



<dns> SDK Reference Manual

(c) 2008 by zplane.development

September 23, 2008

Contents

1	<dns> SDK Documentation	2
1.1	Introduction	2
1.2	API Documentation	2
1.2.1	Memory Allocation	2
1.2.2	Naming Conventions	3
1.2.3	API description	3
1.2.4	Usage example	5
1.3	Delivered Files (example project)	7
1.3.1	File Structure	7
1.4	Coding Style minimal overview	7
1.5	Command Line Usage Example	8
1.6	Support	8
2	Directory Hierarchy	8
2.1	Directories	8
3	File Index	9
3.1	File List	9
4	Directory Documentation	9
4.1	incl/ Directory Reference	9
5	File Documentation	9
5.1	dithernsapi.h File Reference	9
5.1.1	Define Documentation	10
5.1.2	Enumeration Type Documentation	10
5.1.3	Function Documentation	11
5.2	docugen.txt File Reference	15
5.2.1	Detailed Description	15

1 <dns> SDK Documentation

1.1 Introduction

The <dns> SDK provides high quality audio conversion between different bit resolutions (or word lengths) and sample rates. Dither is added to avoid correlation between signal and quantization error, while high-order noise-shaping is used to shape the spectrum of the introduced quantization error in a way that its most energy is at high frequencies. Sample Rate Conversion (SRC) uses a high quality conversion algorithm that allows to convert to any – even time varying – sample rate.

Both Noise-Shaping and SRC allow the selection of different quality levels in order to allow the application developer to adjust a trade-off between output quality and produced workload.

The SDK is able to handle any number of channels and any sample rate; however, the higher the output sample rate, the more satisfying the results of the noise shaping process will be.

The probability density function (pdf) of the dither can be selected to be rectangular or triangular. As additional option, the triangular dither may be high pass filtered. While the implementation of other pdfs is no problem, theory (and experience) has shown that high-pass filtered triangular dither is the most advantageous distribution for high quality results. Therefore, additional forms of dither have been discarded intentionally.

As a rule of thumb, a noise-shaping algorithm works the more satisfying the higher its order is. The order however has direct impact on the workload produced by the noise-shaper. The higher-order noise-shaping provided in the higher-order modes is optimized with respect to psychoacoustic principles.

The project contains the required libraries for the operating system the SDK was licensed for with the appropriate header files.

The structure of this document is as following: First, the API of the SDK is being described. The API documentation contains naming conventions and function descriptions. The following usage example (available as source code for compiling the test application) gives a clear example on how to use the API in a real world application. Afterwards, a short usage description of the compiled example application is given.

1.2 API Documentation

The SDK offers an ANSI-C-API which can be accessed via the file [dithernsapi.h](#). All variable types needed are either defined in this file or are standard C-types.

1.2.1 Memory Allocation

The SDK does not allocate buffers handled by the calling application. The input and output buffers have to be allocated by the calling application. The audio buffers are expected to contain the data in interleaved format, i.e. for a stereo file the order is: left[0] right[0] left[1] right[1] etc.

1.2.2 Naming Conventions

When talking about **frames**, the number of audio samples per channel is meant. I.e. 512 stereo frames correspond to 1024 float values (samples). If the sample size is 32bit float, one sample has a memory usage of 4 byte and one stereo frame is 8 byte large.

1.2.3 API description

Using the Dither and Noise-Shaping API is straight forward: just create a new instance, adjust the required settings and initialize the instance, do the processing, and destroy the instance afterwards.

1.2.3.1 Complete Function Description

Instance Handling Functions

- **int [DitherNS_CreateInstance](#) (void **ppHandle, [DnsInputData_t](#) InputDataType, int iInputSampleRate, int iNumOfChannels)**

Creates a new instance of the of the dns. The handle to the new instance is returned in parameter ppHandle. The input data resolution, as given in parameter InputDataType, can be either integer or float, as specified in [DnsInputData_t](#). In the case of integer input data that has less the 32 bit resolution, the bits have to be copied to a 32 bit buffer and shifted towards the high bits. If the application uses floating point data at any processing point, it is recommended to use floating point as input data type to avoid any unnecessary conversions. Naturally, the output data type is integer in any case.

The sample rate and the number of channels of the input audio data to be processed is given in parameter iInputSampleRate resp. iNumOfChannels.

If the function fails, the return value is not 0.

- **int [DitherNS_DestroyInstance](#) (void* pHandle)**

Destroys an instance of the SDK. The handle to instance is in parameter *pHandle.

If the function fails, the return value is not 0.

Parameter Setting and Initialization

- **int [DitherNS_SetBitDepths](#) (void* pHandle, unsigned int uiInputBits, unsigned int uiOutputBits)**

Parameter pHandle contains the previously created instance. Parameters uiInputBits and uiOutputBits specify the required input and output word lengths, respectively.

The instance **has to be initialized after the call of this function.**

If the function fails, the return value is not 0.

- **int DitherNS_SetSrcQuality** (void* pHandle, float fQuality)

Parameter pHandle contains the previously created instance. Parameter fQuality sets the required quality of the resampling in a range between 0 (lowest quality) and 1 (highest quality). This setting may have also huge impact on the workload produced by the processing.

The instance **has to be initialized after the call of this function**.

If the function fails, the return value is not 0.

- **int DitherNS_SetDitherType** (void* pHandle, DnsDither_t iDitherType)

Parameter pHandle contains the previously created instance. Parameter iDitherType specifies the pdf shape of the dithering noise. It offers the options defined in [DnsDither_t](#). The usage of triangular noise, possibly high-passed, is strongly recommended.

If the function fails, the return value is not 0.

- **int DitherNS_SetNoiseShapeType** (void* pHandle, DnsNoiseShaping_t iNSType)

Parameter pHandle contains the previously created instance. Parameter iNSType specifies the mode (i.e. the order) of the noise shaping. It offers the options defined in [DnsNoiseShaping_t](#). The usage of higher orders is usually recommended.

If the function fails, the return value is not 0.

- **int DitherNS_Initialize** (void* pHandle)

Parameter pHandle contains the previously created instance.

This function **has to be called at least once** before the process function is called and after parameters bit depths and src quality have been set.

If the function fails, the return value is not 0.

Processing Functions

- **int DitherNS_GetRequiredNumOfInputFrames** (void* pHandle, float fTargetSampleRate, int iTargetNumOfOutputFrames)

Parameter pHandle contains the previously created instance, and fTargetSampleRate is the output sample rate that will be the argument of the following process call. Parameter iTargetNumOfOutputFrames is the required number of output frames.

The function returns the number of input frames that are required to produce the exact amount of output frames (iTargetNumOfOutputFrames). Thus, **this function has to be called before every call of [DitherNS_Process](#)** to ensure that the output buffer is long enough.

- **int DitherNS_Process** (void* pHandle, const void* pInputBuffer, void* pOutputBuffer, float fTargetSampleRate, unsigned int uiNumOfInputFrames, unsigned int uiNumOfOutputFrames)

Parameter pHandle contains the previously created instance, and pInputBuffer is the pointer to the input data in the specified format. pOutputBuffer, the buffer to

where the processed data is written by this function, is 32 bit int in any case. `fTargetSampleRate` is the required output sample rate (that theoretically may change over time). Parameter `uiNumOfInputFrames` contains the number of frames in `pInputBuffer`, Parameter `uiNumOfOutputFrames` the length of the output buffer (see [DitherNS_GetRequiredNumOfInputFrames](#)).

The instance **has to be initialized before the call of this function**.

If the function fails, the return value is not 0.

Optional Functions The call of the following functions is not mandatory but optional, as they return only information on the internal state of the instance.

- **[DnsDither_t DitherNS_GetDitherType](#) (void* pHandle)**

Parameter `pHandle` contains the previously created instance. The function returns the currently selected dither type as defined in [DnsDither_t](#) (see also [DitherNS_SetDitherType](#)).

- **[DnsNoiseShaping_t DitherNS_GetNoiseShapeType](#) (void* pHandle)**

Parameter `pHandle` contains the previously created instance. The function returns the currently selected noise shaping type as defined in [DnsNoiseShaping_t](#) (see also [DitherNS_SetDitherType](#)).

- **[float DitherNS_GetSrcQuality](#) (void* pHandle)**

Parameter `pHandle` contains the previously created instance. The function returns the currently selected sample rate conversion quality between 0 and 1 (see also [DitherNS_SetSrcQuality](#)). As the sample rate conversion quality is quantized internally, it could slightly differ from the value that has been set.

- **[int DitherNS_GetBitDepths](#) (void* pHandle, unsigned int *puiInputBits, unsigned int *puiOutputBits)**

Parameter `pHandle` contains the previously created instance. Parameters `puiInputBits` and `puiOutputBits` will be written with the previously adjusted input and output word lengths, respectively (see also [DitherNS_SetBitDepths](#)).

If the function fails, the return value is not 0.

1.2.4 Usage example

The complete code can be found in the example source file `DitherNSTestCLMain.cpp` that is included in the SDK package.

Please note that the example source uses a wrapper to the open source library `sndlib` for wav file reading. The corresponding libraries (`libzplAudioFile.lib`) are not needed for using the SDK itself, just for the demo application.

In the first step, a handle to the instance has to be declared:

```
void*          pInstanceHandle = 0;          //!< instance handle
```

Then, the new instance has to be created. Note that the required settings (dither type, noise shaping type, input data type, input resolution, output resolution, sample rate, number of channels) are required for instance creation.

```
// create new instance for dither and noise shaping
iErr = DitherNS_CreateInstance ( &pInstanceHandle,
                                kDnsInpDataFloat,
                                pFInputFile->GetSampleRate (),
                                pFInputFile->GetNumOfChannels ());

if (iErr)
{
    fprintf(stdout, "Instance could not be created!\n");
    return -1;
}
```

Afterwards, the parameters can be adjusted and the instance can be initialized.

```
// set some parameters and initialize
DitherNS_SetDitherType (pInstanceHandle, kDnsDitherNone);
DitherNS_SetNoiseShapeType (pInstanceHandle, kDnsNoiseShapeNone);
DitherNS_SetBitDepths (pInstanceHandle, 32, iOutputBits);
DitherNS_Initialize (pInstanceHandle);
```

Then, the processing can begin!

```
// find out how many frames to read
iNumFrames2Read = DitherNS_GetRequiredNumOfInputFrames ( pInstanceHandle,
                                                         iOutputSR,
                                                         kBlockSize);

// read input data from file in float format
iNumFramesRead = pFInputFile->Read (pfInpDataSp, iNumFrames2Read);
// process
iNumFramesRead = DitherNS_Process( pInstanceHandle,
                                   (void*) pfInpDataI1,
                                   (void*) piIntData,
                                   iOutputSR,
                                   iNumFramesRead,
                                   kBlockSize);
```

In this example application, we write the output data into a text file instead of a binary file. To do so, the output data has to be shifted to the correct range.:

```
// shift and write output data (always in 32 bit integer)
for (i = 0; i < iNumFramesRead*iNumChannels; i+=iNumChannels)
{
    int iTmp = ((piIntData[i])<<(32-iOutputBits));
    for (int j = 0; j < iNumChannels-1; j++)
    {
        iTmp = ((piIntData[i+j])<<(32-iOutputBits));
        fprintf (pFOutputFile, "%d\t", iTmp);
    }
    iTmp = ((piIntData[i+iNumChannels-1])<<(32-iOutputBits));
    fprintf (pFOutputFile, "%d\n", iTmp);
}
```

After the successful processing, the created instance can be destroyed

```
// destroy the instance  
DitherNS_DestroyInstance (&pInstanceHandle);
```

1.3 Delivered Files (example project)

1.3.1 File Structure

1.3.1.1 Documentation This documentation and all other documentation can be found in the directory **./doc**.

1.3.1.2 Project Files The Workspaces, Projectfiles and or Makefiles can be found in the directory **./build** and its subfolders, where the subfolders names correspond to the project names.

1.3.1.3 Source Files All source files are in the directory **./src** and its subfolders, where the subfolder names equally correspond to the project names.

1.3.1.4 Include Files Include files can be found in **./incl**.

1.3.1.5 Resource Files The resource files, if available can be found in the subdirectory **./res** of the corresponding build-directory.

1.3.1.6 Library Files The directory **./lib** is for used and built libraries.

1.3.1.7 Binary Files The final executable can be found in the directory **./bin**. In debug-builds, the binary files are in the subfolder **/Debug**.

1.3.1.8 Temporary Files The directory **./tmp** is for all temporary files while building the projects. In debug-builds, the temporary files can be found in the subfolder **/Debug**.

1.4 Coding Style minimal overview

Variable names have a preceding letter indicating their types:

unsigned:	u
pointer:	p
array:	a
class:	C
bool:	b
char:	c
short (int16):	s
int (int32):	i
__int64:	l
float (float32):	f
double (float64):	d
class/struct:	c

For example, a pointer to a buffer of unsigned ints will be named `puiBufferName`.

1.5 Command Line Usage Example

The compiled example is a command line application that reads and optionally writes audio files in WAV format. The output file is the input file with additional beat ticks.

Since the example application has no sophisticated command line parser, the order of the arguments is crucial. The command line synopsis is:

```
CL input_file output_file output_resolution
```

1.6 Support

Support for the SDK is - within the limits of the agreement - available from:

[zplane.development](mailto:info@zplane.de)

katzbachstr. 21

D-10965 berlin

germany

fon: +49.30.854 09 15.0

fax: +49.30.854 09 15.5

@: info@zplane.de

2 Directory Hierarchy

2.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

[incl](#)

9

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

[dithernsapi.h](#)

9

4 Directory Documentation

4.1 incl/ Directory Reference

Files

- file [dithernsapi.h](#)

5 File Documentation

5.1 dithernsapi.h File Reference

Defines

- `#define _DITHERNSAPI_HEADER_INCLUDED`

Enumerations

- enum [DnsInputData_t](#) { [kDnsInpDataInt](#), [kDnsInpDataFloat](#), [kDnsNumInputDataTypes](#) }
possible input data type
- enum [DnsDither_t](#) {
[kDnsDitherNone](#), [kDnsDitherRect](#), [kDnsDitherTriang](#), [kDnsDitherTriangHp](#),
[kDnsNumDitherTypes](#) }
different dither noise types
- enum [DnsNoiseShaping_t](#) {
[kDnsNoiseShapeNone](#), [kDnsNoiseShapeLow](#), [kDnsNoiseShapeMid](#),
[kDnsNoiseShapeHigh](#),
[kDnsNoiseShapeUltra](#), [kDnsNumNoiseShapeTypes](#) }

different noise shaping modes, for sampling rates above 48kHz, always use DNS_NSTYPE_ULTRA

Functions

- int [DitherNS_CreateInstance](#) (void **ppHandle, [DnsInputData_t](#) InputDataType, int iInputSampleRate, int iNumOfChannels)
- int [DitherNS_SetBitDepths](#) (void *pHandle, unsigned int uiInputBits, unsigned int uiOutputBits)
- int [DitherNS_SetSrcQuality](#) (void *pHandle, float fQuality)
- int [DitherNS_SetDitherType](#) (void *pHandle, [DnsDither_t](#) iDitherType)
- int [DitherNS_SetNoiseShapeType](#) (void *pHandle, [DnsNoiseShaping_t](#) iNSType)
- int [DitherNS_Initialize](#) (void *pHandle)
- int [DitherNS_GetRequiredNumOfInputFrames](#) (void *pHandle, float fTargetSampleRate, int iTargetNumOfOutputFrames)
- int [DitherNS_Process](#) (void *pHandle, const void *pInputBuffer, void *pOutputBuffer, float fTargetSampleRate, unsigned int uiNumOfInputFrames, unsigned int uiNumOfOutputFrames)
- [DnsDither_t](#) [DitherNS_GetDitherType](#) (void *pHandle)
- [DnsNoiseShaping_t](#) [DitherNS_GetNoiseShapeType](#) (void *pHandle)
- int [DitherNS_GetBitDepths](#) (void *pHandle, unsigned int *puiInputBits, unsigned int *puiOutputBits)
- float [DitherNS_GetSrcQuality](#) (void *pHandle)
- int [DitherNS_DestroyInstance](#) (void **ppHandle)

5.1.1 Define Documentation

5.1.1.1 #define _DITHERNSAPI_HEADER_INCLUDED

Definition at line 38 of file dithernsapi.h.

5.1.2 Enumeration Type Documentation

5.1.2.1 enum DnsDither_t

different dither noise types

Enumerator:

- kDnsDitherNone* no dithering
- kDnsDitherRect* dither noise has equal distribution
- kDnsDitherTriang* dither noise has triangular distribution
- kDnsDitherTriangHp* dither noise has triangular distribution and is high pass filtered (recommended!)
- kDnsNumDitherTypes*

Definition at line 53 of file dithernsapi.h.

5.1.2.2 enum DnsInputData_t

possible input data type

Enumerator:

- kDnsInpDataInt* if input data is int32
- kDnsInpDataFloat* if input data is float32
- kDnsNumInputDataTypes*

Definition at line 45 of file dithernsapi.h.

5.1.2.3 enum DnsNoiseShaping_t

different noise shaping modes, for sampling rates above 48kHz, always use DNS_NSTYPE_ULTRA

Enumerator:

- kDnsNoiseShapeNone* no noiseshaping
- kDnsNoiseShapeLow* one coefficient noiseshaping
- kDnsNoiseShapeMid* f-weighted noiseshaping (44.1 and 48kHz)
- kDnsNoiseShapeHigh* 24 coefficient noiseshaping (44.1 and 48kHz)
- kDnsNoiseShapeUltra* 32 coefficient noiseshaping (44.1 and 48kHz) or 24 coefficient noiseshaping (≥ 88.2 kHz) (recommended!)
- kDnsNumNoiseShapeTypes*

Definition at line 63 of file dithernsapi.h.

5.1.3 Function Documentation

5.1.3.1 int DitherNS_CreateInstance (void **ppHandle, DnsInputData_t InputDataType, int iInputSampleRate, int iNumOfChannels)

creates an instance of the Dither/Noiseshaping module

Parameters:

- ppHandle* : returns a pointer to the module
- InputDataType* : see DnsInputData_t
- iInputSampleRate* : the samplerate of the input signal
- iNumOfChannels* : number of channels

Returns:

- int : error code (0: no error)

5.1.3.2 int DitherNS_DestroyInstance (void ** *ppHandle*)

destroys an instance of the module

Parameters:

ppHandle : pointer to the module instance to be destroyed

Returns:

int : error code (0: no error)

5.1.3.3 int DitherNS_GetBitDepths (void * *pHandle*, unsigned int * *puiInputBits*, unsigned int * *puiOutputBits*)

returns the current word length settings for input and output

Parameters:

pHandle : a pointer to the instance

**puiInputBits* : input word length, to be written

**puiOutputBits* : output word length, to be written

Returns:

int : error code

5.1.3.4 DnsDither_t DitherNS_GetDitherType (void * *pHandle*)

returns the current dither type

Parameters:

pHandle : a pointer to the instance

Returns:

DnsDither_t : dither type

5.1.3.5 DnsNoiseShaping_t DitherNS_GetNoiseShapeType (void * *pHandle*)

returns the current noise shaping type

Parameters:

pHandle : a pointer to the instance

Returns:

DnsNoiseShaping_t : noise shaping type

5.1.3.6 int DitherNS_GetRequiredNumOfInputFrames (void * *pHandle*, float *fTargetSampleRate*, int *iTargetNumOfOutputFrames*)

returns the number of required input frames for a required number of output frames (required even if input sample rate equals output sample rate)

Parameters:

pHandle : a pointer to the instance

fTargetSampleRate : output sample rate (may change over time)

iTargetNumOfOutputFrames : number of required output frames (the same for mono or stereo)

Returns:

int : error code (0: no error)

5.1.3.7 float DitherNS_GetSrcQuality (void * *pHandle*)

returns the current sample rate quality setting

Parameters:

pHandle : a pointer to the instance

Returns:

float : sample rate conversion quality setting

5.1.3.8 int DitherNS_Initialize (void * *pHandle*)

initializes the instance (HAS TO BE CALLED BEFORE PROCESSING!!)

Parameters:

pHandle : a pointer to the instance

Returns:

int : error code (0: no error)

5.1.3.9 int DitherNS_Process (void * *pHandle*, const void * *pInputBuffer*, void * *pOutputBuffer*, float *fTargetSampleRate*, unsigned int *uiNumOfInputFrames*, unsigned int *uiNumOfOutputFrames*)

does the format conversion

Parameters:

pHandle : pointer to the module instance

pInputBuffer : pointer to input data (see DnsInputData_t)
pOutputBuffer : pointer to output data, may be the same as the input data pointer, output data is always int32
fTargetSampleRate : output sample rate (may change over time)
uiNumOfInputFrames : number of input frames (the same for mono or stereo)
uiNumOfOutputFrames : number of output frames (see DitherNS_GetRequiredNumOfInputFrames)

Returns:

int : number of output frames

5.1.3.10 int DitherNS_SetBitDepths (void * pHandle, unsigned int uiInputBits, unsigned int uiOutputBits)

sets the input and output word length (HAS TO BE CALLED BEFORE INITIALIZATION!!)

Parameters:

pHandle : a pointer to the instance
uiInputBits : word length of input in bit
uiOutputBits : wordlength of output in bit

Returns:

int : error code (0: no error)

5.1.3.11 int DitherNS_SetDitherType (void * pHandle, DnsDither_t iDitherType)

sets the type of the dither

Parameters:

pHandle : a pointer to the instance
iDitherType : dither type

Returns:

int : error code (0: no error)

5.1.3.12 int DitherNS_SetNoiseShapeType (void * pHandle, DnsNoiseShaping_t iNSType)

sets the type/order of the noise shaping

Parameters:

pHandle : a pointer to the instance

iNSType : noise shaping type

Returns:

int : error code (0: no error)

5.1.3.13 int DitherNS_SetSrcQuality (void * *pHandle*, float *fQuality*)

sets the quality level of the sample rate conversion (HAS TO BE CALLED BEFORE INITIALIZATION!!)

Parameters:

pHandle : a pointer to the instance

fQuality : value between 0 (lowest quality, high performance) and 1 (highest quality, low performance) - will be quantized internally

Returns:

int : error code (0: no error)

5.2 docugen.txt File Reference**5.2.1 Detailed Description**

source documentation main file

Definition in file [docugen.txt](#).