Extensions and Applications of the SDIF Sound Description Interchange Format

Diemo Schwarz (schwarz@ircam.fr)
IRCAM – CENTRE GEORGES POMPIDOU

Matthew Wright (matt@cnmat.CNMAT.Berkeley.EDU)
CNMAT – UNIVERSITY OF CALIFORNIA, BERKELEY

ABSTRACT

This paper concentrates on recent extensions and applications of the well established SDIF Sound Description Interchange Format: **SDIF selection** is a standard way to access a part of an SDIF file. It allows users to select SDIF frames of a certain type and time, and certain parts of the matrix data. It can be appended to a file name, making it easy to use for command-line programs. Other applications include choosing output description types, or mapping them to display modes or actions, e.g., when converting from untyped legacy formats. Foreign **programming language interfaces** allow processing of SDIF data by languages other than C. Interfaces and support functions for Lisp (CLOS) and *Matlab* are presented. **Dispatching file I/O** is a new programming style to be supported by IRCAM's SDIF library. Users of the library specify a set of callback functions each with an SDIF selection indicating the frames and matrices that should be passed to that callback. The library takes care of reading the file, and dispatches selected SDIF data to the appropriate callback(s). Writing works analogously. The **applications** described in this paper include utility programs for Unix and Macintosh for conversion, viewing, extracting, and merging of SDIF, the integration of SDIF into the widely used software-systems *Additive*, *jMax*, *Max/MSP*, *Diphone*, *OpenMusic*, and the synergetic effects induced. We propose an **SDIF Frame Directory**, which allows rapid random access to data at a given time in an SDIF file by recording the file position, type, time, and other information about each data frame in an SDIF file.

1 INTRODUCTION

Seeing that SDIF, the open, portable, well-defined, extensible Sound Description Interchange Format (Wright, Chaudhary, Freed, Khoury & Wessel, 1999), is now well established and accepted within the computer music community, this paper concentrates on recent SDIF extensions and applications. Most of these are implemented in IRCAM's and CNMAT's free SDIF libraries for Unix, Windows, and Macintosh that can be downloaded from `http://www.ircam.fr/sdif` and `http://www.cnmat.berkeley.edu/SDIF`.

The following sections are ordered to minimize forward references: Section 2 presents the SDIF selection, section 3 programming interfaces to the SDIF library, and section 4 new applications of SDIF. They are followed by a proposal for an SDIF frame directory in section 5, and conclusions in section 6.

2 SDIF SELECTION

SDIF files can contain aggregates of different sound descriptions for one or multiple sound objects, hence the need to address a subset of these. The SDIF selection defines a syntax convention to specify a part of the data of an SDIF file. For example, certain time ranges, certain streams, certain frame or matrix types, or certain rows or columns of the SDIF matrices can be selected. This selection specification can be conveniently added to a file name, so that command-line programs transparently inherit powerful selection capabilities. They only have to use a selection-aware SDIF library, such as IRCAM's, which takes care of parsing and executing the selection. There are more applications for the SDIF selection syntax: It can choose the output description type of programs, or can map them to display modes or actions.

2.1 Selection Syntax

The syntax for a filename, possibly including the directory path, with an SDIF selection is:

[*filename*]`::`[`#`*stream*][`:`*frame*][`/`*matrix*][`.`*column*][`_`*row*][`@`*time*]

The start of the selection specification is marked by the last '`::`' occurring, followed by 6 optional selection elements. This way, there is no ambiguity with filenames containing the selection element markers.[1] The order of the selection elements is not significant, and the selection specification can contain white space All element specifications can be comma-separated lists of values. Numerical specifications can also be ranges *lower-upper*, or delta ranges *value+delta*, selecting the range from *value−delta* to *value+delta*. The element markers have been chosen to minimize clashes with Unix shell special characters. Their mnemonics and meanings are:

- `#`*stream-id* as with "number" or `$`*stream-id*[2] as with "stream" selects all streams with the given numbers or names.
- `:`*frame*`/`*matrix*, as in a file hierarchy, selects the data with matching signatures. If only the frame element is present, all matrices are selected. If only the matrix element is given, all matrices of that type are selected, independent of the frames they occur in. This allows, for instance, to view all comments in `1CMT` matrices in a file.
- `.`*column* as in a C-struct and `_`*row* as in a LaTeX index select a sub-matrix in the selected matrices.
  Beware that column, and sometimes row and matrix selections can produce invalid SDIF output lacking required columns for a given matrix type or required matrices for a given frame type. However, it is very useful to be able to do this for external analysis of the data in an SDIF file. SDIF tools should check if it is allowed to select columns or rows. If it is, the order of the column and row selection is significant, allowing re-ordering.
  Columns can be given either as a number or as a name. The column names are given by the matrix type definition[3].
- `@`*time*, as in English "at time t", selects the frames in the given time range.

For example, to specify the part of the SDIF file piano.sdif which is contained in stream number 1 in `1HRM` frames and matrices, and to select only columns 3 and 2 (amplitude and frequency) of rows 1 through 50 (the first fifty partials appearing in each matrix), between the times 1.999 and 2.001, you can say:

```
piano.sdif :: #1 :1HRM /1HRM .3,2 _1-50 @2+0.001
```

As a shortcut, if the first selection element is the frame, you can drop the frame element marker '`:`'. So, instead of *filename*`:::`*frame*`@`*cetera* it is *filename*`::`*frame*`@`*cetera*.

2.2 Applications

The SDIF selection defines a syntax and a standard semantics for programs reading SDIF files: Specifying a file name

1. To specify a filename containing '`::`' itself, simply append '`::`', which means an empty selection matching all data.
2. The dollar sign can be used in URLs instead of `#`, which has a special meaning there. This is to allow selections with an SDIF-aware web server.
3. For the moment, types are defined in IRCAM's `1TYP` frames, later in an XML-based language.

plus a selection is exactly equivalent to removing all but the selection from the SDIF file and then specifying that new file (which can be empty). Implementing that semantics is made very easy in the IRCAM SDIF library by high level functions: Opening a file automatically parses the selection specification, the `SdifFReadNextSelectedFrameHeader`[4] and `SdifFCurrMatrixIsSelected` functions find selected frames and matrices for you, and there are functions which perform the row and column mapping given by the selection.

All IRCAM SDIF tools accept an SDIF selection for **reading**. This works also with standard input, using as filename `::`*select-spec* or `-::`*select-spec*.

For **writing**, the selection can be used to specify the output description type. For example, in IRCAM's converter programs from untyped ASCII data to SDIF, you can choose `1HRM` or `1TRC` output for partial data, or the output type of the ubiquitous two-column break-point function files:

```
bpftosdif glass.f0 glass.sdif::1FQ0/1FQ0
```

As these programs also read SDIF, we immediately have an SDIF type converter (use at your own risk).

The selection can also choose the **display mode** of description types. In the *Matlab* application *ViewEnv* (Schwarz, 1998), for instance, the x- and y-axis for frame-wise plotting are given by the column selection. Thus, `mongol.sdif::/1HRM,1TRC.2,3` as filename would plot partial amplitude over frequency.

The SDIF selection can also be used for **mapping actions** to description types, e.g.: "Edit this frame type with that program".

3 PROGRAMMING INTERFACES

This section treats programming language interfaces (APIs) for foreign languages and for C, that allow using the functionality of the SDIF library by applications. The advantages of using the library via one of these interfaces, vs. rewriting the functionality are evident: The programmer profits immediately from all the capabilities of the library, some of which might not be possible at all with a foreign language, and from all future enhancements of the library. Interfaces to other languages, such as Perl or Java, are being considered.

3.1 Matlab

The SDIF interface for the mathematical modelling and programming language *Matlab* is realized using its "*mex*" extension interface, based on dynamic link libraries. Its present version allows to read and write SDIF files frame-wise. It is particularly straightforward to use, because the fundamental object of *Matlab* is the matrix of numbers, which nicely matches the base concept of SDIF. The mex binaries can be downloaded from `http://www.ircam.fr/sdif`.

For **reading**, the `loadsdif` mex-function is called with a filename argument to open an SDIF file, returning header data (`1NVT` name–value tables). An optional argument can specify a types file with private description type definitions. Subsequent calls read and return one matrix a time, along with the frame time, stream ID, and frame and matrix signatures. An empty matrix signals end-of-file. A further call with the argument `'close'` closes the file. This function is aware of the SDIF selection appended to the filename, returning only the selected data.

For **writing**, the `writesdif` mex-function opens a file when called with a filename argument. Again, a private types file can be specified. Frames with any number of matrices can be written by one call with the arguments *stream ID, frame time, frame signature* and one pair *matrix signature, matrix data* per matrix. A further call with the argument `'close'` closes the file[5].

Some **support functions** have been written in the *Matlab* programming language itself to further simplify using SDIF:

- To check the existence of an SDIF file, and whether its file header is correct, you can use the function `sdifexist`.
- To load an entire SDIF file at once, preserving its frame/matrix structure, you can use `loadsdiffile`. It returns a cell array of matrices, and arrays of frame time, stream, and signature.
- For even more straightforward access, `loadsdifflat` loads all selected data from an SDIF file into one *Matlab* matrix, concatenating the SDIF matrices vertically, the frame time being inserted as the first column. Optionally, it also returns arrays for the stream ID, frame and matrix signatures, and indices that link frames to their matrices. For this to be possible, the user has to make sure that all selected matrices have the same number of columns, which can always be achieved using a column selection. For example

  ```
  m = loadsdifflat('piano.sdif::/1HRM.2,3_1')
  ```

  loads the first partial of any additive analysis data found in file `piano.sdif` into the matrix `m`. Its columns are: frame time, partial frequency, partial amplitude. Similarly,

  ```
  f = loadsdifflat('f0.sdif::1FQ0/1FQ0.1')
  plot(f(:,1), f(:,2))
  ```

  loads and plots the fundamental frequency break-point function over time from file `f0.sdif`.
- `writesdiffile` and `writesdifflat` are the counterparts of the respective load functions. They write a complete SDIF file, using the same representation of the data to be written as is returned by `loadsdiffile` and `loadsdifflat`. For your convenience, the stream ID and signature parameters can be scalars if they stay the same throughout the file.

3.2 Lisp

The functions of IRCAM's SDIF library have been made directly callable from the Lisp dialect CLOS (*Common Lisp Object System*) on Macintosh by Carlos Agon. This forms the basis of *OpenMusic*, described in section 4.3. The functions allow reading of frame header and matrix data, searching for data according to the selection, accessing and setting file header data needed by certain IRCAM applications[6], and writing frames and matrices.

CLOS allows calling the functions of a dynamic C-library directly. It can also access all scalar C-types, including pointers (it cannot dereference them, though). Because the IRCAM SDIF library follows an object-oriented design, all manipulations are carried out on the library's basic objects (SDIF file, frame and matrix headers, selection, list, hash) by calling access functions with the object pointers.

3.3 Dispatching File I/O

For more efficient usage of the library, and better shielding of the programmer from the details of reading and writing SDIF, a callback-oriented API will be made available as part of the IRCAM SDIF library. It supports an alternative programming style for reading and writing by using a "read/write manager". Users of the library specify callback functions associated with one SDIF selection each. The library takes care of reading/writing one or more files, and dispatches selected SDIF data to the appropriate callback(s). Writing works analogously: each callback is called in turn to provide data to be written to one or possibly several files by the library. The callbacks can be from independent modules of a system, each handling different parts of SDIF files.

4 APPLICATIONS

4.1 Utilities

These utilities for Unix and Macintosh are part of the IRCAM SDIF library:

4. This function will later make use of the SDIF frame, directory described in section 5, for direct access to selected frames.
5. This will in future automatically write the SDIF directory (section 5).
6. Name–value tables in `1NVT` frames for spectral envelopes and bookkeeping information, stream info tables in `1IDS` frames for *Chant*.

querysdif Displays a summary of the data in an SDIF file, and the ASCII header information, in `1NVT` name–value tables and others, e.g.:

```
Data in file piano-res.sdif (4792 bytes):
    1 1REB frames in stream 1 between time 0.0 and\
                                    0.0 containing
      1 1RES matrices with max. 174 rows, 5 columns
      1 1CHA matrices with max. 174 rows, 1 columns
    4 1NOI frames in stream 2 between time 0.0 and\
                                    1.3 containing
      4 1DIS matrices with max.   1 rows, 2 columns
```

sdifextract Uses the SDIF selection to output part of the data of an SDIF file. The output can be in one of the 3 formats SDIF, multi-bpf (text lines of frame time and all selected columns), or in the frame-oriented text format of *Additive*. For example, to view the confidence of a pitch estimate over time, pipe the output into the bpf-plotter *XSedit*:

```
sdifextract -bpf mongol.sdif::1FQ0.2 | XSedit
```

sdiftotext, texttosdif Converts to text form and back to SDIF, while preserving the frame and matrix structure.

pmconvert Convert between *Additive* text and binary formats and SDIF, deducing the types from the filename extensions. It is the basis of a **Macintosh drag&drop converter**, distributed with *Diphone* (Rodet & Lefèvre, 1997).

formattosdif, fmttosdif, picstosdif, f0tosdif, sdiftoformat, sdiftofmt, sdiftof0 Aliases to **pmconvert** with fixed input/output types, SDIF input allowing SDIF selection.

CNMAT's SDIF library provides these SDIF utilities:

spew-sdif Print all data in an SDIF file, checking it for validity

copy-sdif Copy an SDIF file by parsing it and writing each frame header and matrix to the new file (pedagogical example program)

sf2sdif, sdif2sf Convert a sound file to an SDIF file with 1TDS frames, vice versa to one or more sound file(s)

dotformat2sdif, f0tosdif, F0tosdif, pics2sdif, i2sdif, res2sdif Conversion utilities from legacy analysis result formats

merge-sdif Combine 2 or more SDIF files into a single SDIF file, renumbering streams if necessary

4.2 Sound Analysis–Synthesis

IRCAM's additive sinusoidal analysis–synthesis packages *Additive*, *Partial++*, and *Hmm* all read and write pitch and partial data in SDIF. So does *Diphone*, which also analyses and synthesizes resonance model and FOF data for *Chant* in SDIF.

A general SDIF reader/writer has been integrated into *jMax* (Déchelle, Schnell, Borghesi & Orio, 2000) by Patrice Tisserand. The reader/writer for *Max/MSP* (Wright, Dudas, Khoury, Wang & Zicarelli, 1999) has been improved: An SDIF-buffer may now hold any number of SDIF streams, and MSP can now look up data in an SDIF stream based on "relative" time, a ratio from zero (the smallest time-tag in the stream) to one (the largest), rather than giving an absolute time in seconds. CNMAT's *Open Sound World* (Chaudhary, Freed & Wright, 2000) real time synthesis and processing environment also includes support for SDIF.

The **Analysis–Synthesis Comparison** panel session at ICMC 2000 has been a major impetus for the adoption of SDIF outside of IRCAM and CNMAT. The idea of the panel is all participants analyse the same set of input sound files with their various analysis–synthesis software, writing results as SDIF files. This allows a meaningful and interesting comparison of subtle and not-so-subtle differences among the different analysis techniques.

Here is the list of participants in the comparison session at the time of writing this paper: *Additive* (Rodet, 1997), *Hmm* (Depalle, Garcia & Rodet, 1993), *Loris* (Fitz, Haken & Chirstensen, 2000), *MDSynth* (Wakefield, 1998), *Partial++*, *SINOLA* (Peeters & Rodet, 1999), *SNDAN* (Beauchamp, 1993), *SMS* (Serra, Bonada, Herrera & Loureiro, 1997). All have committed to supporting SDIF input and output with their analysis–synthesis systems.

4.3 Computer Aided Composition

OpenMusic (Assayag, Rueda, Laurson, Agon & Delerue, 1999) is an object-oriented visual programming environment for music composers, based on CLOS (section 3.2). Objects are symbolized by icons and operations are performed by dragging an icon from a particular place and dropping it onto another. A multitude of classes implementing musical data and behaviour are provided. They are associated with graphical editors and may be visually subclassed by the user to meet specific needs.

SDIF files and access functions from the Lisp interface (section 3.2) are reified as graphical classes and methods. These allow high-level access to SDIF data (see figure 1), editing, and writing. An application of SDIF in *OpenMusic* for synthesis is described in (Assayag, Agon & Stroppa, 2000): generation of control data for the *Chant*-synthesizer.

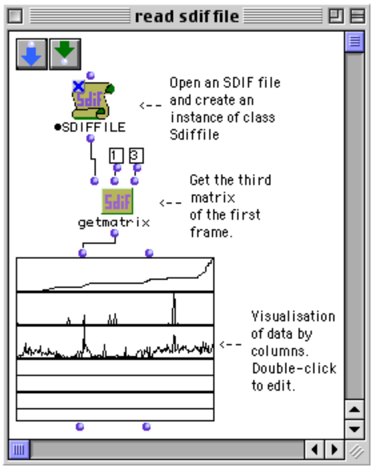

Figure 1: Reading from an SDIF file with *OpenMusic*

5 SDIF FRAME DIRECTORY

Many applications would benefit from rapid random access to data at a given time in an SDIF file, without scanning the whole file. We propose a way to add a frame directory to an SDIF file, which records the file position, type, time, and other information about each data frame occurring in the file.

The SDIF frame directory is best represented as a normal SDIF description type. Thus, programs unaware of this extension can ignore it completely. However, as soon as a program uses a directory-aware SDIF library, the frame directory will be generated and used automatically, e.g. for SDIF selection.

According to our proposition, an SDIF file with a frame directory looks like in figure 2. The directory pointer is the first frame of the file, containing the file position of the directory so that it can be found immediately. The SDIF directory contains the stream IDs, signatures, times, and file positions of all SDIF data frames.

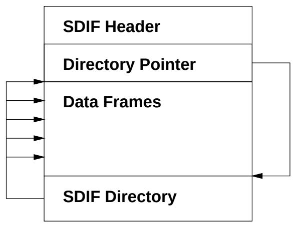

Figure 2: Schema of an SDIF file with frame directory

5.1 Representation of the Directory

An SDIF Directory is represented as a `1DIR` SDIF frame. Its time is the time of the last data frame, to preserve the time ordering, as stipulated by the SDIF standard. The SDIF directory consists of directory entries, one for each data frame. The directory pointer is an SDIF directory itself with a single entry, recording the file position of the SDIF directory frame.

To represent the directory entries, we propose a flat representation, which stores the entries as one list of tuples of *frame time*, *signature*, *stream ID*, *position*. Although introducing redundancy, it offers the most efficient handling and access of the directory. A detailed discussion of the representation can be found on IRCAM's SDIF pages at `http://www.ircam.fr/sdif`.

The list of directory entries is represented in two SDIF Matrices `IDIR` and `1DIR` as shown below, with their data types and column names. One entry is constituted by corresponding rows from the matrices.

```
IDIR double {time}
1DIR uint32 {frame-type, stream-id, filepos}
```

5.2 More Details on Representation

File positions are represented by the number of bytes counting from the start of the file, usable with `fseek()`.[7]

The **directory pointer** is an SDIF directory containing a single entry: the file and time position of the SDIF directory it points to. Even if we don't write the SDIF directory, each file should include a null directory pointer right after the header, which can be used as the standard place to store the size and end time of a file.

Extension of the SDIF directory is possible by adding more information about the frames as additional matrices in the `1DIR` frame, or additional columns in the matrices, which are ignored by other programs.

Updating and consistency: When the frame time, type, or stream ID get overwritten, the directory has to be updated or invalidated by setting the directory pointer to null. On re-writing a file, the library would regenerate the SDIF directory. To catch the worst case in which programs change or add some data, and pass the rest through, not being aware of the special meaning of the directory, some sanity checks are performed: First, the target position of the directory pointer can be checked if it actually contains the SDIF directory, if the frame time is correct, and if it is the last frame in the file. Second, every access through the directory can be validated by cross-checking if the frame found at the recorded position is really of the recorded type, time, and stream.

5.3 Design Decisions to be Taken

Before our proposal of an SDIF frame directory is standardised, some design decisions have to be made, based on questions like: What is the most frequent access/query on a directory? It will probably be something like: Where is the frame of stream n with type `1ABC` closest to time xyz?

How often do updates within a file occur versus appends, or do files normally get rewritten when they are edited? This determines whether to support **multiple SDIF directories**: The definition could be extended to allow later addition of data to an existing SDIF file, without having to rewrite its frame directory. The SDIF directory would then end with a directory pointer, containing the file position of the next directory for data that has been appended, so that we get a linked list of SDIF directories.

6 CONCLUSIONS AND FUTURE WORK

The synergetic effects of the interchange standard SDIF are already visible: *OpenMusic* generates control data for *Chant* and *jMax*, *Diphone* edits data for *jMax* and soon will work as a generic SDIF data editor. All applications will immediately benefit from direct access through the SDIF directory. Notably *Diphone*, *OpenMusic*, and all applications using the SDIF selection will only have to be relinked with the enhanced IRCAM SDIF library.

An important extension is the **SDIF Stream Relationships Language**, discussed in (Wright, Chaudhary, Freed, Khoury, Momeni, Schwarz & Wessel, 2000), a formal language for describing the relationships between streams in an SDIF file, based on XML.

We are working to define an XML-based formal **SDIF Type Definition Language** for defining description types, to replace IRCAM's `1TYP` frames. It would define the matrices that must appear in a frame of a given type, define the required and optional columns of a given matrix type, including name, units, and legal ranges, and define the legal matrix data types for each matrix type.

The extensions described here constitute considerable enhancements for SDIF users (utilities and selection), programmers (foreign languages and APIs), and existing and future applications (frame directory), thus accelerating the proliferation of the SDIF format in a multitude of fields and applications.

ACKNOWLEDGEMENTS

The authors wish to thank Xavier Rodet, Carlos Agon, Amar Chaudhary, and David Wessel for their support with this article.

REFERENCES

Assayag, G., Agon, C., & Stroppa, M. (2000). High Level Musical Control of Sound Synthesis in OpenMusic. In *Proc. ICMC*.

Assayag, G., Rueda, C., Laurson, M., Agon, C., & Delerue, O. (1999). Computer Assisted Composition at Ircam: PatchWork & OpenMusic. *Computer Music Journal*, *23*(3).

Beauchamp, J. W. (1993). Unix Workstation Software for Analysis, Graphics, Modification, and Synthesis of Musical Sounds. In *Proc. AES*.

Chaudhary, A., Freed, A., & Wright, M. (2000). An Open Architecture for Real-time Music Software. In *Proc. ICMC*, Berlin.

Déchelle, F., Schnell, N., Borghesi, R., & Orio, N. (2000). The *jMax* Environment: An Overview of New Features. In *Proc. ICMC*, Berlin.

Depalle, P., Garcia, G., & Rodet, X. (1993). Tracking of Partials for Additive Sound Synthesis Using Hidden Markov Models. In *IEEE Trans.*, (pp. 225–228).

Fitz, K., Haken, L., & Chirstensen, P. (2000). A New Algorithm for Bandwidth Association in Bandwidth-Enhanced Additive Sound Modeling. In *Proc. ICMC*, Berlin.

Peeters, G. & Rodet, X. (1999). SINOLA: A New Analysis/Synthesis Method using Spectrum Peak Shape Distortion, Phase and Reassigned Spectrum. In *Proceedings of the International Computer Music Conference (ICMC)*, Beijing.

Rodet, X. (1997). Musical Sound Signals Analysis/Synthesis: Sinusoidal+Residual and Elementary Waveform Models. In *Proc. IEEE Time–Frequency/Time–Scale Workshop*.

Rodet, X. & Lefèvre, A. (1997). The Diphone Program: New Features, new Synthesis Methods and Experience of Musical Use. In *Proc. ICMC*, Tessaloniki.

Schwarz, D. (1998). *Spectral Envelopes in Sound Analysis and Synthesis*. Diplomarbeit, Universität Stuttgart, Informatik.

Serra, X., Bonada, J., Herrera, P., & Loureiro, R. (1997). Integrating Complementary Spectral Models in the Design of a Musical Synthesizer. In *Proc. ICMC*, Tessaloniki.

Wakefield, G. H. (1998). Time–Pitch Representations: Acoustic Signal Processing and Auditory Representations. In *Proc. IEEE Intl. Symp. Time–Frequency/Time–Scale*, Pittsburgh.

Wright, M., Chaudhary, A., Freed, A., Khoury, S., Momeni, A., Schwarz, D., & Wessel, D. (2000). An XML-based SDIF Stream Relationships Language. In *Proc. ICMC*, Berlin.

Wright, M., Chaudhary, A., Freed, A., Khoury, S., & Wessel, D. (1999). Audio Applications of the Sound Description Interchange Format Standard. In *AES 107th convention*.

Wright, M., Dudas, R., Khoury, S., Wang, R., & Zicarelli, D. (1999). Supporting the Sound Description Interchange Format in the Max/MSP Environment. In *Proc. ICMC*, Beijing.

7. In 32 bits, we could then have files up to 4 GB in size. As soon as this will be not enough for a future application, we can use `2DIR` frames with 64 bit file positions.