



## Segmentation SDK Documentation

Alexander Lerch

(c) 2006 by zplane.development

September 7, 2006

## Contents

<b>1 Segmentation SDK Documentation</b>	<b>1</b>
1.1 Introduction	1
1.2 API Documentation	1
1.2.1 Naming Conventions	1
1.2.2 Memory Allocation	1
1.2.3 Instance Handling Functions	2
1.2.4 Processing Functions	2
1.2.5 Additional Functions	3
1.2.6 Calling Conventions	4
1.3 MSVC Workspace Segmentation.dsw	6
1.3.1 File Structure	6
1.3.2 Project Structure	6
1.3.3 Project Configurations	7
1.4 Coding Style minimal overview	7
1.5 Licensing Issues	7
1.6 Graph Element descriptions	7
1.7 Support	8
<b>2 Segmentation SDK Directory Hierarchy</b>	<b>9</b>
2.1 Segmentation SDK Directories	9
<b>3 Segmentation SDK Data Structure Index</b>	<b>9</b>
3.1 Segmentation SDK Data Structures	9
<b>4 Segmentation SDK File Index</b>	<b>9</b>
4.1 Segmentation SDK File List	9
<b>5 Segmentation SDK Directory Documentation</b>	<b>9</b>
5.1 incl/ Directory Reference	9
<b>6 Segmentation SDK Data Structure Documentation</b>	<b>9</b>
6.1 SegmentationResult_t_tag Struct Reference	9
6.1.1 Detailed Description	10
6.1.2 Field Documentation	10

<b>7 Segmentation SDK File Documentation</b>	<b>11</b>
7.1 docugen.txt File Reference . . . . .	11
7.1.1 Detailed Description . . . . .	11
7.2 Segmentation_C.h File Reference . . . . .	11
7.2.1 Define Documentation . . . . .	12
7.2.2 Typedef Documentation . . . . .	12
7.2.3 Enumeration Type Documentation . . . . .	12
7.2.4 Function Documentation . . . . .	12

# 1 Segmentation SDK Documentation

## 1.1 Introduction

The Segmentation SDK provided by zplane analyzes audio input signals like radio broadcasts. The analysis result are succeeding segments classified as music and non-music. The intention is to classify an audio signal as music even if the music is only in background.

For the classification task, the algorithm extracts 55 features from the audio signal, transforms them with a matrix for dimensionality reduction, calculates the distance between the signal under test to some reference values for the classes, and finds the minimum cost path through these distances over time.

## 1.2 API Documentation

The SDK provides a C-API, which is available in the file [Segmentation\\_C.h](#). All variable types needed are either defined in this file or are standard C-types.

### 1.2.1 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.

### 1.2.2 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. Audio data buffers are allocated in interleaved format as single arrays of length [frames].

### 1.2.3 Instance Handling Functions

The following functions have to be called when using the Segmentation library:

- **Segment\_CreateInstance** (**void\*\* pphSegmentationHandle, int iSampleRate, int iNumberOfChannels**)  
Creates a new instance of for Segmentation. The parameter pphSegmentationHandle is written, the parameter iSampleRate and iNumberOfChannels specify the format of the incoming audio data.  
The function returns 0 in the case of no error.
- **Segment\_DestroyInstance** (**void\*\* pphSegmentationHandle**)  
Destroys an instance of for Segmentation. The parameter pphSegmentationHandle is set to NULL.  
The function returns 0 in the case of no error.
- **Segment\_Initialize** (**void\* phSegmentationHandle, int iOverallInputFileLengthInFrames**)  
Initializes a newly created instance. This function does mainly the internal memory allocation. The parameter phsegmentationHandle is the handle to the newly created instance, the parameter iOverallInputFileLengthInFrames indicates the length of the input data to be analyzed.  
The function returns 0 in the case of no error.

### 1.2.4 Processing Functions

- **Segment\_Process** (**void\* phSegmentationHandle, float \*pfInputBufferInterleaved, int iNumberOfFrames**)  
Processes the input data for the feature extraction. This function produces by far the highest workload of all API function. The parameter phsegmentationHandle is the handle to the newly created instance, the parameter pfInputBufferInterleaved contains succeeding blocks of interleaved audio data, with its length given in parameter iNumberOfFrames. The call of the function is obligatory.  
The function returns 0 in the case of no error.
- **Segment\_FinishProcess** (**void\* phSegmentationHandle**)  
Signals that no more input data is available. The parameter phsegmentationHandle is the handle to the newly created instance. This function has to be called after the call of Segmentation\_Process.  
The function returns 0 in the case of no error.
- **Segment\_PostProcess** (**void\* phSegmentationHandle, float fTransitionWeight = 0.8F, float fMinimumClassTimeInS = 1.0F, float fAPrioriProbabilityOfMusic = 0.5F**)  
This function does the actual classification from the extracted feature data. The parameter phsegmentationHandle is the handle to the newly created instance.

The following three function parameters influence the classification result; `fTransitionWeight` is an additional weight that makes the transition from one class to the other more costly or more easy. It has to be a non-negative value, preferably between 0 and 1.0; the higher the value is, the more difficult is the jump from one class to the other in time. `fMinimumClassTimeInS` is the minimum time a class has to be active; if the time span of a classification is below the range, it will be ignored. The time format is seconds. `fAPrioriProbabilityOfMusic` adjusts the expected probability of Music in the to be classified data. If it is set to values higher 0.5 the amount of music in the classification result will increase. Before the call of this function, it is required to call either `Segmentation_Process` or `Segment_SetIntermediateResult`.

**Important:** Please be sure that the two delivered txt-files can be read by this function.

05.01.2006 14:06 The function returns 0 in the case of no error.

- **Segment\_GetSizeOfResult** (`void* phSegmentationHandle`)  
Returns the size of the calculated result. The parameter `phsegmentationHandle` is the handle to the newly created instance. Before the call of this function, it is required to call `Segment_PostProcess`. The return value is the number of result structures of type `SegmentationResult_t`.
- **Segment\_GetResult** (`void* phSegmentationHandle`, `SegmentationResult_t *pstResult`)  
Copies the calculated result to parameter `pstResult`. The parameter `phsegmentationHandle` is the handle to the newly created instance. Note that the memory `pstResult` points to has to be allocated by the user of the SDK. It has to be at least of the size returned by function `Segment_GetSizeOfResult`.  
The function returns 0 in the case of no error.

### 1.2.5 Additional Functions

These function allow to store and retrieve the results calculated by the function `Segment_Process`. Since this is the function producing the highest workload, it could make sense in some cases, to do the features extraction in a separate task. The data is available in matrix resp. double array form.

- **Segment\_GetSizeOfIntermediateResult** (`void* phSegmentationHandle`, `int *piRows`, `int *piColumns`)  
Write the number of elements that need to be allocated in float format to parameters `*piRows` and `*piColumns`. The array to be allocated has to be of size `ppfFloatArray[iRows][iColumns]`. The parameter `phsegmentationHandle` is the handle to the newly created instance.  
The function returns 0 in the case of no error.
- **Segment\_GetIntermediateResult** (`void* phSegmentationHandle`, `float **ppfIntermediateResult`)  
Copies the calculated intermediate result to parameter `ppfIntermediateResult`, which size was allocated acc. function `Segment_GetSizeOfIntermediateResult`.

The parameter `phsegmentationHandle` is the handle to the newly created instance.

The function returns 0 in the case of no error.

- **Segment\_SetIntermediateResult** (`void* phSegmentationHandle, float **ppfIntermediateResult, int iRows, int iColumns`)

Copies the previously calculated intermediate result to the instance. The parameter `phsegmentationHandle` is the handle to the newly created instance, parameters `iRows` and `iColumns` give the dimension of `ppfIntermediateResult`, acc. to the description of `Segment_GetSizeOfIntermediateResult`. This function is the "counterpart" of `Segment_GetIntermediateResult`.

The function returns 0 in the case of no error.

- **Segment\_SetTxtFilePath** (`void* phSegmentationHandle, char *pcTxtFilePath`)

Optionally sets the directory path the the required txt files "means.txt" and "scale.txt". This function is only required if these files are not in the path.

The function returns 0 in the case of no error.

### 1.2.6 Calling Conventions

This is a Step-by-step introduction for the usage of the SDK. The complete code can be found in the example source file `SegmentationTestCLMain.cpp`.

Please note that the example source uses the open source library `libSndFile` for wav file reading and writing. The corresponding libraries `libSndFile.lib` and `libSndFile.dll` are not needed for using the [aufTAKT] SDK itself.

In the first step, a handle to the instance and the results structure have to be declared:

```
void                *pInstanceHandle    = 0;                // handle to segmentation :
SegmentationResult_t *pstResult        = 0;                // pointer of result struct
```

Then, the instance is created and initialized

```
// create class instance and initialize it
Segment_CreateInstance (&pInstanceHandle, sfInputInfo.samplerate, sfInputInfo.channels);
Segment_Initialize (pInstanceHandle, sfInputInfo.frames);
```

After that, the actual processing can be done with `Segment_Process` called with succeeding blocks of input audio data:

```
// now we can begin to process!
while (bReadNextFrame)
{
    // read data from file
    iNumFramesRead = (int) (sf_readf_float (pFInputFile, afFloatData, _BLOCKSIZE));
    //processing
    Segment_Process (pInstanceHandle, afFloatData, iNumFramesRead);
} // process loop
```

and has to be finished with the call of `Segment_FinishProcess`

```
// finish processing
Segment_FinishProcess (pInstanceHandle);
```

If, optionally, the intermediate result of the processing should be stored for later usage, one can request the size of the information to be stored and write them to a file:

```
// get required buffer size
Segment_GetSizeOfIntermediateResult (pInstanceHandle, &aiDimensions[0], &aiDimensions[1]);

// alloc memory
ppfFeatures = new float* [aiDimensions[0]];
for ( i = 0; i < aiDimensions[0]; i++)
    ppfFeatures[i] = new float [aiDimensions[1]];

// get result
Segment_GetIntermediateResult (pInstanceHandle, ppfFeatures);

// open output file
pFOutputFile = fopen (acOutputFileName, "wt");

// write result to file
for (i = 0; i < aiDimensions[0]; i++)
{
    for (j = 0; j < aiDimensions[1]; j++)
        fprintf (pFOutputFile, "%1.10e\t", ppfFeatures[i][j]);
    fprintf (pFOutputFile, "\n");
}

// close output file
fclose (pFOutputFile);
```

Finally, the results can be computed by calling the function `Segment_PostProcess`

```
// now do the classification based on the extracte features
Segment_PostProcess (pInstanceHandle); //, .0F, 1.0F, .85F, .1F);
```

The computed result can then be requested with the following calls:

```
// get size of result
iSizeOfResult = Segment_GetSizeOfResult (pInstanceHandle);

// alloc required memory
pstResult = new SegmentationResult_t [iSizeOfResult];

// get result
Segment_GetResult (pInstanceHandle, pstResult);
```

Finally, the instance can be destroyed by

```
// destroy instance
Segment_DestroyInstance (&pInstanceHandle);
```

The above code snippets demonstrated the basic functionality of the Segmentation library. Most of the additional functions can be used similar to the given code examples. The exact functionality of the functions is described above.

## 1.3 MSVC Workspace Segmentation.dsw

### 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 MS VisualC++-Workspace (.dsw) and all Projectfiles (.dsp) 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** If include files are project intern, they are in the source directory **./src** of the project itself. If include files are to be included by other projects they can be found in **./src/include**. If a SDK-like library/DLL is built for which a header is needed, such a header can be found in **./inc**.

**1.3.1.5 Resource Files** The resource files 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 as well as the distributable Dynamic Link Libraries 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**.

### 1.3.2 Project Structure

The project structure is as following:

- **libSndFile**: library for audio file parsing. Note that libSndfile is an open source project under the license LGPL and can only be used if the licensing restrictions are met.

The project output is a Dynamic Link Library (DLL).

- **SegmentationTestCL**: example test project that links the SDK-DLL. The project output is an executable binary (EXE).

### 1.3.3 Project Configurations

For all projects included in the workspace, the default configurations Win32 Release and Win32 Debug are available.

## 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
char:	c

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

## 1.5 Licensing Issues

The SDK can be used under the terms of the license agreement. Note that the library `libSndfile`, included in the test project, can only be used under the restrictions of the LGPL (GNU Lesser General Public License).

## 1.6 Graph Element descriptions

This is an excerpt from the doxygen documentation:

*The elements in the class diagrams in HTML and RTF have the following meaning:*

- *A yellow box indicates a class. A box can have a little marker in the lower right corner to indicate that the class contains base classes that are hidden. For the class diagrams the maximum tree width is currently 8 elements. If a tree is wider some nodes will be hidden. If the box is filled with a dashed pattern the inheritance relation is virtual.*
- *A white box indicates that the documentation of the class is currently shown.*
- *A grey box indicates an undocumented class.*
- *A solid dark blue arrow indicates public inheritance.*
- *A dashed dark green arrow indicates protected inheritance.*

- A dotted dark green arrow indicates private inheritance.

The elements in the class diagram in PDF have the following meaning:

- A white box indicates a class. A marker in the lower right corner of the box indicates that the class has base classes that are hidden. If the box has a dashed border this indicates virtual inheritance.
- A solid arrow indicates public inheritance.
- A dashed arrow indicates protected inheritance.
- A dotted arrow indicates private inheritance.

The elements in the graphs (...) have the following meaning:

- A white box indicates a class or struct or file.
- A box with a red border indicates a node that has more arrows than are shown! In other words: the graph is truncated with respect to this node. The reason why a graph is sometimes truncated is to prevent images from becoming too large. For the graphs generated with dot doxygen tries to limit the width of the resulting image to 1024 pixels.
- A black box indicates that the class' documentation is currently shown.
- A dark blue arrow indicates an include relation (for the include dependency graph) or public inheritance (for the other graphs).
- A dark green arrow indicates protected inheritance.
- A dark red arrow indicates private inheritance.
- A purple dashed arrow indicated a "usage" relation, the edge of the arrow is labled with the variable(s) responsible for the relation. Class A uses class B, if class A has a member variable *m* of type C, where B is a subtype of C (e.g. C could be B, B\*, T<B>\* ).

## 1.7 Support

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

[zplane.development](http://zplane.development)

katzbachstr.21

d-10965 berlin

germany

fon: +49.30.854 09 15.0

fax: +49.30.854 09 15.5

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

## 2 Segmentation SDK Directory Hierarchy

### 2.1 Segmentation SDK Directories

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

[incl](#)

9

## 3 Segmentation SDK Data Structure Index

### 3.1 Segmentation SDK Data Structures

Here are the data structures with brief descriptions:

[SegmentationResult\\_t\\_tag](#)

9

## 4 Segmentation SDK File Index

### 4.1 Segmentation SDK File List

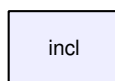
Here is a list of all files with brief descriptions:

[Segmentation\\_C.h](#)

11

## 5 Segmentation SDK Directory Documentation

### 5.1 incl/ Directory Reference



#### Files

- file [Segmentation\\_C.h](#)

## 6 Segmentation SDK Data Structure Documentation

### 6.1 SegmentationResult\_t\_tag Struct Reference

```
#include <Segmentation_C.h>
```

### 6.1.1 Detailed Description

declaration of result structure

Definition at line 22 of file Segmentation\_C.h.

#### Data Fields

- float **iStartTimeInS**  
*segment start*
- float **iStopTimeInS**  
*segment stop*
- **SegmentationClasses\_t eEstimatedClass**  
*class*
- float **fEstimationReliability**  
*estimate of result reliability*

### 6.1.2 Field Documentation

#### 6.1.2.1 **SegmentationClasses\_t SegmentationResult\_t\_tag::eEstimatedClass**

class

Definition at line 27 of file Segmentation\_C.h.

#### 6.1.2.2 float **SegmentationResult\_t\_tag::fEstimationReliability**

estimate of result reliability

Definition at line 29 of file Segmentation\_C.h.

#### 6.1.2.3 float **SegmentationResult\_t\_tag::iStartTimeInS**

segment start

Definition at line 24 of file Segmentation\_C.h.

#### 6.1.2.4 float **SegmentationResult\_t\_tag::iStopTimeInS**

segment stop

Definition at line 24 of file Segmentation\_C.h.

The documentation for this struct was generated from the following file:

- **Segmentation\_C.h**

## 7 Segmentation SDK File Documentation

### 7.1 docugen.txt File Reference

#### 7.1.1 Detailed Description

source documentation main file

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

### 7.2 Segmentation\_C.h File Reference

#### Data Structures

- struct [SegmentationResult\\_t\\_tag](#)

#### Defines

- #define [\\_\\_libSegmentationC\\_HEADER\\_INCLUDED\\_\\_](#)

#### Typedefs

- typedef enum [SegmentationClasses\\_t\\_tag](#) [SegmentationClasses\\_t](#)
- typedef [SegmentationResult\\_t\\_tag](#) [SegmentationResult\\_t](#)

#### Enumerations

- enum [SegmentationClasses\\_t\\_tag](#) { [kDoesNotContainMusic](#) = 0, [kContainsMusic](#) = 1, [kNumOfSegmentationClasses](#) }

#### Functions

- int [Segment\\_CreateInstance](#) (void \*\*pphSegmentationHandle, int iSampleRate, int iNumberOfChannels)
- int [Segment\\_DestroyInstance](#) (void \*\*pphSegmentationHandle)
- int [Segment\\_Initialize](#) (void \*phSegmentationHandle, int iOverallInputFileLengthInFrames)
- int [Segment\\_PreProcess](#) (void \*phSegmentationHandle, float \*pfInputBufferInterleaved, int iNumberOfFrames)
- int [Segment\\_Process](#) (void \*phSegmentationHandle, float \*pfInputBufferInterleaved, int iNumberOfFrames)
- int [Segment\\_FinishProcess](#) (void \*phSegmentationHandle)
- int [Segment\\_SetTxtFilePath](#) (void \*phSegmentationHandle, char \*pcTxtFilePath)

- int **Segment\_PostProcess** (void \*phSegmentationHandle, float fTransitionWeight=0.6F, float fMinimumClassTimeInS=1.0F, float fAPrioriProbabilityOfMusic=0.5F, float fMinimumEnergy=0.1F)
- int **Segment\_GetSizeOfResult** (void \*phSegmentationHandle)
- int **Segment\_GetResult** (void \*phSegmentationHandle, **SegmentationResult\_t** \*pstResult)
- int **Segment\_GetSizeOfIntermediateResult** (void \*phSegmentationHandle, int \*piRows, int \*piColumns)
- int **Segment\_GetIntermediateResult** (void \*phSegmentationHandle, float \*\*ppfIntermediateResult)
- int **Segment\_SetIntermediateResult** (void \*phSegmentationHandle, float \*\*ppfIntermediateResult, int iRows, int iColumns)

### 7.2.1 Define Documentation

#### 7.2.1.1 #define **\_\_libSegmentationC\_HEADER\_INCLUDED\_\_**

Definition at line 2 of file Segmentation\_C.h.

### 7.2.2 Typedef Documentation

#### 7.2.2.1 typedef enum **SegmentationClasses\_t\_tag SegmentationClasses\_t**

list of the valid classes

#### 7.2.2.2 typedef struct **SegmentationResult\_t\_tag SegmentationResult\_t**

declaration of result structure

### 7.2.3 Enumeration Type Documentation

#### 7.2.3.1 enum **SegmentationClasses\_t\_tag**

list of the valid classes

#### Enumerator:

- kDoesNotContainMusic* there is no music
- kContainsMusic* there was music detected
- kNumOfSegmentationClasses*

Definition at line 12 of file Segmentation\_C.h.

### 7.2.4 Function Documentation

#### 7.2.4.1 int **Segment\_CreateInstance** (void \*\* *pphSegmentationHandle*, int *iSampleRate*, int *iNumberOfChannels*)

Creates a new instance of the segmentation lib

**Parameters:**

*pphSegmentationHandle* : handle to the new instance

*iSampleRate* : sample rate of audio file

*iNumberOfChannels* : number of audio channels

**Returns:**

int : 0 if no error

**7.2.4.2 int Segment\_DestroyInstance (void \*\* pphSegmentationHandle)**

destroys a previously created instance of the segmentation lib

**Parameters:**

*pphSegmentationHandle* : handle to the instance

**Returns:**

int : 0 if no error

**7.2.4.3 int Segment\_FinishProcess (void \* phSegmentationHandle)**

called to signal there is no more audio data available

**Parameters:**

*phSegmentationHandle* : handle to the instance

**Returns:**

int : 0 if no error

**7.2.4.4 int Segment\_GetIntermediateResult (void \* phSegmentationHandle, float \*\* ppfIntermediateResult)**

returns the internal feature data

**Parameters:**

*phSegmentationHandle* : handle to the instance

*\*\*ppfIntermediateResult* : two-dimensional matrix with the size from GetSize-OfIntermediateResult

**Returns:**

int : 0 if no error

**7.2.4.5 int Segment\_GetResult (void \* *phSegmentationHandle*, **Segmentation-Result\_t** \* *pstResult*)**

copies the result to *pstResult*

**Parameters:**

*phSegmentationHandle* : handle to the instance

\**pstResult* : pointer to allocated memory where the result can be copied into

**Returns:**

int : 0 if no error

**7.2.4.6 int Segment\_GetSizeOfIntermediateResult (void \* *phSegmentationHandle*, int \* *piRows*, int \* *piColumns*)**

returns the size of intermediate result after *FinishProcess* (if intermediate results should be stored for later usage)

**Parameters:**

*phSegmentationHandle* : handle to the instance

\**piRows* : number of rows of result matrix (to be written)

\**piColumns* : number of columns of result matrix (to be written)

**Returns:**

int : 0 if no error

**7.2.4.7 int Segment\_GetSizeOfResult (void \* *phSegmentationHandle*)**

returns the size of the result vector (as number of structs)

**Parameters:**

*phSegmentationHandle* : handle to the instance

**Returns:**

int : 0 if no error

**7.2.4.8 int Segment\_Initialize (void \* *phSegmentationHandle*, int *iOverallInput-FileLengthInFrames*)**

initializes an instance of the segmentation lib

**Parameters:**

*phSegmentationHandle* : handle to the instance

*iOverallInputFileLengthInFrames* : number of audio frames

**Returns:**

int : 0 if no error

**7.2.4.9** int Segment\_PostProcess (void \* *phSegmentationHandle*, float *fTransitionWeight* = 0.6F, float *fMinimumClassTimeInS* = 1.0F, float *fAPrioriProbabilityOfMusic* = 0.5F, float *fMinimumEnergy* = 0.1F)

postprocesses the data and calculates result

**Parameters:**

*phSegmentationHandle* : handle to the instance

*fTransitionWeight* : additional cost for the transition between classes (the higher the less jumps between classes), must be positive

*fMinimumClassTimeInS* : minimum active time of a class

*fAPrioriProbabilityOfMusic* : if class 1 and 2 are not equally probably, change this (between 0...1)

*fMinimumEnergy* : if a processing frame does contain an sqrt(rms)-energy below this threshold, this frame is not regarded as able to contain music (between 0...1)

**Returns:**

int : 0 if no error

**7.2.4.10** int Segment\_PreProcess (void \* *phSegmentationHandle*, float \* *pfInputBufferInterleaved*, int *iNumberOfFrames*)

preprocessing loop of the audio data

**Parameters:**

*phSegmentationHandle* : handle to the instance

*pfInputBufferInterleaved* : audio input data in pcm interleaved format

*iNumberOfFrames* : number of audio frames in buffer

**Returns:**

int : 0 if no error

**7.2.4.11 int Segment\_Process (void \* *phSegmentationHandle*, float \* *pfInputBufferInterleaved*, int *iNumberOfFrames*)**

processes the audio data

**Parameters:**

*phSegmentationHandle* : handle to the instance

*pfInputBufferInterleaved* : audio input data in pcm interleaved format

*iNumberOfFrames* : number of audio frames in buffer

**Returns:**

int : 0 if no error

**7.2.4.12 int Segment\_SetIntermediateResult (void \* *phSegmentationHandle*, float \*\* *ppfIntermediateResult*, int *iRows*, int *iColumns*)**

sets the internal feature data (no Segment\_Process is required then)

**Parameters:**

*phSegmentationHandle* : handle to the instance

*ppfIntermediateResult* : two dimensional matrix with intermediate results

*iRows* : number of rows of matrix

*iColumns* : number of columns of matrix

**Returns:**

int : 0 if no error

**7.2.4.13 int Segment\_SetTxtFilePath (void \* *phSegmentationHandle*, char \* *pcTxtFilePath*)**

set path to input txt files (means.txt and scale.txt)

**Parameters:**

*phSegmentationHandle* : handle to the instance

*pcTxtFilePath* : path

**Returns:**

int : 0 if no error