SDK User guide

1. Introduction

Copula is a unique audio engine that separates time from pitch. It produces professional quality sound, and it's simple to use. Perfect for any audio or video software, and professional digital audio workstations.

Input feed accuracy.
Copula adds any changes to the sound straight to the input stream so the output sound is always adjusted to that a particular feed sample.

Time stretching.
Copula allows you to stretch or compress a sound without changing pitch. Creating glitch free tempo changes in audio tracks allows the user to manipulate freely any audio in real-time.
It doesn't split the spectrum up into multiple independent sections like other pitch shifters. Processing everything at once allows for a very stable and well timed transients.

Pitch shifting.
This pitches the sound up or down independently from time, allowing tuning of instruments irrespective of the desired tempo.
It can process sound to well over a 4 octave range. Copula's unique pitch shifting engine eliminates the warble effects heard on all other pitch-shifters available at extreme shifts.

Copula SDK key design goals:

- Artefact free pitch and time shifting.
- Efficient and single threaded, suitable for many channels playing at once.
- Should work for music and percussion without user differentiation.
- Easy to use.

2. Overview

Copula is a very small library with just a few functions.
Included is the file 'Copula.h' that defines the Copula interface class, and error codes. This also includes a key that the library licensee inserts to register the SDK.
The class is instantiated with basic information on sample rate and number of channels etc. It will then build all internal memory allocations, and be ready to receive incoming audio.

Processing model

Copula uses a 'push' system so you have full control of it. There are no callback functions, you just simply ask it how many sample frames it wants, feed in the data, call the processing function and grab whatever samples you like. There is also a unique flush system that enables sample accurate pitch and tempo changes, without the need to fade audio out and in at the change points.

3. Installation

PC: The included 'zip' file contains the the library 'dll' file and the 'copula.h' and 'Copula.lib' files located in the 'common' directory. The 'dll' is also in the project release and debug folders. There is a directory with audio samples for the demos called 'SoundFiles,' and when the examples are executed the processed files go into 'ResultFiles.'

Registering the Copula library:

Copula uses a variable key-code to unlock the library, this code is simply copied into the 'Copula.h' file, and full instructions are sent with your purchase.

4. Getting started

The demo:
The two demo files show the basic functioning of Copula. They are in a Visual Studio 2008 compatible solution format. If you can't use that then a quick look at the files will show how simple they really are.
'Copula_Example_1.cpp' and 'Copula_Example_2.cpp' is where all the processing is done. The comments in the code are best place to find out about the process.
Example 2 shows a way of simply setting the pitch, and feeding one sample at a time to achieve sample accurate pitch shifting.

5. SDK Functions

Global functions

coError CopulaCreateInstance(Copula *&inst, int sampleRate, int numChannels);

This creates an instance of the Copula engine. Here you must set the sample rate, number of channels to be processed,
The maximum number of channels is 8, and Copula is expecting an interleaved format of IEEE floating point data.
You can repeatedly use this instance for different operations on the data, but it must be destroyed if you want to change the buffer attributes using the Destroy function.
Note that it is a static function, so you create the instance using the following:

```
coError err = Copula::CreateInstance(cop, 44100, 2);
if (err)
{
	printf(Copula::GetErrorString(err));
	return 0;
}
```

coError CopulaDestroyInstance(Copula *&inst);

Use this function to remove an instance, and nulls the pointer.

int CopulaGetVersion(void);

Returns the version number of the current SDK.

char *CopulaGetErrorString(int coError);

Some of the functions in Copula return a simple error code, use this function to retrieve the meaning of the error.

Class methods

coError InitProcess();
Always call this before you start to call Process. It clears any latency buffers to prevent sounds from the previous process leaking through to the start of the current sound.

int Process();
Process is where all the work is done, it takes any input data it has and outputs data whenever it can, if there is not enough data then it simply doesn't nothing until something is available. The following is a typical frame chunk inside a audio processing function that has a given number of frames.

int Flush();
Flush any remaining data from the inputs and sends the remaining frames to the output. Use this when you're ending your processing, it will automatically sort out any internal needs for more data.
Returns the amount of out frames available after flushing.
*NB The output may be a few samples shorter or longer than expected, this is due to the phase adjustment nature of Copula, what it does mean is that you retrieve all samples given, with no missing inputs. This was considered better than simply cutting off a sample just to satisfy the maths. Remember Copula is all about what goes in!

coError SetPitchSemiTone(float pitch);
Sets the pitch shift in semitone values. A setting of 0, means no shift. Set it to -6.5 means shifting the sound down 6.5 semitones. Any value can be used, so setting to 0.00001f is valid and works as expected.
Accepted pitch values are **-24% to 24%**.

coError SetPitchPercent(float pitch);
Sets the pitch required in percentages rather than semitones. So 100% will be 1:1 as in normal pitch. And 200% would be twice the pitch. 50% would be at half the pitch, or an octave down. Any value can be used, so setting it to 100.00001f is valid.
Accepted pitch values are **25% to 400%**.

coError SetFormantSemiTone(float formant);
Set the format envelope to a semi-tone from normal. Used in the same way as setting pitch for semi-tones. Set at the same value as 'pitch semi-tone' for formant preserving in vocals and instruments e.t.c. Setting 0.0 disables the formant processing.
Accepted formant values are **-24% to 24%**.

coError SetFormantPercent(float formant);
Set the format envelope to a percentage of normal. Used in the same way as setting pitch.
Set at the same value as 'pitch percent' for formant preserving in vocals and instruments e.t.c. 100% disables the formant processing.

coError SetSpeedPercent(float speed);
Sets the speed in percentage values. So for example 100% would mean normal speed forward, -200% is backwards twice the speed, and 0% is frozen. You cannot change pitch while the speed is set to 0.
Note that after setting the speed you must read the 'GetInputFrames' value as this will have changed. Any value can be used, so setting it to 100.00001f is valid.
Accepted speed values are **0% to 400%**.

int GetNumFramesWanted();
Returns a value for the number of frames the process needs to carry on with data processing.

coError FeedInputFrames(float *frames, int num, coFeedType type);
This is called after GetNumFramesWanted, with the required data.
* All current Pitch and Time settings are given to the first sample fed in!
To get sample accuracy you need to set the parameter then feed the data with this function 1 frame at a time. See the 'Copula_Example_2.cpp' file.

This function also uses a special helper paraemter called 'feed type'.
This can be set to Forwards, Backwards, MergeToBackwards, or MergeToForwards.
You will generally use 'Forwards' but if you're creating an audio buffer that goes both forwards AND later backwards it's handy to use this feed type.
* *You don't need to use this feed type at all if you're going to handle the interleaving and reverse feeding yourself. It's just handy to have.*

So to reverse an audio process:

Your own postions pointer is shown as the top number below:

0	512	1024	1024	512
Forwards	Fowards	Forwards	MergeToBackwards	Backwards

Forwards: Just simple feeds in the frame block sequencially.
Backwards: Feed the given block in from last to first.
MergeToBackwards: Used as a cross-over from forwards to backwards direction.
MergeToForwards: Used as a cross-over from backwards to forwards direction.

The quality of the merge depends on the length if the block being fed in, something like 1024 frames will be OK.

int GetNumFramesAvailable();
This returns the number of output frames that can be collected, used for the next function.
coError GetOutputFrames(float *frames, int num);
This sends the 'num' amount of data to '*frames.' Remember this value is in frames so if it's in stereo then the data is twice as much as 'num.' It's up to you to make sure you data buffer can accept the correct amount of data.

int GetNumChannels();
Returns the number of channels the current audio has. This value was given with the intialisation function.

6. Error codes

coNoError
No error from call.

coIncorrectNumChannels
You have set the number of channels to be less 0 or less or over 8.

coOutOfMemory
Somewhere on instantiation the computer ran out of memory.

coInstanceDoesNotExist
Cannot destroy instance as it is null.

coNullBufferAndNoCallbackFunctionSet
You've set the instance's 'inBuffer' to zero, and haven't set the call-back function.

coValueOutOfRange,
You've set the sample rate over 196Khz or below 32Khz, or a pitch/time shift over the limit.

coNotInitialised
Internal error when call any 'Set...' function, possibly run out of main memory?

coNullBuffer
You've tried to grab data from a null pointer.

7. Example loop

```
fopen_s(&file, "sample data", "wb");
// 'bufferIn' and 'bufferOut' are just blocks for sample frames...
// numInFrames is the number of input frames in the sample to process
while (position != numInFrames)
{
	int wants = cop->GetNumFramesWanted();
	if (wants)
	{
		// canHave is easily limited by what's left and what it wants...
		int canHave = numInFrames - position;
		if (canHave > wants)
			canHave = wants;
		// Give Copula what you have...
		cop->FeedInputFrames(bufferIn + position * numChannels, canHave,
					coFeedType::Forwards);
		// Advance input data pointer...
		position += canHave;
	}
}
// Must call Process even if it didn't want any samples this time around...
numOutFrames = cop->GetNumFramesAvailable();
if (numOutFrames)
{
	cop->GetOutputFrames(bufferOut, numOutFrames);
	fwrite(bufferOut, sizeof(float), numOutFrames * numChannels, file);
	printf(".");

}
// Flush out the remaining samples in the output buffer...
cop->Flush();
// Then read out hte output frames like in the above code and close the file.
//.........
// then you must use this to kill the instance...
err = CopulaDestroyInstance(cop);
```

138 Fairview road
Cheltenham
England
GL52 2EU