

# Sound Processing Kit

## An Object-Oriented Signal Processing Framework

*Kai Lassfolk*

Department of Musicology  
P.O.Box 35 (Vironkatu 1)  
00014 University of Helsinki  
Email: kpl@elisir.helsinki.fi

### Abstract

This paper describes a new version of the Sound Processing Kit software system, a C++ class library for audio signal processing. Basic architecture and features are described. Among the new features discussed are enhanced interconnection of objects and new classes for sound processing, synthesis, analysis and editing.

## 1 Introduction

Sound Processing Kit (abbr. SPKit) is an object-oriented signal processing environment designed especially to allow easy prototyping and experimentation with new signal processing algorithms. Since its introduction in 1995, SPKit has been used in a wide variety of signal processing tasks also beyond musical data processing. Enhancements made to the basic software architecture, and invention of new signal processing modules, led to the release of a new, partly rewritten version, hence called SPKit2.

SPKit is aimed at programmers and students involved with audio signal processing. The system consists of a class library written in the C++ programming language and a set of example programs demonstrating the use of the SPKit classes and development of new classes. SPKit is distributed with complete source code and includes online documentation in HTML format.

SPKit provides an object-oriented and thus modular environment for using and constructing signal processing networks. SPKit contains both primitive and complex objects, the former performing simple tasks, such as filtering or amplification, and the latter for performing more complicated tasks, such as reverberation.

SPKit provides a simple API for developing new algorithms especially for time domain signal processing tasks. SPKit provides a common base class that implements default generic functions for connecting signal processing objects and for transmitting audio samples objects. By using inheritance the base class can serve as a jacket for new algorithms or as a "patch-bay" for connecting existing SPKit objects.

SPKit was originally developed to serve as a teaching aid for a course on audio signal processing aimed at an audience with little or no prior knowledge

of signal processing or C++ programming. Thus, the fundamental design principle was to enable fast and simple implementation and testing of algorithms. Advanced C++ language features were avoided to reduce the learning curve. Run time efficiency was not a primary concern. However, with current PC hardware, SPKit is fast enough to be used for realtime processing.

The first major SPKit version [Lassfolk 1995], hereafter referred to as SPKit1, contained more than 20 classes. The class selection was focused on processing prerecorded sound files. The new major version was prepared to enable the construction of more advanced processing networks and to exploit the processing power of modern computer hardware.

SPKit1 had restrictions in connecting certain objects to signal multiplexers. To solve the problem, complete backwards compatibility with the old version had to be sacrificed. However, only minor syntactic modifications were required in porting the old classes.

SPKit1 was originally developed on hardware that was not fast enough to run SPKit tasks in real time. Therefore, I/O was supported only on sound files. Abstractions for AD and DA converters were added to the new version. In the design process of SPKit2, issues of sound synthesis and frequency domain signal processing were also addressed.

## 2 Classes and features

SPKit contains classes for signal flow control (multiplexing, mixing, signal feedback etc.), time domain filtering, amplification, reverberation, dynamics processing and tube distortion [Lassfolk 1996]. Among the new classes in SPKit2 are an interpolating oscillator, variable and multi-tap delay lines and a sample rate converter. Also, a framework for short

time Fourier transform and a class set for nondestructive signal editing are included.

While SPKit2 includes several command line programs, including a command line based signal editor, the system still remains principally a programming framework. In particular, no graphical user interface is included or planned for the regular SPKit distribution.

In addition to sound file manipulation, SPKit2 supports realtime sound processing through classes interfacing to audio devices.

### 3 Using SPKit classes

A signal processing task using SPKit classes is defined by constructing a signal flow by interconnecting SPKit objects. A typical signal flow consists of a “reader” object, one or more “processor” objects, and a “writer” object. The reader and writer handle signal input and output, respectively, while the processor objects take care of the actual processing.

After the objects have been connected, initial signal processing parameters may be set for the processor objects to complete the initialization phase. Next, a “run” function is called on the writer object to start the actual processing. The signal (e.g. a prerecorded sound file) may be processed either completely by a single run function call or in user defined chunks by calling a run function repeatedly.

SPKit provides classes named SPKitReader and SPKitWriter for instantiating reader and writer objects. The processor objects are instances of classes derived from the base class SPKitProcessor. Among its subclasses are SPKitAmp (amplifier), SPKitOscillator (a table lookup oscillator), SPKitDelay (delay line), SPKitCompressor (amplitude compressor), SPKitSchroederReverb (diffusive reverberator) and others, some having less obvious names.

Complex objects are often constructs of other, more primitive and generic objects. For example, SPKitSchroederReverb is a combination of SPKitAmp, SPKitMux (signal multiplexer), SPKitComb (comb filter) SPKitAllpassNetwork (allpass filter) and SPKitSum (signal mixer) classes. SPKitComb and SPKitAllpassNetwork are also constructs of other SPKit classes.

### 4 Protocol issues

An SPKit processor object has two principal member functions for performing a signal processing task: 1) `setInput()`, for connecting the object to its signal input (a reader or another processor object) and 2) `getSample()`, for retrieving a processed audio sample. The SPKit root class, `SPKitProcessor`, contains a default implementation of both functions. Derived classes

typically override one or both of the functions to add their own functionality.

The `setInput()` function is used to perform initialization operations of an object. For example, parameters of the processed audio signal, such as sampling rate and channel count are automatically obtained from an object’s signal input by the `setInput()` implemented in the `SPKitProcessor` class.

`SPKitProcessor` also implements a `setOutput()` function, which is called automatically by `setInput()` to build a two-way connection between an object and its input. This enables the input to identify and count the objects that it is connected to.

When a run function is called on the writer object, it requests samples from its signal input by repeatedly calling the `getSample()` function which produces a single sample on each call. The signal input may, in turn, request audio or control signal samples from its own signal input or inputs.

As the return value `getSample()` provides an integer indicating possible end of input signal. The requested sample is assigned to a variable passed to the function by reference. Audio samples are processed and transmitted as floating point numbers.

In SPKit2, a pointer to the function caller was added as a new parameter to `getSample()`. This enabled an object to distinguish which object is requesting a sample. While most objects discard this information, it removed a limitation of SPKit1 concerning signal multiplexing. Now, it is possible to connect objects that read samples in chunks (i.e. by several consecutive `getSample()` calls) to a general purpose multiplexer object.

The SPKit Writer class has two variants of the run function: `run()` and `runFor()`. The former calls `getSample()` in a tight loop until end of signal is encountered. The latter function takes a sample count as a parameter and returns control to the caller when either the requested amount of samples have been processed or at end of signal. This allows the caller to perform periodical adjustments to signal processing parameters or to perform other operations.

Each `getSample()` call issued by the writer starts a chain of identical calls ending up at an object that can either generate a sample without an input signal (e.g. an oscillator) or retrieve a sample from an external source, such as an AD-converter or file. Control is returned to the writer after all the resulting calls have returned. This remotely resembles the control message passing mechanism used in MAX [Puckette 1991], except that SPKit `getSample()` calls are issued at an logically opposite order to MAX messages (i.e. from output to input vs. top to bottom).

### 5 Portability issues

SPKit has been developed primarily for UNIX compatible environments. However, the class library is

portable with little or no modifications to almost any system with a reasonably modern C++ compiler and ANSI C standard libraries. SPKit supports processors with either little and big endian byte ordering. Sound file support varies between different operating system implementations. On most platforms, Sun/NeXT and WAV formats are supported but under the SGI Irix the default format is AIFF. Realtime signal processing via AD/DA-converters is currently implemented using the Linux sound card API. However, classes using file I/O or AD/DA converters may be easily modified to use alternate APIs.

The main development environment is now Linux running on PC hardware. Occasionally, the code is also tested under other Linux and UNIX variants including mkLinux, HP/UX, SunOS/Solaris, NEXT-STEP and SGI Irix.

## 6 Future directions

Support for real time processing is still at an early stage. Especially, there are potential problems in adjusting processing parameters on the fly. Also, support for multithreaded operation is an open question.

Frequency domain signal processing capabilities are still limited to testing and demonstration purposes. Usable analysis/synthesis applications require additional classes which are under planning. A phase vocoder application remains on the wish list.

With the inclusion of the oscillator class and other synthesis classes currently under development, sound synthesis provides an important extension to SPKit. This also raises a need for additional classes and refinements to existing classes to take full advantage of the new application area.

## 7 Conclusion and acknowledgements

SPKit is a programming library that allows fast prototyping of new signal processing algorithms and offers a set of predefined classes so that complex algorithms don't have to be, in many cases, written from scratch. SPKit offers a portable, object-oriented interface to sound file and AD/DA input and output. Distributed in source form and licensed under the GNU Library General Public Licence, SPKit also offers user-modifiable examples of common signal processing algorithms.

Several students from the Department of Musicology, University of Helsinki, contributed to SPKit2 by writing new classes many of which are included in the software distribution. Among the contributors are Elviira Hartikainen, Anna Immonen, Taneli Mielikäinen, Georgij Putilin, Samppa Saarela, Jaska Uimonen and Jussi Virolainen.

SPKit can be downloaded from the Internet at <http://www.music.helsinki.fi/research/spkit>.

## References

- [Lassfolk, 1995] Kai Lassfolk. Sound Processing Kit. In *Proceedings of the 1995 International Computer Music Conference*. The International Computer Music Association, San Francisco, California, pp. 503-504, 1995.
- [Lassfolk, 1996] Kai Lassfolk. Simulation of Electron Tube Audio Circuits. In *Proceedings of the 1996 International Computer Music Conference*. The International Computer Music Association, San Francisco, California, pp. 222-223, 1996.
- [Puckette 1991] Miller Puckette. Combining Event and Signal Processing in the MAX Graphical Programming Environment. In *Computer Music Journal* Vol. 15 No. 3, pp. 68-77.