can be one of:

cmtRef	**Meaning**
0	General Remark
1	Name of the operator
2	Name or type of the creating machine
3	Time zone information
4	Revision of the file
5..65535	Reserved for future use

The format for *File History-Place or Zone* is "(GMT ±hh:mm)", if desired followed by a textual description. A space (0x20) is used after GMT. When the *File History-Place or Zone* is omitted the indicating time is Greenwich Mean Time "(GMT +00:00)".

The format for *File History-Revision* is "(N)", where N is the revision number starting with 1 for the first revision.

count is the length of the *commentText[]* that makes up the comment.

commentText[] is the description of the Comment. This text must be padded with a byte at the end, if needed, to make it an even number of bytes long. This pad byte, if present, is not included in *count*.

Remarks:
- The history of the file content can be tracked from the timestamp of each comment.
- When a time stamp with a *timeStampYear* earlier than 2000 occurs, the order of the comments in the file designates the sequence of changes described by the comments.
- File history is useful for inquiry if a problem has occurred.
- Comments describing the same action are linked together by means of equal time stamps.

The Comment Chunk is optional but if used it may appear only once in the Form DSD Chunk.

3.7 EDITED MASTER INFORMATION CHUNK

The Edited Master Information Chunk is a container chunk for storing edited master information. The format for the data within an Edited Master Information Chunk is shown below:

```
typedef struct {
    ID                  ckID;               // 'DIIN'
    double ulong        ckDataSize;
    Chunk               EmChunks[];         // container
} EditedMasterInformationChunk;
```

ckID is always 'DIIN'.

ckDataSize is the size of the data portion of the chunk, in bytes. It does not include the 12 bytes used by *ckID* and *ckDataSize*. It is the total length of the local chunks.

EmChunks[] contain local chunks.
The local chunks are:
- Edited Master ID Chunk
- Marker Chunk
- Artist Chunk
- Title Chunk

The Edited Master Information Chunk is optional but if used it may appear only once in the Form DSD Chunk.

3.7.1 Edited Master ID Chunk

The Edited Master ID Chunk stores an identifier.
The format for the data within Edited Master ID Chunk is shown below:

```
typedef struct {
    ID                  ckID;               // 'EMID'
    double ulong        ckDataSize;
    text byte           emid[];             // unique sequence of bytes
} EditedMasterIDChunk;
```

ckID is always 'EMID'.

ckDataSize is the size of the data portion of the chunk, in bytes. It does not include the 12 bytes used by *ckID* and *ckDataSize*.

emid[] contains the identifier. The length and contents of the Edited Master identifier are not specified, they are application specific. It is recommended that the *emid[]* is unique for each Edited Master file. Therefore it is recommended to use date, time, machine name, serial number, and so on, for an *emid[]*.

The Edited Master ID Chunk is optional but if used it may appear only once in the Edited Master Information Chunk.

3.7.2 Marker Chunk

The Marker Chunk defines a marker within the sound data. It defines a type of marker, the position within the sound data and a description. The format for the data within a Marker Chunk is shown below:

```
typedef struct {
    ID              ckID;           // 'MARK'
    double ulong    ckDataSize;
    ushort          hours;          // marker position in hours
    uchar           minutes;        // marker position in minutes
    uchar           seconds;        // marker position in seconds
    ulong           samples;        // marker position  in samples
    Long            offset;         // marker offset in samples
    ushort          markType;       // type of marker
    ushort          markChannel;    // channel reference
    ushort          TrackFlags;     // special purpose flags
    ulong           count;          // string length
    Text byte       markerText[];   // description
} MarkerChunk;
```

ckID is always 'MARK'.

ckDataSize is the size of the data portion of the chunk, in bytes. It does not include the 12 bytes used by *ckID* and *ckDataSize*.

hours defines the hours on the time axis. This value is within the range [0..23].

minutes defines the minutes on the time axis. This value is within the range [0 ..59].

seconds defines the seconds on the time axis. This value is within the range [0..59].

samples defines the samples on the time axis. This value is within the range [0 .. (*sampleRate*-1)]. The *sampleRate* is defined in the Sample Rate Chunk.

offset defines the offset of the marker in *samples* in range of $[-2^{31},2^{31}-1]$ with respect to marker time. The offset can be used for modifying a marker position, without changing the original position.

The position of the marker is determined by *hours*, *minutes*, *seconds* and *samples* plus *offset*.

Marker times must be regarded as absolute times.

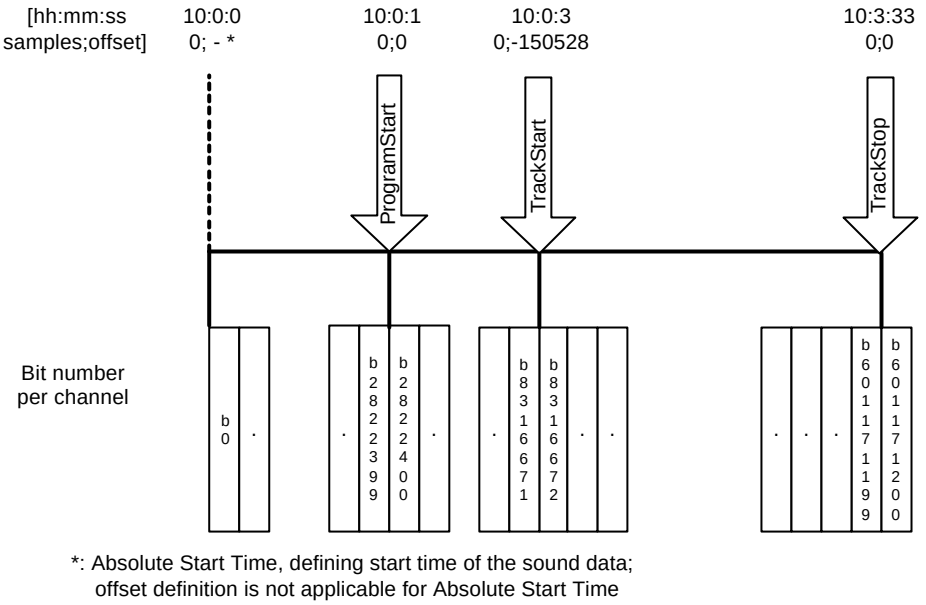

Figure 6: relation between (marker) times and samples in DSDIFF files.

markType defines the type of marker. Currently the following types have been defined:

markType	**Name**	**Meaning**
0	*TrackStart*	Entry point for a *Track* start
1	*TrackStop*	Entry point for ending a *Track*
2	*ProgramStart*	Start point of 2-channel or multi-channel area
3		Obsolete
4	*Index*	Entry point of an *Index*
5..65535		Reserved for future use

TrackStop of the last *Track* is also called *ProgramEnd*.

markChannel defines the channel to which this marker belongs.

markChannel	**Meaning**
0	All channels
1	First channel in the file
2	Second channel in the file
..	..
numChannels	Last channel of the file

The value of *markChannel* is limited to [0...*numChannels*].

TrackFlags define a series of flags to be used with a marker. The behaviour of the *TrackFlags* is determined by the number and order of the channels defined in the file. Bit 0 is the least significant bit of *TrackFlags*.

Restrictions on the *TrackFlags*:

- *TrackFlags* is only meaningful when *markType* is zero (*TrackStart*); if *markType* is not zero *TrackFlags* must be zero
- *TrackFlags* is only meaningful when *markChannel* is zero (All channels)

- If a *TrackFlag* is set, the corresponding channel(s) must contain a Silence Pattern [ScarletBook] during the whole *Track*, with two exceptions:
  1. it is not required to have a Silence Pattern at the start of the *Track* only if the channel is available (i.e. the same *TrackFlag* is not set) within the previous *Track*
  2. it is not required to have a Silence Pattern at the end of the *Track* only if the channel is available (i.e. the same *TrackFlag* is not set) within the next *Track*

Flags have been defined for files
- with *numChannels* = 2 using the channel ID's : 'SLFT', 'SRGT'. or
- with *numChannels* = 5 using the channel ID's : 'MLFT', 'MRGT', 'C ', 'LS ', 'RS '. or
- with *numChannels* = 6 using the channel ID's : 'MLFT', 'MRGT', 'C ', 'LFE ', 'LS ', 'RS '

Currently the following flags have been defined

TrackFlag	**Name**	**Meaning**
Bit 0	TMF4_Mute	Indicates whether the 4th channel of this *Track* is muted
Bit 1	TMF1_Mute	Indicates whether the 1st and 2nd channel of this *Track* are muted
Bit 2	TMF2_Mute	Indicates whether: a) *numChannels* = 5, the 4th and 5th channel of this *Track* are muted b) *numChannels* = 6, the 5th and 6th channel of this *Track* are muted
Bit 3	TMF3_Mute	Indicates whether the 3rd channel of this *Track* is muted
Bit 4..7		Reserved
Bit 8..10	Extra_Use	Indicates whether the 4th channel is used for an LFE loudspeaker
Bit11..15		Reserved for future expansion

Restrictions on the *TMFn_Mute* flags (n=1..4):
- Value one indicates muting, value zero indicates no-muting.
- 1st and 2nd channel are always available (TMF1_Mute is equal to zero)
- For *numChannels* = 2, TMF1_Mute, TMF2_Mute, TMF3_Mute and TMF4_Mute must be set to zero.
- For *numChannels* = 5, TMF4_Mute must be set to zero.
- For *numChannels* = 5, minimal one of TMF1_Mute, TMF2_Mute, TMF3_Mute must be set to zero.
- For *numChannels* = 5 or 6, minimal three channels are available.
- For *numChannels* = 6, minimal one of TMF1_Mute, TMF2_Mute, TMF3_Mute, TMF4_Mute must be set to zero.
- For *numChannels* = 6 and TMF4_Mute is equal to zero, minimal four channels are available.

Restrictions on the *Extra_Use* Flags:
- For *numChannels* = 5, the three bit values must be zero, no LFE loudspeaker available.
- For *numChannel* = 6, the three bit values must be zero (Implicit usage of the LFE loudspeaker).

Note: The previously defined LFE_mute flag is still compatible, because the LFE channel is the 4th channel of a 5.1 multi-channel file.

count is the length of the markerText that makes up the description of the marker.

markerText[] is the description of the marker. This text must be padded with a byte at the end, if needed, to make it an even number of bytes long. This pad byte, if present, is not included in *count*.

Creating Tracks from Markers.

A *Track* is defined as a [*TrackStart*,<*TrackStart*|*TrackStop* >].
The last *Track* of the file is ended by means of a *TrackStop* marker.
Time between *Tracks* is denoted as *Pause*.
Time between *ProgramStart* and the first *TrackStart* is denoted as *Pause[1]*. An *Index Marker* belongs to the *Track* when the timestamp of the *Index* lies between the timestamps of the start and the end of the *Track*.

Time between the last *TrackStop* and the end of the data is denoted as *Post-roll*.

Definition	Meaning
Pause[1]	Starts from *ProgramStart* and ends at the first occurrence of *TrackStart*. *Pause[1]* is optional, recommended length is 2 seconds.
Pause[2..N]	Starts from a *TrackStop* and ends at the following *TrackStart* *Pause[2..N]* is optional.
Track	Starts from a *TrackStart* and ends at the first following : [*TrackStop*	*TrackStart*]
Program	[<*Track*>|<*Pause*><*Track*>]+ Starts Always with a *ProgramStart* Marker
Post-roll	Starts from last *TrackStop* and ends at the end of the data

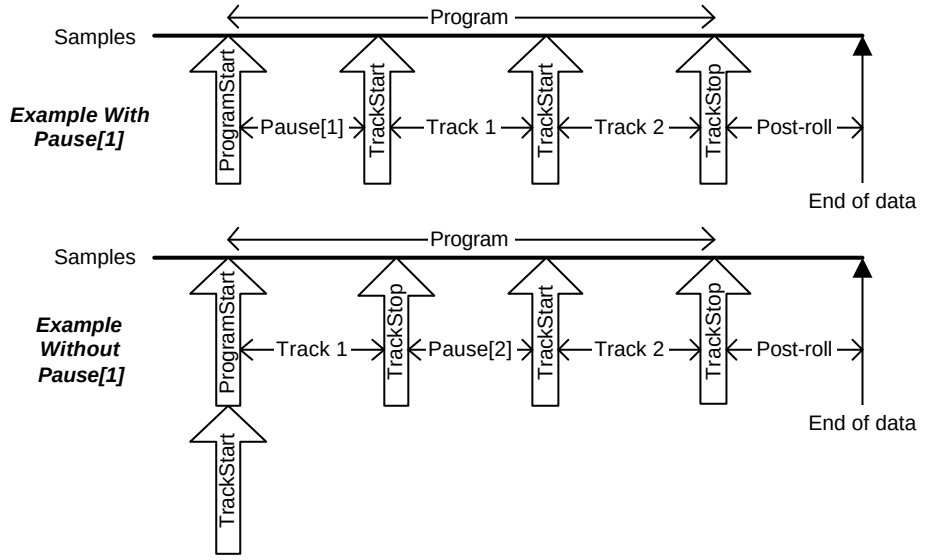

Figure 7: examples of the conversion from DSDIFF markers to Tracks.

The Marker Chunk is optional and if used it may appear more than once in the Edited Master Information Chunk.

3.7.3 Artist Chunk

The Artist Chunk defines the name of the Artist. The format for the data within an Artist Chunk is shown below:

```
typedef struct {
    ID              ckID;           // 'DIAR'
    double ulong    ckDataSize;
    ulong           count;          // string length
    text byte       artistText[];   // description
} ArtistChunk;
```

ckID is always 'DIAR'.

ckDataSize is the size of the data portion of the chunk, in bytes. It does not include the 12 bytes used by *ckID* and *ckDataSize*.

count is the length of the *artistText[]* that makes up the name of the artist.

artistText[] is the name of the Artist. This text must be padded with a byte at the end, if needed, to make it an even number of bytes long. This pad byte, if present, is not included in *count*.

The Artist Chunk is optional, but if it exists it may appear only once in the Edited Master Information Chunk.

3.7.4 Title Chunk

The Title Chunk defines the title of the project in the file. The format for the data within a Title Chunk is shown below:

```
typedef struct {
    ID              ckID;           // 'DITI'
    double ulong    ckDataSize;
    ulong           count;          // string length
    text byte       titleText[];    // description
} TitleChunk;
```

ckID is always 'DITI'.

ckDataSize is the size of the data portion of the chunk, in bytes. It does not include the 12 bytes used by *ckID* and *ckDataSize*.

count is the length of the *titleText[]* that makes up the working title of the project.

titleText[] is the name of the project in the file. This text must be padded with a byte at the end, if needed, to make it an even number of bytes long. This pad byte, if present, is not included in *count*.

The Title Chunk is optional, but if it exists it may appear only once in Edited Master Information Chunk.

3.8 MANUFACTURER SPECIFIC CHUNK

The Manufacturer Specific Chunk is a chunk for storing manufacturer specific information. The form for the data within a Manufacturer Specific Chunk is shown below:

```
typedef struct {
    ID              ckID;           // 'MANF'
    double ulong    ckDataSize;
    ID              manID;          // unique manufacturer ID [4 characters]
    uchar           manData[];      // manufacturer specific data
} ManufacturerSpecificChunk;
```

ckID is always 'MANF'.

ckDataSize is the size of the data portion of the chunk, in bytes. It does not include the 12 bytes used by *ckID* and *ckDataSize*. It is the total length of manID and manData[].

manID contains the manufacturer identifier which must contain a unique ID.
A manufacturer who wants to use this chunk must request a unique manID from the administrator of DSDIFF.

manData[] contains the manufacturer specific data. If *manData[]* contains an odd number of bytes, a pad byte must be added at the end. The pad byte is not included in *ckDataSize*.

It is recommended to maintain the chunk structure within the *manData[]*- field of the Manufacturer Specific Chunk according to the common chunk structure, as defined in 2.3 ("File structure").

The Manufacturer Specific Chunk is optional, but if used it may appear only once in the Form DSD Chunk and it must be placed behind the Sound Data Chunk. A chunk with an unrecognized *manID* must be handled as defined in section 2.4 ("Handling of unrecognized chunks").

4 Edited Master

4.1 INTRODUCTION

This chapter defines requirements for the data in an Edited Master.
An Edited Master is a classification of a DSDIFF file. It contains data which will be used for creating a disc (image) [ScarletBook].
A disc can consist of 1 or 2 areas. The following combinations of areas on a disc are allowed:

- 2-channel area
- 5-channel area
- 6-channel area
- 2-channel and 5-channel area
- 2-channel and 6-channel area

An Edited Master contains the data for one area, therefore a disc will be created from the following combinations of Edited Masters:

- 2-channel Edited Master
- 5-channel Edited Master
- 6-channel Edited Master.
- 2-channel Edited Master and 5-channel Edited Master
- 2-channel Edited Master and 6-channel Edited Master

Because the content of an Edited Master is used for disc creation, the restrictions on Edited Masters are derived from the disc specification [ScarletBook]. Furthermore identification of Edited Master files is needed, especially to indicate that two Edited Masters are belonging to one disc. An Edited Master can be either a DSD or DST file, because DSD and DST files can be translated into each other.

4.2 REQUIRED CHUNKS IN AN EDITED MASTER

In an Edited Master the following chunks must be available:

- Form DSD Chunk
  - Format Version Chunk
  - Property Chunk
    - Sample Rate Chunk
    - Channels Chunk
    - Compression Type Chunk
    - Absolute Start Time Chunk
    - Loudspeaker Configuration Chunk
  - DSD Sound Data Chunk or DST Sound Data Chunk
  - Edited Master Information Chunk
    - Edited Master ID Chunk
    - Marker Chunks

If the DST Sound Data Chunk is available then the following chunks must be available :
- DST Frame Information Chunk
- DST Frame Data Chunks

4.3 RESTRICTIONS ON AN EDITED MASTER

The following restrictions apply to an Edited Master :
- The DSD or DST sound data must have a *sampleRate* of 2822400 Hz.
- If the sound data is DST, the DST *frameRate* must be 75 Hz.
- The Absolute Start Time must have a value denoting a *Super Audio CD Frame* boundary; the following condition must be met:
  - samples mod 37632 = 0.
- The Marker Chunks are required and must represent a *Program.*
- All markers must be stored in the file in ascending order of time stamp.
- All markers must be placed at a *Super Audio CD Frame* boundary; the following conditions must be met:
  - samples mod 37632 = 0;
  - offset mod 37632 = 0.
- The duration of each *Track* must be at least 1 second.
- The maximum number of *Tracks* in an Edited Master is 255.
- The duration of each *Pause* must be at least 0 *Super Audio CD Frames*.
- All *Index* markers must lay inside a *Track*.
- Not more than 254 *Index Markers* are allowed within one *Track*, resulting in 255 *Indexes*; the first *Index* within each *Track* corresponds to *TrackStart*.
- The maximum number of *Index Markers* within one Edited Master file depends on the number of *Tracks* [ScarletBook]. The total number of *Indexes* (within a single Edited Master) is limited to approximately 6000.
- All markers must have a unique time stamp, except for the *ProgramStart* Marker. The *ProgramStart* Marker may coincide with the first *TrackStart* marker.
- The field *markChannel* must be zero for each marker (indicating that the marker applies to all channels).
- The maximum duration of the sound between *ProgramStart* and *ProgramEnd* is limited to 255:59:74 in minutes, seconds and *Super Audio CD Frames*. The total size of the *Program(s)* on a Super Audio CD is bounded by the data capacity of the intended disc.

Specific restrictions for a 2-channel Edited Master:
- *lsConfig* must have the value 0.
- *numChannels* must have the value 2.
- *chID[]* must have the values :
  'SLFT','SRGT'.

Specific restrictions for a 5-channel Edited Master:
- *lsConfig* must have the value 3.
- *numChannels* must have the value 5.
- *chID[]* must have the values :
  'MLFT','MRGT','C ','LS ','RS '.

Specific restrictions for a 6-channel Edited Master:
- *lsConfig* must have the value 4.
- *numChannels* must have the value 6.
- *chID[]* must have the values :
  'MLFT','MRGT','C ','LFE ','LS ','RS '.

LsConfig can be found in the Loudspeaker Configuration Chunk and *numChannels, chID[]* can be found in the Channels Chunk.

4.4 RECOMMENDATIONS FOR AN EDITED MASTER

The following recommendations apply to an Edited Master:

- The Artist and Title chunks should be used to enhance the identification of Edited Masters.
- The Comments chunk should be used to keep track of the history of the file contents.
- It is recommended that both *Pause[1]* and the *Post-roll* have a length of 2 seconds.

4.5 INTERPRETATION OF THE MARKERS

The rules to apply for interpretation of the markers:

- All DSD or DST sound data in the Edited Master before the marker *ProgramStart* is not part of the area.
- The *Post-roll*, i.e. all DSD or DST sound data in the Edited Master behind the end of the last *TrackStop* marker (*ProgramEnd*), is not part of the area.
- If one or more *Index Markers* exist within a *Track*, the *TrackStart* marker is also the first *Index Marker*.

4.6 IDENTIFICATION OF AN EDITED MASTER

The following information is only used for identification of an Edited Master, but is not placed on the disc:

- The Edited Master ID in the Edited Master ID Chunk, *emid[]*.
- The name of the Artist in the Artist Chunk, *artistText[]*.
- The title in the Title Chunk, *titleText[]*.
- Comments, especially the *File History*.