Supporting the Sound Description Interchange Format in the Max/MSP Environment

Matthew Wright, Richard Dudas, Sami Khoury, Raymond Wang, David Zicarelli*

CNMAT, UC Berkeley, {matt,dudas,khoury,raywang}@cnmat.berkeley.edu
* Cycling '74, zicarell@cycling74.com

We have added support for the Sound Description Interchange Format to the Max/MSP environment. We briefly introduce SDIF and Max/MSP, describe how SDIF data is represented in MSP and how to write programs to manupulate SDIF data, and demonstrate example applications.

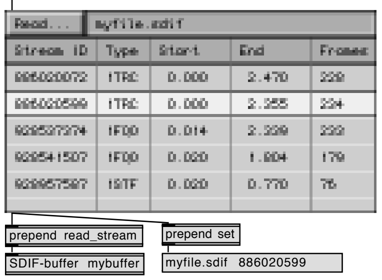
1. Introduction

Max/MSP (Puckette, 1988, Zicarelli, 1998) is a real-time signal- and event-processing environment for music. The Sound Description Interchange Format ("SDIF") is a framework for representing high-level sound descriptions such as sum-of-sinusoids, noise bands, time-domain samples, and formants (Wright, et al., 1999, Wright, et al., 1998). Many of these sound descriptions are mutable models, affording operations such as filtering, morphing, pitch shifting, and time stretching (Laroche, 1998). We describe the situating of these models within an environment that provides a variety of ways in which to do the mutating.

2. Representing SDIF data in Max

SDIF data consists of time-tagged *frames*, each containing one or more two-dimensional *matrices* of data such as floating-point numbers and a *frame type ID* indicating what kind of sound description the frame represents. For example, an SDIF frame representing additive synthesis data has a single matrix where rows represent individual sinusoids and columns represent parameters such as frequency, amplitude, and phase. A *stream* is a sequence of frames of the same frame type that represents a single "sonic object" evolving through time. An SDIF file may contain one stream, or multiple streams with interleaved frames. Each frame has a *Stream ID*, a meaningless 32-bit number that uniquely identifies the stream to which it belongs.

Unfortunately, Max/MSP's limited language of data objects does not support SDIF's structure of matrices within frames within streams. We circumvent this limitation with an object called `SDIF-buffer` that represents an SDIF stream in memory, analogous to MSP's `buffer~` object, which represents audio samples in memory. This allows SDIF data to be represented with C data structures. Each `SDIF-buffer` has a symbolic name chosen by the Max programmer.



Stream ID	Type	Start	End	Frames
886020072	1TRC	0.000	2.470	229
886020599	1TRC	0.000	2.355	224
929527374	1FQ0	0.014	2.329	222
929541507	1FQ0	0.020	1.804	179
929957587	1STF	0.020	0.770	76

Figure 1: A graphical interface for selecting a stream from an SDIF file

One difference between SDIF and sound files is that a single SDIF file can contain multiple sonic objects in different streams. In our design an `SDIF-buffer` holds a single SDIF stream. Therefore, to read SDIF data, the user must specify both a file and a stream within that file. To keep the user from having to type Stream ID numbers, we provide a graphical tool, shown in Figure 1, that displays information about each stream in a given file, including the type of the frames in the stream, the total number of frames, and the times of the first and last frames. Clicking inside this display selects a stream; this outputs a list containing the name of the SDIF file and the appropriate Stream ID.

MSP has objects that provide various control structures to read data from a `buffer~` and output signals or events usable by other Max/MSP objects, and other objects that write data to a `buffer~`. By analogy, we have created "SDIF selector" objects that select data from an `SDIF-buffer` and shoehorn it into a standard Max/MSP data type, and "SDIF mutator" objects that write new data into an `SDIF-buffer`. We provide a header file that allows programmers to create new SDIF selectors and mutators in C as Max external objects.

3. Applications

3.1. Resonance Synthesis

SDIF's resonance type consists of frequency, amplitude, decay rate, and phase for any number of resonances. This data may be interpreted as specifications for a collection of exponentially decaying sinusoids or as parameters for a parallel bank of resonant filters. (These are equivalent because the impulse response of the specified filter bank should be the same as the specified collection of decaying sinusoids.)

Two MSP externals have been written at CNMAT to perform resonance synthesis: `resonators~` implements a parallel bank of resonant filters (Jehan, et al., 1999) and `decaying-sinusoids~` synthesizes a collection of exponentially decaying sinusoids. Both

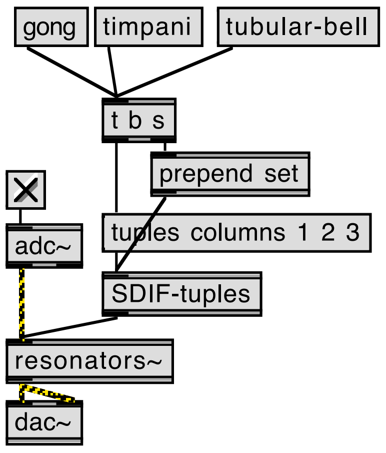


Figure 2: An SDIF-controlled resonant filter bank

of these objects take their parameters as Max lists of concatenated triples of frequency, gain, and decay rate. For example, the list "100. .5 1. 200 .3 1.2" specifies two resonances: 100 Hertz, gain .5, decay rate 1., and 200 Hertz, gain .3, decay rate 1.2.

The SDIF selector `SDIF-tuples` outputs each row of a particular matrix as a tuple containing the desired columns. The "individual" mode outputs each row as a separate list, while the default "concatenate" mode concatenates all of the tuples into one large list.

Figure 2 shows an example of live sound input exciting a bank of resonant filters. The three `SDIF-buffers` holding the gong, timpani, and tubular-bell resonance models are not shown. Clicking on any of the three message boxes first sets the `SDIF-tuples` selector to the desired `SDIF-buffer`, then causes it to output a concatenated list of triplets of columns 1, 2, and 3 from the SDIF matrix. (The `resonators~` object does not support control of phase, the fourth column of the SDIF resonance matrix.) Figure 3 shows the same collection of resonance models used as inputs to `decaying-sinusoids~`.

3.2. Time-Domain Samples

SDIF has a frame type for time-domain samples. Each column is a channel (e.g., left and right), and each row is a sample frame. The SDIF selector `SDIF-extract-samples` copies this sample data from an `SDIF-buffer` into a `buffer~`, making it available for use in the rest of MSP. Optional arguments select a subset of the channels in the SDIF stream.

3.3. Synthesizing F0 estimates

SDIF has a type to represent a time-varying fundamental frequency ("F0") estimate. The rows are the different estimates and the columns are the fundamental frequency (in Hertz) and a confidence factor between 0 and 1. Obviously, there is not enough information to reconstruct an original signal from its F0 envelope, but it can be useful to "synthesize" an F0 envelope by applying it to an arbitrary synthesized timbre, as shown in Figure 4.

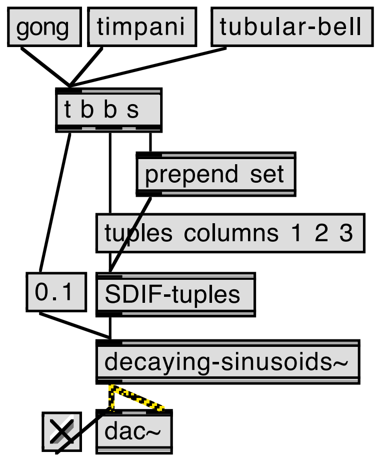


Figure 3: Synthesis of an SDIF resonance model with exponentially decaying sinusoids

Since an F0 envelope changes over time, we need a control structure for moving through the time axis of the SDIF stream. To replay the original F0 envelope at the original speed, we start at time zero in the SDIF stream and advance at the rate of one SDIF second per real-time second. Of course we can advance our position in the SDIF stream in more interesting ways for time-scale modification. We use the term "virtual time" to mean a time position in an SDIF stream.

In Figure 4 we use MSP's `line~` object to generate a function of virtual time as it varies over "real" time. This particular F0 trajectory is two seconds long, so to play it at normal speed we send the list "0 0. 2000 2." to `line~` so that it will generate a signal that starts at zero and goes to two over 2000 milliseconds. (There is an SDIF selector called `SDIF-info`, that outputs information about an `SDIF-buffer`

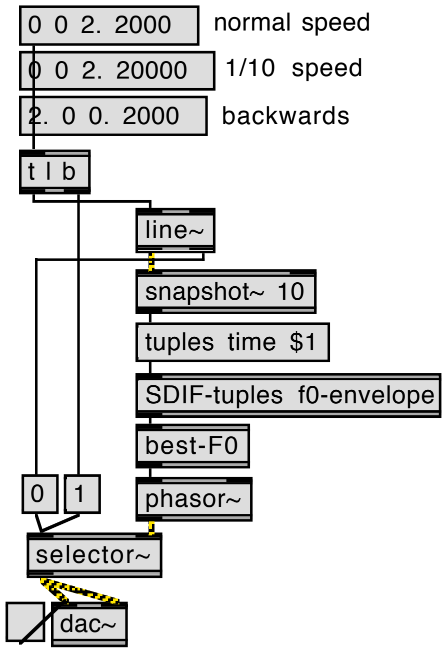


Figure 4: "Synthesis" of a two-second fundamental frequency envelope with time-scale modification

such as the times of the first and last frames; this could be used instead of building the two second length of this particular SDIF stream into the patch.)

By default, `SDIF-tuples` selects the frame whose time tag is closest to zero, but the optional "time" argument to the "tuples" message asks `SDIF-tuples` to find the frame closest to a given time. The `snapshot~` object samples the virtual time signal every 10 ms to produce a time value for `SDIF-tuples`. The `best-F0` subpatch (not shown) selects the highest-confidence F0 estimate, which is passed to `phasor~` to generate a sawtooth wave at that frequency. The `selector~` object turns on the sawtooth wave when we start and turns it off again when the virtual time trajectory is complete. This `selector~` object could be replaced by a ramp attenuation subpatch for click-free performance.

3.4. Interpolating SDIF selectors

When we slow down virtual time enough, the frame-by-frame nature of the SDIF data starts to become audible as frequency discontinuities. We can avoid this problem by performing interpolation of the SDIF data. By default, `SDIF-tuples` returns data from the frame closest to the given time. The "interp" argument causes interpolation along the time axis of the SDIF stream. Given a desired virtual time, `SDIF-tuples` finds the next earlier and next later frames in the stream, and the ratio of the given time between the times of the two frames. `SDIF-tuples` matches each datum in the first frame with the corresponding datum in the second, scales each by the appropriate ratio, and adds them together to produce the result.

3.5. Additive Synthesis

SDIF's sinusoidal track type contains frequencies, amplitudes, and phases for a collection of sinusoids as they evolve over time. The first column is an "index number," used to identify each sinusoid so that it can maintain its identity from frame to frame.

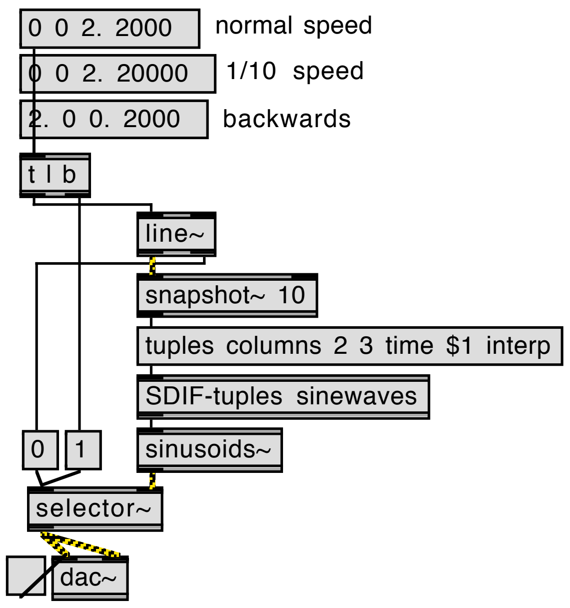


Figure 5: Additive synthesis of SDIF sinusoidal track data

The easiest way to synthesize an arbitrary number of sinusoids in Max/MSP is with CNMAT's `sinusoids~` object (Jehan, et al., 1999), which takes a list of alternating frequencies and amplitudes for a bank of sinusoids. (The `sinusoids~` object does not support control of phase.) Upon receipt of a new list, the freqency and amplitude of each currently-sounding sinusoid is interpolated smoothly from the current value to the new value.

Figure 5 shows additive synthesis of SDIF sinusoidal track data with the `sinusoids~` object. When we ask `SDIF-tuples` to use interpolation with sinusoidal track data, it matches the sinusoids in each frame based on their index numbers rather than their row positions in the matrices.

3.6. More Sophisticated Control of Virtual Time

All the examples shown so far have used MSP's `line~` object to create a signal containing the instantaneous virtual time. MSP's rich collection of signal processing objects can be used to create and modify these virtual time signals in interesting and musically expressive ways, for example, `phasor~` could be used for looping.

We have also written an external object called `time-machine~` that generates virtual time signals under real-time control. This object is a reimplementation of the CAST time machine (Freed and Wright, 1998) and has the same features.

3.7. STFT Analysis/Synthesis

Transform-based algorithms are the commonly preferred solution for convolution, pitch analysis, and additive synthesis. We use SDIF's frame type for windowed Short Time Fourier Transform results to realize a pair of spectral signal processing objects (`SDIF-stft~` and `SDIF-istft~`) for MSP.

Each object which processes STFT data has the name of an `SDIF-buffer` as a first argument; it writes its output to this `SDIF-buffer` and passes the name via its outlet to subsequent objects that will read from that `SDIF-buffer`. Thus, once an incoming audio signal has been buffered and transformed, it can be processed in the SDIF domain without needing to be re-buffered into an MSP audio signal until it is inverse-transformed, thereby decreasing overall latency.

The MSP signal connections between the SDIF processing objects send a "sync" signal that has the value one when the object's `SDIF-buffer` is ready to be processed, and zero otherwise. Using a sync signal keeps the processing of `SDIF-buffers` within MSP's audio interrupt and causes MSP's signal compiler to correctly order the execution of SDIF processing. `SDIF-stft~` and `SDIF-istft~` perform windowing and frame overlap, so when the overlap rate is high they communicate new buffers more frequently.

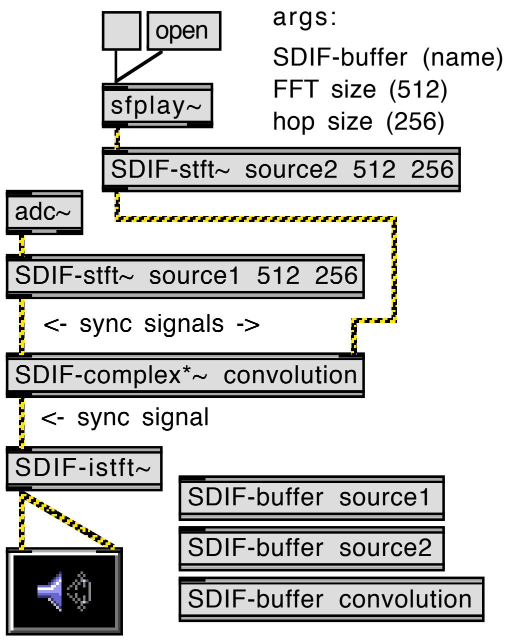


Figure 6: Low-latency convolution using SDIF

Figure 6 shows an example of frequency-domain convolution using the `SDIF-stft~`, `SDIF-istft~`, and `SDIF-complex*~` objects.

3.8. Streaming SDIF

We have started to pursue Internet streaming applications of SDIF. Early results suggest that existing streaming protocols such as the Real-time Transport Protocol ("RTP") (Schulzrinne, et al., 1996) and Real Time Streaming Protocol ("RTSP") (Schulzrinne, et al., 1998) provide an adequate framework for delivery of SDIF data, and support multicasting. As research continues towards more efficient and musically interesting streaming protocols, Max/MSP interfaces will be implemented as SDIF mutator objects that read incoming streamed data and insert it in real time into an `SDIF-buffer`.

Our work is focused on streaming protocols carried over TCP/IP and UDP, the two main transport protocols used in the Internet. We have written two Max external objects, `OTUDP` and `OTTCP`, that use Apple's Open Transport networking architecture to send and receive UDP packets and TCP streams, respectively. These objects do not impose any particular protocol on the data they transmit, so they can be used with Open Sound Control, http, telnet, RTSP, etc.

3.9. Display of SDIF Data

Max/MSP can be used for graphical display of SDIF data as well as modification and synthesis.

Resonance data can be displayed using special features of the LCD object, as illustrated elsewhere in these proceedings (Jehan, et al., 1999). The LCD object can also be used to display F0 trajectories and other parameters as a function of time. The LCD object can also display time-varying data by updating its display in real time; this is useful for spectral data such as STFT results and frequencies and amplitudes of sinusoids.

The Open Sound Edit ("OSE") framework (Chaudhary and Freed, 1999, Chaudhary, et al., 1998) provides 3D graphical display and editing of SDIF data. Max can communicate with OSE by sending OpenSound Control (Wright and Freed, 1997) messages containing data from SDIF frames. OSE is based on OpenGL, which currently runs on the Macintosh (Apple, 1999). Work is underway to port OSE to the Macintosh.

4. References

Apple (1999), "Apple Ships OpenGL For Macintosh" http://www.apple.com/pr/library/1999/may/10opengl.html

A. Chaudhary and A. Freed (1999), "Visualization, Editing and Spatialization of Timbral Resources using the OSE Framework," proceedings of the Audio Engineering Society 107th Convention.

A. Chaudhary, A. Freed, and L. A. Rowe (1998), "OpenSoundEdit: An Interactive Visualization and Editing Framework for Timbral Resources," proceedings of the International Computer Music Conference, Ann, Arbor, Michigan.

A. Freed and M. Wright (1998), "CNMAT's Additive Synthesis Tools," http://www.cnmat.berkeley.edu/CAST

T. Jehan, A. Freed, and R. Dudas (1999), "Musical Applications of New Filter Extensions to Max/MSP," proceedings of the ICMC, Beijing, China.

J. Laroche (1998), "Time and pitch scale modification of audio signals," in *Applications of Signal Processing to Audio and Acoustics*, M. Kahrs and K. Brandenburg, Eds. New York: Kluwer Academic, pp. 279-310.

M. Puckette (1988), "The Patcher," proceedings of the Proceeings of the 14th International Computer Music Conference, Koln.

H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson (1996), "RFC 1889: RTP: A Transport Protocol for Real-Time Applications" http://www.faqs.org/rfcs/rfc1889.html

H. Schulzrinne, A. Rao, and R. Lanphier (1998), "RFC 2326: Real Time Streaming Protocol (RTSP)" http://www.faqs.org/rfcs/rfc2326.html

M. Wright, A. Chaudhary, A. Freed, S. Khoury, and D. Wessel (1999), "Audio Applications of the Sound Description Interchange Format Standard," proceedings of the Audio Engineering Society 107th Convention.

M. Wright, A. Chaudhary, A. Freed, D. Wessel, X. Rodet, D. Virolle, R. Woehrmann, and X. Serra (1998), "New Applications of the Sound Description Interchange Format," proceedings of the International Computer Music Conference, Ann, Arbor, Michigan.

M. Wright and A. Freed (1997), "Open Sound Control: A New Protocol for Communicating with Sound Synthesizers," proceedings of the International Computer Music Conference, Thessaloniki, Hellas.

D. Zicarelli (1998), "An Extensible Real-Time Signal Processing Environment for Max," proceedings of the International Computer Music Conference, Ann Arbor, Michigan.