



Training SDK Documentation

Alexander Lerch

(c) 2006 by zplane.development

September 7, 2006

Contents

1 Training SDK Documentation	1
1.1 Introduction	1
1.2 Training Data	1
1.3 Internal API Documentation	2
1.3.1 Naming Conventions	2
1.3.2 Memory Allocation	2
1.3.3 Instance Handling Functions	2
1.3.4 Processing Functions	3
1.3.5 Additional Functions	3
1.4 CTraining_If Documentation	4
1.4.1 Calling Conventions	5
1.5 MSVC Workspace Training.dsw	6
1.5.1 File Structure	6
1.5.2 Project Structure	6
1.5.3 Project Configurations	7
1.6 Coding Style minimal overview	7
1.7 Licensing Issues	7
1.8 Graph Element descriptions	7
1.9 Support	9
2 Training SDK Directory Hierarchy	9
2.1 Training SDK Directories	9
3 Training SDK Namespace Index	9
3.1 Training SDK Namespace List	9
4 Training SDK Data Structure Index	9
4.1 Training SDK Data Structures	9
5 Training SDK File Index	10
5.1 Training SDK File List	10
6 Training SDK Directory Documentation	10
6.1 incl/ Directory Reference	10

6.2	src/ Directory Reference	10
6.3	src/TrainingCL/ Directory Reference	11
7	Training SDK Namespace Documentation	11
7.1	std Namespace Reference	11
8	Training SDK Data Structure Documentation	11
8.1	CFileList Class Reference	11
8.1.1	Detailed Description	12
8.1.2	Constructor & Destructor Documentation	12
8.1.3	Member Function Documentation	13
8.1.4	Field Documentation	15
8.2	ClassifierInfo_t_tag Struct Reference	16
8.2.1	Detailed Description	16
8.2.2	Field Documentation	16
8.3	CTraining_If Class Reference	17
8.3.1	Detailed Description	17
8.3.2	Member Enumeration Documentation	19
8.3.3	Constructor & Destructor Documentation	20
8.3.4	Member Function Documentation	21
8.3.5	Field Documentation	29
9	Training SDK File Documentation	31
9.1	FileList.cpp File Reference	31
9.1.1	Detailed Description	31
9.1.2	Define Documentation	31
9.2	HelperFunctions.cpp File Reference	31
9.2.1	Detailed Description	31
9.2.2	Function Documentation	32
9.3	training.txt File Reference	34
9.4	Training_C.h File Reference	34
9.4.1	Define Documentation	36
9.4.2	Typedef Documentation	36
9.4.3	Enumeration Type Documentation	36
9.4.4	Function Documentation	38

9.5	Training_If.cpp File Reference	40
9.5.1	Detailed Description	40
9.5.2	Define Documentation	41
9.5.3	Variable Documentation	42
9.6	Training_If.h File Reference	42
9.6.1	Detailed Description	42
9.6.2	Define Documentation	43
9.7	TrainingMain.cpp File Reference	43
9.7.1	Detailed Description	43
9.7.2	Define Documentation	44
9.7.3	Enumeration Type Documentation	44
9.7.4	Function Documentation	45

1 Training SDK Documentation

1.1 Introduction

The Training SDK provided by zplane generates a customized parametrization for the classification engine used by the zplane Segmentation SDK. It allows to use a custom-defined training set to allow high user optimized class definition.

The Training SDK consists of three parts: source code of an example command line application, source code for the extraction and management of the training data (which makes use of the Segmentation.dll as being delivered with the Segmentation SDK), and the Training-library itself with a C-interface that will be referred to as **internal API**.

1.2 Training Data

To ensure best functionality, the training database should have the following properties:

- there should be a significant amount of audio data for training (at least 3-5 hours)
- the training data should have similar properties as the data that is used for later classification (test data) with respect to:
 - quality (sample rate, bit depth, background noise, level, ...)
 - content (speakers (male + female), music, ...)
- the audio quality of the training (and naturally of the test set as well) should be as high as possible
- the amount of training data should preferably be nearly the same for both classes

- the classes should be selected in a way that they contain as similar training files as possible
- the classes should contain a not negligible amount of (subjectively) very clear-to-classify data

1.3 Internal API Documentation

The SDK's training library provides a C-API, which is available in the file [Training_C.h](#). All variable types required are either defined in this file or are standard C-types.

1.3.1 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.

The term **feature** refers to a kind of metainformation that is extracted from the audio data. In this context, several features are extracted, where every feature consists of a floating point value for each time frame.

1.3.2 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].

1.3.3 Instance Handling Functions

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

- **Train_CreateInstance** (**void** pphTrainingHandle, int iSampleRate, int iNumberOfChannels**)
Creates a new instance for Training. The parameter pphTrainingHandle 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.
- **Train_DestroyInstance** (**void** pphTrainingHandle**)
Destroys an instance of for Training. The parameter pphTrainingHandle is set to NULL.
The function returns 0 in the case of no error.

1.3.4 Processing Functions

- **Train_Process** (void* phTrainingHandle)

Processes the generation of the output data. This function produces the highest workload of all API functions. The parameter phTrainingHandle is the handle to the newly created instance. The function Train_Process requires feature data for training; therefore the preceding successful call of function **Train_AppendFeatureData** is mandatory. The call of the function is mandatory.

Note that the internal data is modified by this function. It cannot be called twice without setting the complete new feature data.

The function returns 0 in the case of no error.

- **Train_GetResultDimension** (void* phTrainingHandle, TrainingResults_t eResultIdx, int *piResultDimensions)

Returns the size of the calculated result. The parameter phTrainingHandle is the handle to the newly created instance, the parameter eResultIdx is the index of the result (currently, two results are possible), and the matrix dimensions of the result are written to the memory where parameter piResultDimensions points to (piResultDimensions[0] equals the number of rows, piResultDimensions[1] equals the number of columns. Before the call of this function, it is required to call **Train_Process**.

The function returns 0 in the case of no error.

- **Train_GetResult** (void* phTrainingHandle, TrainingResults_t eResultIdx, float **ppfResult, const int *piResultDimensions)

Copies the calculated result to array ppfResult. The parameter phTrainingHandle is the handle to the newly created instance. Note that the memory ppfResult points to has to be allocated by the user of the SDK. It has to be at least of the size returned by function **Train_GetResultDimension**. The content of parameter piResultDimension has to be identical to the values given back by the function **Train_GetResultDimension**.

The function returns 0 in the case of no error.

1.3.5 Additional Functions

To provide information about the training data and to make sure that its properties are correct, this function may be called before **Train_Process**

- **Train_GetClassifierInfo** (void* phTrainingHandle, ClassifierInfo_t *pInfo)

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

The function returns 0 in the case of no error.

1.4 CTraining_If Documentation

The class `CTraining_If` that is delivered with its complete sources is on the one hand a simple wrapper for the Segmentation and Training libraries, on the other hand manages some additional functionality like directory parsing, output file writing, etc. Its interface consists of the following methods:

- **CTraining_If::CreateInstance** (`CTraining_If*& pCTraining_If`)
Creates a new instance of `CTraining_If`. The handle of the new instance is written to parameter `pCTraining_If`.
The function returns 0 in the case of no error.
- **CTraining_If::DestroyInstance** (`CTraining_If*& pCTraining_If`)
Destroys an instance of `CTraining_If`. The parameter `pCTraining_If` is set to NULL.
The function returns 0 in the case of no error.
- **CTraining_If::SetParamDirectoryPaths** (`string strPathToMusicDir, string strPathToNonMusicDir`)
Sets the path to the two distinct directories containing audio files for each class respectively.
The function returns 0 in the case of no error.
- **CTraining_If::SetParamAudioFileExtension** (`string strAudioFileExtension`)
Sets the extension of the audio files that are taken into account (default ".wav"). Note the the audio file IO library has to support the file format, otherwise the feature data cannot be extracted.
The function returns 0 in the case of no error.
- **CTraining_If::SetParamOutFilePath** (`string strPath`)
Sets the directory path of the generated output files used for the classification process (default working directory).
The function returns 0 in the case of no error.
- **CTraining_If::CalculateFeatures** ()
After all directories are set, we are ready to calculated the features for the data and store them internally.
The function returns 0 in the case of no error.
- **CTraining_If::Train** ()
After all computed all features, we are able to complete the actual training process.
The function returns 0 in the case of no error.
- **CTraining_If::WriteTrainingResults** ()
After successful training the output files can be written for later usage.
The function returns 0 in the case of no error.

- **CTraining_If::SetTrainingData** (float **ppfFeatures, TrainingClasses_t eClass, int iNumOfObservations, int iNumOfFeatures)

The use of this function is usually **not** required. It allows to feed the training instance with feature data extracted "by hand". Parameter ppfFeatures contains the feature data as a matrix with dimensions [iNumOfFeatures]x[iNumOfObservations]. Parameter eClass is the class index of the class this data represents.

The function returns 0 in the case of no error.

1.4.1 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 [TrainingMain.cpp](#).

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

```
CTraining_If *pCTrainingInstance = 0;          //!< instance handle for training
```

Then, the instance is created

```
CTraining_If::CreateInstance (pCTrainingInstance);
```

After that, have to set the input and output directories:

```
// set input and output directories
pCTrainingInstance->SetParamDirectoryPaths (strDirectories[kDirMusic], strDirectories[kDirNoMus
pCTrainingInstance->SetParamOutFilePath (strDirectories[kDirOutfile]);
cout << "Parameters set...\n";
```

and can calculate the features.

```
// do feature calculation
pCTrainingInstance->CalculateFeatures ();
cout << "Features calculated...\n";
```

Finally, we are ready to train...

```
// after we have our training data available, we are able to train
pCTrainingInstance->Train ();
cout << "Training done...\n";
```

and can write the results (txt files) to the previously defined output directory:

```
// now write the data to the txt files
pCTrainingInstance->WriteTrainingResults ();
cout << "Output files written...\n";
```

In the end, the instance can be destroyed by

```
// destroy instance
CTraining_If::DestroyInstance (pCTrainingInstance);
cout << "Destroyed Training Instance...\n";
```

The above code snippets demonstrated the basic functionality of the `CTraining_If` interface. The exact functionality of the functions is described above.

1.5 MSVC Workspace Training.dsw

1.5.1 File Structure

1.5.1.1 Documentation This documentation and all other documentation can be found in the directory `./doc`.

1.5.1.2 Project Files The MS VisualC++-Workspace (`.sln`) and all Projectfiles (`.vcproj`) can be found in the directory `./build` and its subfolders, where the subfolders names correspond to the project names.

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

1.5.1.4 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`.

1.5.1.5 Resource Files The resource files can be found in the subdirectory `/res` of the corresponding build-directory.

1.5.1.6 Library Files The directory `./lib` is for used and built libraries.

1.5.1.7 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`.

1.5.1.8 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`.

1.5.2 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).

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

1.5.3 Project Configurations

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

1.6 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.

1.7 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).

1.8 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*).*

1.9 Support

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

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

2 Training SDK Directory Hierarchy

2.1 Training SDK Directories

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

incl	10
src	10
TrainingCL	11

3 Training SDK Namespace Index

3.1 Training SDK Namespace List

Here is a list of all namespaces with brief descriptions:

std	11
------------	-----------

4 Training SDK Data Structure Index

4.1 Training SDK Data Structures

Here are the data structures with brief descriptions:

CFileList (Used internally by CTraining_If to organize file names)	11
ClassifierInfo_t_tag (Info structure on the training data)	16

CTraining_If (This class provides an interface and some additional functionality in the context of the for the training library) 17

5 Training SDK File Index

5.1 Training SDK File List

Here is a list of all files with brief descriptions:

FileList.cpp (Implementation of the **CFileList** class) 31

HelperFunctions.cpp (Helper functions) 31

Training_C.h 34

Training_If.cpp (Implementation of the **CTraining_If** class) 40

Training_If.h (Interface of the **CTraining_If** class) 42

TrainingMain.cpp (Implementation of the training command line application) 43

6 Training SDK Directory Documentation

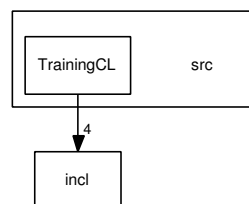
6.1 incl/ Directory Reference



Files

- file **Training_C.h**

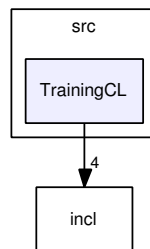
6.2 src/ Directory Reference



Directories

- directory [TrainingCL](#)

6.3 src/TrainingCL/ Directory Reference



Files

- file [FileList.cpp](#)
implementation of the `CFileList` class.
- file [HelperFunctions.cpp](#)
helper functions
- file [Training_If.cpp](#)
implementation of the `CTraining_If` class.
- file [Training_If.h](#)
interface of the `CTraining_If` class.
- file [TrainingMain.cpp](#)
implementation of the training command line application.

7 Training SDK Namespace Documentation

7.1 std Namespace Reference

8 Training SDK Data Structure Documentation

8.1 CFileList Class Reference

```
#include <Training_If.h>
```

8.1.1 Detailed Description

used internally by `CTraining_If` to organize file names

organizes a list of filenames

Definition at line 265 of file `Training_If.h`.

Public Member Functions

- `CFileList ()`
- virtual `~CFileList ()`
- int `GetNumOfEntries ()`
returns the number of file name entries in the list
- `TrainingError_t Add2List (string strFileName)`
add new file name entry to the end of the list
- `TrainingError_t GetEntry (string &strFileName, int iIndex)`
retrieves one file name/entry from the list
- `TrainingError_t Reset ()`
removes all entries from the list

Private Attributes

- int `m_iNumOfEntries`
number of entries
- int `m_iMaxNumOfEntries`
current available memory for entries
- string * `m_pstrFilePaths`
list of entries

8.1.2 Constructor & Destructor Documentation

8.1.2.1 CFileList::CFileList ()

Definition at line 63 of file `FileList.cpp`.

References `kDefaultMaxNumOfEntries`, `m_iMaxNumOfEntries`, `m_pstrFilePaths`, and `Reset()`.

```
64 {  
65     m_iMaxNumOfEntries = kDefaultMaxNumOfEntries;  
66 }
```

```
67 // alloc list with default length
68 m_pstrFilePaths = new string [m_iMaxNumOfEntries];
69 for (int i = 0; i < m_iMaxNumOfEntries; i++)
70     m_pstrFilePaths[i].erase ();
71
72 this->Reset ();
73
74 }
```

8.1.2.2 CFileList::~CFileList () [virtual]

Definition at line 77 of file FileList.cpp.

References m_pstrFilePaths.

```
78 {
79     // delete allocated memory
80     delete [] m_pstrFilePaths;
81 }
```

8.1.3 Member Function Documentation

8.1.3.1 TrainingError_t CFileList::Add2List (string strFileName)

add new file name entry to the end of the list

Parameters:

strFileName file name to add

Returns:

0 if no error

Definition at line 114 of file FileList.cpp.

References kTrainMemoryAllocError, kTrainNoError, m_iMaxNumOfEntries, m_iNumOfEntries, and m_pstrFilePaths.

Referenced by CTraining_If::ParseDirectory().

```
115 {
116     // check if list is full or not, if yes, realloc memory
117     if (m_iNumOfEntries == m_iMaxNumOfEntries)
118     {
119         int i;
120         string *pstrTmp = 0;
121
122         // realloc memory
123         pstrTmp = new string [m_iMaxNumOfEntries<<1];
124
125         if (!pstrTmp)
126         {
127             m_iMaxNumOfEntries >>= 1;
128             return kTrainMemoryAllocError;
129         }
130     }
```

```
130
131     for (i = 0; i < m_iMaxNumOfEntries; i++)
132         pstrTmp[i].assign (m_pstrFilePaths[i]);
133
134     for (i = m_iMaxNumOfEntries; i < (m_iMaxNumOfEntries<<1); i++)
135         pstrTmp[i].erase ();
136
137     delete [] m_pstrFilePaths;
138
139     m_pstrFilePaths      = pstrTmp;
140     m_iMaxNumOfEntries  <<= 1;
141 }
142
143 // add the new entry to the end of the list and increment number of list entries
144 m_pstrFilePaths[m_iNumOfEntries].assign (strFileName);
145
146 m_iNumOfEntries++;
147
148 return kTrainNoError;
149 }
```

8.1.3.2 **TrainingError_t** CFileList::GetEntry (string & *strFileName*, int *iIndex*)

retrieves one file name/entry from the list

Parameters:

strFileName the retrieved file name is written to this parameter

iIndex index of the entry to retrieve, must be lower than GetNumOfEntries ()

Returns:

0 if no error

See also:

[GetNumOfEntries](#)

Definition at line 89 of file FileList.cpp.

References `kTrainInvalidArgumentError`, `kTrainNoError`, `m_iNumOfEntries`, and `m_pstrFilePaths`.

Referenced by `CTraining_If::CalculateFeatures()`.

```
90 {
91     if (iIndex >= m_iNumOfEntries)
92         return kTrainInvalidArgumentError;
93
94     strFileName.assign (m_pstrFilePaths[iIndex]);
95
96     return kTrainNoError;
97 }
```

8.1.3.3 int CFileList::GetNumOfEntries ()

returns the number of file name entries in the list

Returns:

returns the number of file name entries in the list

Definition at line 84 of file FileList.cpp.

References `m_iNumOfEntries`.

```
85 {
86     return m_iNumOfEntries;
87 }
```

8.1.3.4 TrainingError_t CFileList::Reset ()

removes all entries from the list

Returns:

0 if no error

Definition at line 100 of file FileList.cpp.

References `kTrainNoError`, `m_iMaxNumOfEntries`, `m_iNumOfEntries`, and `m_pstrFilePaths`.

Referenced by `CFileList()`.

```
101 {
102     int i;
103
104     for (i = 0; i < m_iMaxNumOfEntries; i++)
105         m_pstrFilePaths[i].erase ();
106
107     m_iNumOfEntries = 0;
108
109     return kTrainNoError;
110
111 }
```

8.1.4 Field Documentation

8.1.4.1 int CFileList::m_iMaxNumOfEntries [private]

current available memory for entries

Definition at line 321 of file Training_If.h.

Referenced by `Add2List()`, `CFileList()`, and `Reset()`.

8.1.4.2 int CFileList::m_iNumOfEntries [private]

number of entries

Definition at line 321 of file Training_If.h.

Referenced by Add2List(), GetEntry(), GetNumOfEntries(), and Reset().

8.1.4.3 string* CFileList::m_pstrFilePaths [private]

list of entries

Definition at line 323 of file Training_If.h.

Referenced by Add2List(), CFileList(), GetEntry(), Reset(), and ~CFileList().

The documentation for this class was generated from the following files:

- [Training_If.h](#)
- [FileList.cpp](#)

8.2 ClassifierInfo_t_tag Struct Reference

```
#include <Training_C.h>
```

8.2.1 Detailed Description

info structure on the training data

Definition at line 63 of file Training_C.h.

Data Fields

- float **afPrior** [kNumOfTrainingClasses]
probability of class in the training set
- int **iNumOfFeatures**
number of features
- int **iOverallNumOfObservations**
number of observations for all classes

8.2.2 Field Documentation**8.2.2.1 float ClassifierInfo_t_tag::afPrior**[kNumOfTrainingClasses]

probability of class in the training set

Definition at line 65 of file Training_C.h.

8.2.2.2 int ClassifierInfo_t_tag::iNumOfFeatures

number of features

Definition at line 66 of file Training_C.h.

8.2.2.3 int ClassifierInfo_t_tag::iOverallNumOfObservations

number of observations for all classes

Definition at line 66 of file Training_C.h.

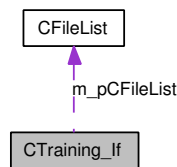
The documentation for this struct was generated from the following file:

- [Training_C.h](#)

8.3 CTraining_If Class Reference

```
#include <Training_If.h>
```

Collaboration diagram for CTraining_If:

**8.3.1 Detailed Description**

this class provides an interface and some additional functionality in the context of the for the training library

See also:

[Training_C.h](#)

Definition at line 76 of file Training_If.h.

Public Member Functions

- [CTraining_If \(\)](#)
- [virtual ~CTraining_If \(\)](#)
- [TrainingError_t SetParamDirectoryPaths](#) (string strPathToMusicDir, string strPathToNonMusicDir)
sets the directory paths to the audio files per class used for training
- [TrainingError_t SetParamAudioFileExtension](#) (string strAudioFileExtension)

sets the extension of the searched audio files (default: .wav)

- **TrainingError_t SetParamOutFilePath** (string strPath)
sets the path where the output files will be written (default: working directory)
- **TrainingError_t SetTrainingData** (float **ppfFeatures, TrainingClasses_t eClass, int iNumOfObservations, int iNumOfFeatures)
set new classifier training data
- **TrainingError_t CalculateFeatures** ()
calculate all features for the files in the previously defined directories
- **TrainingError_t Train** ()
after the calculation of features and setting all training data, finally do the training itself
- **TrainingError_t WriteTrainingResults** ()
retrieves the training results and writes them to the output files

Static Public Member Functions

- static **TrainingError_t CreateInstance** (CTraining_If *&pCTraining_If)
creates a new instance of CTraining_If
- static **TrainingError_t DestroyInstance** (CTraining_If *&pCTraining_If)
destroys an instance of CTraining_If

Private Types

- **kFileMeans**
- **kFileScale**
- **kNumOfOutFiles**
- enum **OutFiles_t** { **kFileMeans**, **kFileScale**, **kNumOfOutFiles** }
this is an internal define for the output files to be written

Private Member Functions

- **TrainingError_t ParseDirectory** (string strPath, CFileList *pCFileList)
parses a complete directory for files with a special extension and adds the file names to a list
- **TrainingError_t CalculateFeaturesAndAppendData4Training** (string strFileName, TrainingClasses_t eClass)
calculates the features for a file and appends data to training set for the given class

Private Attributes

- string **m_strDirectoryPath** [kNumOfTrainingClasses]
class directory paths
- string **m_strFileExtension**
audio file extension to parse for
- string **m_strOutFileNames** [kNumOfOutFiles]
output file names
- **CFileList * m_pCFileList** [kNumOfTrainingClasses]
list of all audio files per class directory
- float ** **m_ppfTmpFeatureMatrix**
internal memory for feature data
- int **m_aiMaxDimensionsOfFeatureMatrix** [2]
size of m_ppfTmpFeatureMatrix
- void * **m_pCTrainingInstance**
training instance

8.3.2 Member Enumeration Documentation

8.3.2.1 enum **CTraining_If::OutFiles_t** [private]

this is an internal define for the output files to be written

Enumerator:

kFileMeans

kFileScale

kNumOfOutFiles

Definition at line 208 of file Training_If.h.

```
209     {
210         kFileMeans,
211         kFileScale,
212
213         kNumOfOutFiles
214     };
```

8.3.3 Constructor & Destructor Documentation

8.3.3.1 CTraining_If::CTraining_If ()

Definition at line 96 of file Training_If.cpp.

References kDefaultAudioFileExtension, kNumOfOutFiles, kNumOfTrainingClasses, m_aiMaxDimensionsOfFeatureMatrix, m_pCFileList, m_pCTrainingInstance, m_ppfTmpFeatureMatrix, m_strDirectoryPath, m_strFileExtension, m_strOutFileNames, and Train_CreateInstance().

Referenced by CreateInstance().

```

97 {
98     int i;
99
100    // initialize members and alloc memory
101    for (i = 0; i < kNumOfTrainingClasses; i++)
102    {
103        m_strDirectoryPath[i].erase ();
104        m_pCFileList[i] = 0;
105        m_pCFileList[i] = new CFileList ();
106    }
107
108    for (i = 0; i < kNumOfOutFiles; i++)
109        m_strOutFileNames[i].erase ();
110
111    m_ppfTmpFeatureMatrix = 0;
112    memset (m_aiMaxDimensionsOfFeatureMatrix, 0, sizeof(m_aiMaxDimensionsOfFeatureMatrix));
113
114    // set extension to default value
115    m_strFileExtension.assign (kDefaultAudioFileExtension);
116
117    // create new instance of training class
118    Train_CreateInstance (&m_pCTrainingInstance);
119 }

```

8.3.3.2 CTraining_If::~~CTraining_If () [virtual]

Definition at line 122 of file Training_If.cpp.

References hlpFreeMatrix(), kNumOfTrainingClasses, m_aiMaxDimensionsOfFeatureMatrix, m_pCFileList, m_pCTrainingInstance, m_ppfTmpFeatureMatrix, and Train_DestroyInstance().

```

123 {
124     int i;
125
126    // free allocated memory
127    for (i = 0; i < kNumOfTrainingClasses; i++)
128        delete m_pCFileList[i];
129
130    hlpFreeMatrix (m_ppfTmpFeatureMatrix, m_aiMaxDimensionsOfFeatureMatrix);
131
132    // destroy instance of training class
133    Train_DestroyInstance (&m_pCTrainingInstance);
134 }

```

8.3.4 Member Function Documentation

8.3.4.1 **TrainingError_t** CTraining_If::CalculateFeatures ()

calculate all features for the files in the previously defined directories

Returns:

0 if no errors

Definition at line 269 of file Training_If.cpp.

References CalculateFeaturesAndAppendData4Training(), CFileList::GetEntry(), kClassMusic, kClassNoMusic, kNumOfTrainingClasses, kTrainNoError, kTrainNothingToDo, and m_pCFileList.

Referenced by main().

```

270 {
271     int          iClass;
272     TrainingError_t rErr          = kTrainNoError;
273
274     // check if we really have data to train
275     if ((m_pCFileList[kClassNoMusic]->GetNumOfEntries () == 0) || (m_pCFileList[kClassMusic]->
276         return kTrainNothingToDo;
277
278     for (iClass = 0; iClass < kNumOfTrainingClasses; iClass++)
279     {
280         int iNumOfEntries    = m_pCFileList[iClass]->GetNumOfEntries ();
281
282         // do feature extraction for all files
283         for (int iFile = 0; iFile < iNumOfEntries; iFile++)
284         {
285             string CurrentFile;
286             m_pCFileList[iClass]->GetEntry (CurrentFile, iFile);
287
288             // do the feature calculation and add the features to the training set
289             rErr    = this->CalculateFeaturesAndAppendData4Training (CurrentFile, (TrainingClas
290             if (rErr != kTrainNoError)
291                 return rErr;
292         }
293     }
294
295     return kTrainNoError;
296 }

```

8.3.4.2 **TrainingError_t** CTraining_If::CalculateFeaturesAndAppend-Data4Training (string *strFileName*, **TrainingClasses_t** *eClass*) [private]

calculates the features for a file and appends data to training set for the given class

Parameters:

strFileName audio file where the features should be extracted from
eClass definition of class

Returns:

0 if no error

See also:

Segmentation_C.h|Training_C.h|sndfile.h

Definition at line 340 of file Training_If.cpp.

References `hlpReallocMatrix()`, `kNumOfAudioSamples2Read`, `kTrainFileOpenError`, `kTrainNoError`, `m_aiMaxDimensionsOfFeatureMatrix`, `m_pCTrainingInstance`, `m_ppfTmpFeatureMatrix`, and `Train_AppendFeatureData()`.

Referenced by `CalculateFeatures()`.

```

341 {
342     TrainingError_t rErr           = kTrainNoError;
343     SNDFILE         *pFInputFile   = 0;           // input file handle
344     SF_INFO         sfInputInfo;   // libsndfile structure with
345     void            *pCSegmentation = 0;         // handle to segmentation
346     bool            bReadNextFrame = true;
347     int             aiFeatureMatrixDimensions[2] = {0, 0};
348
349     // open sound file
350     pFInputFile     = sf_open (strFileName.c_str (),
351                               SFM_READ,
352                               &sfInputInfo);
353     if (pFInputFile == NULL)
354         return kTrainFileOpenError;
355
356     // create class instance and initialize it
357     Segment_CreateInstance (&pCSegmentation, sfInputInfo.samplerate, sfInputInfo.channels);
358     Segment_Initialize (pCSegmentation, sfInputInfo.frames);
359
360     // now do the feature calculation
361     while(bReadNextFrame)
362     {
363         float    afAudioData[kNumOfAudioSamples2Read];
364
365         // read data from file
366         int iNumSamplesRead = (int)(sf_read_float (pFInputFile, afAudioData, kNumOfAudioSamples2Read));
367
368         if(iNumSamplesRead < kNumOfAudioSamples2Read)
369         {
370             //memset (&afAudioData[iNumSamplesRead], 0, (kNumOfAudioSamples2Read - iNumSamplesRead));
371             bReadNextFrame = false;
372         }
373
374         //processing
375         Segment_Process (pCSegmentation, afAudioData, iNumSamplesRead / sfInputInfo.channels);
376
377     } // process loop
378
379     // finish processing
380     Segment_FinishProcess (pCSegmentation);
381
382     // now get the feature data
383
384     // get required buffer size
385     Segment_GetSizeOfIntermediateResult (pCSegmentation, &aiFeatureMatrixDimensions[0], &aiFeatureMatrixDimensions[1]);

```

```

387
388     // allocate temp memory if required
389     rErr     = hlpReallocMatrix (m_ppfTmpFeatureMatrix, aiFeatureMatrixDimensions, m_aiMaxDimen
390     if (rErr != kTrainNoError)
391         return rErr;
392
393     // get result
394     Segment_GetIntermediateResult (pCSegmentation, m_ppfTmpFeatureMatrix);
395
396     // append result to training data
397     rErr     = Train_AppendFeatureData (m_pCTrainingInstance, m_ppfTmpFeatureMatrix, aiFeatureM
398     if (rErr != kTrainNoError)
399         return rErr;
400
401     // destroy feature calculation class
402     Segment_DestroyInstance (&pCSegmentation);
403
404     // close audio file
405     sf_close (pFInputFile);
406
407     return kTrainNoError;
408 }

```

8.3.4.3 TrainingError_t CTraining_If::CreateInstance (CTraining_If *& p-CTraining_If) [static]

creates a new instance of [CTraining_If](#)

Parameters:

pCTraining_If instance handle is written to here

Returns:

0 if no error

Definition at line 136 of file Training_If.cpp.

References [CTraining_If\(\)](#), [kTrainMemoryAllocError](#), and [kTrainNoError](#).

Referenced by [main\(\)](#).

```

137 {
138     TrainingError_t rErr     = kTrainNoError;
139     pCTraining_If = 0;
140
141     // create class instance
142     pCTraining_If = new CTraining_If ();
143
144     if (pCTraining_If == 0)
145         return kTrainMemoryAllocError;
146
147     return kTrainNoError;
148 }
149 }

```

8.3.4.4 TrainingError_t CTraining_If::DestroyInstance (CTraining_If *& p-CTraining_If) [static]

destroys an instance of CTraining_If

Parameters:

pCTraining_If instance handle to be destroyed

Returns:

0 if no error

Definition at line 152 of file Training_If.cpp.

References kTrainInvalidPointerError, and kTrainNoError.

Referenced by main().

```
153 {
154     if (pCTraining_If == 0)
155         return kTrainInvalidPointerError;
156
157     // destroy class instance
158     delete pCTraining_If;
159     pCTraining_If = 0;
160
161     return kTrainNoError;
162
163 }
```

8.3.4.5 TrainingError_t CTraining_If::ParseDirectory (string strPath, CFileList *pCFileList) [private]

parses a complete directory for files with a special extension and adds the file names to a list

Parameters:

strPath Path to be searched (no subdirectories are searched)

pCFileList handle of file list

Returns:

0 if no error

Definition at line 304 of file Training_If.cpp.

References CFileList::Add2List(), kTrainNoError, and m_strFileExtension.

Referenced by SetParamDirectoryPaths().

```
305 {
306     struct _finddata_t      CurrentFile;
307     long                   hFile;
```

```

308     string                               strWildcard;
309
310     strWildcard.assign (strPath + "*" + m_strFileExtension);
311
312     // find first file
313     if( (hFile = _findfirst( strWildcard.c_str (), &CurrentFile )) == -1L )
314         return kTrainNoError;
315     else
316     {
317         string CurrentFileName;
318
319         // file found, add it to the list
320         CurrentFileName.assign (CurrentFile.name);
321         pCFileList->Add2List (strPath + CurrentFileName);
322
323         // Find the rest of the files
324         while( _findnext( hFile, &CurrentFile ) == 0 )
325         {
326
327             // file found, add it to the list
328             CurrentFileName.assign (CurrentFile.name);
329             pCFileList->Add2List (strPath + CurrentFileName);
330         }
331
332         // has to be called at the end
333         _findclose( hFile );
334     }
335
336     return kTrainNoError;
337 }

```

8.3.4.6 **TrainingError_t** CTraining_If::SetParamAudioFileExtension (string *strAudioFileExtension*)

sets the extension of the searched audio files (default: .wav)

Parameters:

strAudioFileExtension extension

Returns:

0 if no error

Definition at line 189 of file Training_If.cpp.

References kTrainNoError, and m_strFileExtension.

```

190 {
191     m_strFileExtension.assign (strAudioFileExtension);
192
193     return kTrainNoError;
194 }

```

8.3.4.7 **TrainingError_t** CTraining_If::SetParamDirectoryPaths (string *strPathToMusicDir*, string *strPathToNonMusicDir*)

sets the directory paths to the audio files per class used for training

Parameters:

strPathToMusicDir sets the path to the music input files

strPathToNonMusicDir sets the path to the non-music input files

Remarks:

the call of this function is mandatory

Returns:

0 if no error

Definition at line 165 of file Training_If.cpp.

References kClassMusic, kClassNoMusic, kNumOfTrainingClasses, kSlash, kTrainNoError, m_pCFileList, m_strDirectoryPath, and ParseDirectory().

Referenced by main().

```

166 {
167     TrainingError_t rErr = kTrainNoError;
168
169     // set directory paths
170     m_strDirectoryPath[kClassMusic].assign (strPathToMusicDir);
171     m_strDirectoryPath[kClassNoMusic].assign (strPathToNonMusicDir);
172
173     // generate file lists
174     for (int i = 0; i < kNumOfTrainingClasses; i++)
175     {
176         // check if we have a slash
177         if (m_strDirectoryPath[i].length ()-1, 1, kSlash) != 0)
178             m_strDirectoryPath[i] += kSlash;
179
180         // now list all audio files
181         rErr = this->ParseDirectory (m_strDirectoryPath[i], m_pCFileList[i]);
182         if (rErr != kTrainNoError)
183             return rErr;
184     }
185
186     return rErr;
187 }

```

8.3.4.8 TrainingError_t CTraining_If::SetParamOutFilePath (string strPath)

sets the path where the output files will be written (default: working directory)

Parameters:

strPath output file path (directory)

Returns:

0 if no error

Definition at line 196 of file Training_If.cpp.

References kFileMeans, kFileScale, kNumOfTrainingClasses, kSlash, kTrainNoError, m_strOutFileNames, and pcOutFileNames.

Referenced by main().

```

197 {
198     // assign directory names
199     m_strOutFileNames[kFileMeans].assign (strPath);
200     m_strOutFileNames[kFileScale].assign (strPath);
201
202     for (int i = 0; i < kNumOfTrainingClasses; i++)
203     {
204         // check if we have a slash
205         if (m_strOutFileNames[i].compare (m_strOutFileNames[i].length ()-1, 1, kSlash) != 0)
206             m_strOutFileNames[i] += kSlash;
207
208         // now set the file names
209         m_strOutFileNames[i].append (pcOutFileNames[i]);
210     }
211
212     return kTrainNoError;
213 }

```

8.3.4.9 **TrainingError_t** CTraining_If::SetTrainingData (float ** *ppfFeatures*, **TrainingClasses_t** *eClass*, int *iNumOfObservations*, int *iNumOfFeatures*)

set new classifier training data

Parameters:

ppfFeatures feature data (dimensions: [iNumOfFeatures]x[iNumOfObservations])

eClass class label

iNumOfObservations number of observations

iNumOfFeatures number of features

Returns:

0 if no error

Definition at line 410 of file Training_If.cpp.

References m_pCTrainingInstance, and Train_AppendFeatureData().

```

411 {
412     int aiDimension[2] = {iNumOfFeatures, iNumOfObservations};
413
414     return Train_AppendFeatureData (m_pCTrainingInstance, ppfFeatures, aiDimension, eClass);
415
416 }

```

8.3.4.10 TrainingError_t CTraining_If::Train ()

after the calculation of features and setting all training data, finally do the training itself

Returns:

0 if no error

Remarks:

either SetTrainingData or CalculateFeatures have to be called before Process can be called successfully

Definition at line 298 of file Training_If.cpp.

References m_pCTrainingInstance, and Train_Process().

Referenced by main().

```

299 {
300     // just do the training here...
301     return Train_Process (m_pCTrainingInstance);
302 }
```

8.3.4.11 TrainingError_t CTraining_If::WriteTrainingResults ()

retrieves the training results and writes them to the output files

Returns:

0 if now error

Definition at line 215 of file Training_If.cpp.

References kNumOfResults, kTrainFileOpenError, kTrainNoError, m_pCTrainingInstance, m_strOutFileNames, Train_GetResult(), and Train_GetResultDimension().

Referenced by main().

```

216 {
217     // do for all results
218     for (int r = 0; r < kNumOfResults; r++)
219     {
220         float          **ppfOutData          = 0;
221         int            i,
222                     aiOutDataDimensions[2] = {0,0};
223         std::ofstream aFFileHandle;
224
225         if (m_strOutFileNames[r].empty ())
226             return kTrainFileOpenError;
227
228         // get output dimensions
229         Train_GetResultDimension (m_pCTrainingInstance, (TrainingResults_t)r, aiOutDataDimensions);
230
231         // alloc memory
232     }
```

```

233     ppfOutData = new float* [aiOutDataDimensions[0]];
234     for (i = 0; i < aiOutDataDimensions[0]; i++)
235         ppfOutData[i] = new float [aiOutDataDimensions[1]];
236
237     // get output data
238     Train_GetResult (m_pCTrainingInstance, (TrainingResults_t)r, ppfOutData, aiOutDataDimen
239
240     // open output file
241     aFFileHandle.open (m_strOutFileNames[r].c_str (), std::ios::out);
242
243     if (aFFileHandle.is_open ())
244     {
245         // write file
246         for (i = 0; i < aiOutDataDimensions[0]; i++) // rows
247         {
248             for (int j = 0; j < aiOutDataDimensions[1]; j++) // cols
249                 aFFileHandle << std::scientific << std::setprecision(10) << ppfOutData[i][j];
250             aFFileHandle << std::endl;
251         }
252     }
253
254     // close file
255     if (aFFileHandle.is_open ())
256         aFFileHandle.close ();
257
258     // free memory
259     for (i = 0; i < aiOutDataDimensions[0]; i++)
260         delete [] ppfOutData[i];
261     delete [] ppfOutData;
262     ppfOutData = 0;
263 }
264
265     return kTrainNoError;
266 }

```

8.3.5 Field Documentation

8.3.5.1 `int` **CTraining_If::m_aiMaxDimensionsOfFeatureMatrix[2]**
 [private]

size of m_ppfTmpFeatureMatrix

Definition at line 255 of file Training_If.h.

Referenced by CalculateFeaturesAndAppendData4Training(), CTraining_If(), and ~CTraining_If().

8.3.5.2 `CFileList*` **CTraining_If::m_pCFileList[kNumOfTrainingClasses]**
 [private]

list of all audio files per class directory

Definition at line 252 of file Training_If.h.

Referenced by CalculateFeatures(), CTraining_If(), SetParamDirectoryPaths(), and ~CTraining_If().

8.3.5.3 `void*` **CTraining_If::m_pCTrainingInstance** [private]

training instance

Definition at line 256 of file Training_If.h.

Referenced by CalculateFeaturesAndAppendData4Training(), CTraining_If(), SetTrainingData(), Train(), WriteTrainingResults(), and ~CTraining_If().

8.3.5.4 float** CTraining_If::m_ppfTmpFeatureMatrix [private]

internal memory for feature data

Definition at line 254 of file Training_If.h.

Referenced by CalculateFeaturesAndAppendData4Training(), CTraining_If(), and ~CTraining_If().

8.3.5.5 string CTraining_If::m_strDirectoryPath[kNumOfTrainingClasses] [private]

class directory paths

Definition at line 249 of file Training_If.h.

Referenced by CTraining_If(), and SetParamDirectoryPaths().

8.3.5.6 string CTraining_If::m_strFileExtension [private]

audio file extension to parse for

Definition at line 249 of file Training_If.h.

Referenced by CTraining_If(), ParseDirectory(), and SetParamAudioFileExtension().

8.3.5.7 string CTraining_If::m_strOutFileNames[kNumOfOutFiles] [private]

output file names

Definition at line 249 of file Training_If.h.

Referenced by CTraining_If(), SetParamOutFilePath(), and WriteTrainingResults().

The documentation for this class was generated from the following files:

- [Training_If.h](#)
- [Training_If.cpp](#)

9 Training SDK File Documentation

9.1 FileList.cpp File Reference

9.1.1 Detailed Description

implementation of the `CFileList` class.

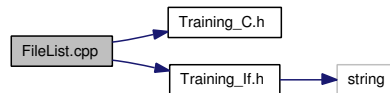
:

Definition in file `FileList.cpp`.

```
#include "Training_C.h"
```

```
#include "Training_If.h"
```

Include dependency graph for `FileList.cpp`:



Defines

- `#define kDefaultMaxNumOfEntries 32`
default length of list at init

9.1.2 Define Documentation

9.1.2.1 `#define kDefaultMaxNumOfEntries 32`

default length of list at init

Definition at line 61 of file `FileList.cpp`.

Referenced by `CFileList::CFileList()`.

9.2 HelperFunctions.cpp File Reference

9.2.1 Detailed Description

helper functions

:

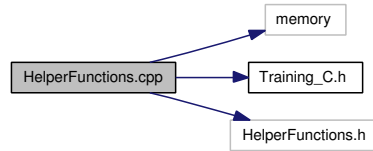
Definition in file `HelperFunctions.cpp`.

```
#include <memory>
```

```
#include "Training_C.h"
```

```
#include "HelperFunctions.h"
```

Include dependency graph for HelperFunctions.cpp:



Functions

- **TrainingError_t hlpReallocMatrix** (float **&ppfMatrix, int *piNewDimensions, int *piOldDimensions)
- **TrainingError_t hlpFreeMatrix** (float **&ppfMatrix, int *piDimensions)
- **TrainingError_t hlpCopyMatrix** (float **ppfDest, float **ppfSrc, int *piDestDimensions, int *piSrcDimensions, int *piDestStartIndices)

9.2.2 Function Documentation

9.2.2.1 **TrainingError_t hlpCopyMatrix** (float ** *ppfDest*, float ** *ppfSrc*, int * *piDestDimensions*, int * *piSrcDimensions*, int * *piDestStartIndices*)

Definition at line 138 of file HelperFunctions.cpp.

References kTrainMatrixDimensionMismatch, and kTrainNoError.

```

139 {
140     int i,
141         iStart = piDestStartIndices[0];
142
143     if (piDestDimensions[0] - piDestStartIndices[0] != piSrcDimensions[0])
144         return kTrainMatrixDimensionMismatch;
145     if (piDestDimensions[1] - piDestStartIndices[1] != piSrcDimensions[1])
146         return kTrainMatrixDimensionMismatch;
147
148     for (i = iStart; i < iStart + piSrcDimensions[0]; i++)
149         memcpy (&ppfDest[iStart][piDestStartIndices[1]], &ppfSrc[i - iStart][0], (piSrcDimensions[0] - i + iStart));
150
151     return kTrainNoError;
152 }
  
```

9.2.2.2 **TrainingError_t hlpFreeMatrix** (float **& *ppfMatrix*, int * *piDimensions*)

Definition at line 122 of file HelperFunctions.cpp.

References kTrainInvalidPointerError, and kTrainNoError.

Referenced by CTraining_If::~CTraining_If().

```

123 {
124     if (!ppfMatrix)
125         return kTrainInvalidPointerError;
126
127     // free columns
128     for (int i = 0; i < piDimensions[0]; i++)
129         free (ppfMatrix[i]);
130
131     // free rows
132     free (ppfMatrix);
133     ppfMatrix = 0;
134
135     return kTrainNoError;
136 }

```

9.2.2.3 **TrainingError_t** hlpReallocMatrix (float **& ppfMatrix, int * piNewDimensions, int * piOldDimensions)

Definition at line 57 of file HelperFunctions.cpp.

References kTrainMemoryAllocError, and kTrainNoError.

Referenced by CTraining_If::CalculateFeaturesAndAppendData4Training().

```

58 {
59     int     i;
60     void    *pTmp;
61
62     // we do not need to reallocate if the previous matrix size is equal or greater
63     if (piNewDimensions[0] <= piOldDimensions[0] && piNewDimensions[1] <= piOldDimensions[1])
64         return kTrainNoError;
65
66     // new alloc of whole matrix
67     if (!ppfMatrix)
68     {
69         // alloc rows
70         ppfMatrix = (float**)malloc (sizeof(float)*piNewDimensions[0]);
71         if (!ppfMatrix)
72             return kTrainMemoryAllocError;
73
74         // alloc columns
75         for (i = 0; i < piNewDimensions[0]; i++)
76         {
77             ppfMatrix[i] = (float*)malloc (sizeof(float)*piNewDimensions[1]);
78             if (!ppfMatrix[i])
79                 return kTrainMemoryAllocError;
80         }
81     }
82     else
83     {
84         // alloc more rows
85         if (piNewDimensions[0] > piOldDimensions[0])
86         {
87             pTmp = realloc(ppfMatrix, sizeof(float)*piNewDimensions[0]);
88             if (!pTmp)
89                 return kTrainMemoryAllocError;
90             ppfMatrix = (float**)pTmp;
91         }
92
93         // alloc more columns

```

```

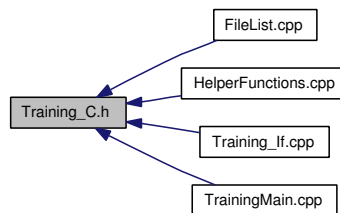
94     if (piNewDimensions[1] > piOldDimensions[1])
95     {
96         for (i = 0; i < piOldDimensions[0]; i++)
97         {
98             pTmp
99             = realloc (ppfMatrix[i], sizeof(float)*piNewDimensions[1]);
100             if (!pTmp)
101                 return kTrainMemoryAllocError;
102             ppfMatrix[i] = (float*)pTmp;
103         }
104     }
105     piNewDimensions[1] = (piNewDimensions[1] > piOldDimensions[1])? piNewDimensions[1] : piOldDimensions[1];
106     // alloc columns for new rows
107     for (i = piOldDimensions[0]; i < piNewDimensions[0]; i++)
108     {
109         pTmp
110         = malloc (sizeof(float)*piNewDimensions[1]);
111         if (!pTmp)
112             return kTrainMemoryAllocError;
113         ppfMatrix[i] = (float*)pTmp;
114     }
115     // update dimensions
116     piOldDimensions[0] = (piNewDimensions[0] > piOldDimensions[0])? piNewDimensions[0] : piOldDimensions[0];
117     piOldDimensions[1] = (piNewDimensions[1] > piOldDimensions[1])? piNewDimensions[1] : piOldDimensions[1];
118     return kTrainNoError;
119 }
120 }

```

9.3 training.txt File Reference

9.4 Training_C.h File Reference

This graph shows which files directly or indirectly include this file:



Data Structures

- struct **ClassifierInfo_t_tag**
info structure on the training data

Defines

- #define **__libTrainingC_HEADER_INCLUDED__**

Typedefs

- typedef enum **TrainingClasses_t_tag** **TrainingClasses_t**
indices of training classes
- typedef enum **TrainingResults_t_tag** **TrainingResults_t**
indices of training results
- typedef enum **TrainingError_t_tag** **TrainingError_t**
definition of training errors
- typedef **ClassifierInfo_t_tag** **ClassifierInfo_t**
info structure on the training data

Enumerations

- enum **TrainingClasses_t_tag** { **kClassNoMusic**, **kClassMusic**, **kNumOfTrainingClasses** }
indices of training classes
- enum **TrainingResults_t_tag** { **kResultMeans**, **kResultScale**, **kNumOfResults** }
indices of training results
- enum **TrainingError_t_tag** {
kTrainNoError, **kTrainNothingToDo**, **kTrainMatrixDimensionMismatch**, **kTrainInvalidPointerError**,
kTrainInvalidArgumentError, **kTrainInvalidTrainingData**, **kTrainFileOpenError**,
kTrainMemoryAllocError,
kInternalError }
definition of training errors

Functions

- **TrainingError_t Train_CreateInstance** (void **pphTrainingHandle)
creates a new instance of training
- **TrainingError_t Train_DestroyInstance** (void **pphTrainingHandle)
destroys an instance of training
- **TrainingError_t Train_AppendFeatureData** (void *phTrainingHandle, float **ppfFeatureMatrix, int *piMatrixDimensions, **TrainingClasses_t** eClass)
append new feature data to the training set

- **TrainingError_t Train_GetClassifierInfo** (void *phTrainingHandle, **ClassifierInfo_t** *pInfo)
returns some information about the training data
- **TrainingError_t Train_Process** (void *phTrainingHandle)
do the training
- **TrainingError_t Train_GetResultDimension** (void *phTrainingHandle, **TrainingResults_t** eResultIdx, int *piResultDimensions)
returns the dimension, i.e. number of columns and number of rows of the result with the given index
- **TrainingError_t Train_GetResult** (void *phTrainingHandle, **TrainingResults_t** eResultIdx, float **ppfResult, const int *piResultDimensions)
returns the result with the specific index

9.4.1 Define Documentation

9.4.1.1 #define **__libTrainingC_HEADER_INCLUDED__**

Definition at line 2 of file Training_C.h.

9.4.2 Typedef Documentation

9.4.2.1 typedef struct **ClassifierInfo_t_tag ClassifierInfo_t**

info structure on the training data

9.4.2.2 typedef enum **TrainingClasses_t_tag TrainingClasses_t**

indices of training classes

9.4.2.3 typedef enum **TrainingError_t_tag TrainingError_t**

definition of training errors

9.4.2.4 typedef enum **TrainingResults_t_tag TrainingResults_t**

indices of training results

9.4.3 Enumeration Type Documentation

9.4.3.1 enum **TrainingClasses_t_tag**

indices of training classes

Enumerator:

kClassNoMusic
kClassMusic
kNumOfTrainingClasses

Definition at line 14 of file Training_C.h.

```
15     {  
16         kClassNoMusic,  
17         kClassMusic,  
18  
19         kNumOfTrainingClasses  
20     } TrainingClasses_t;
```

9.4.3.2 enum TrainingError_t_tag

definition of training errors

Enumerator:

kTrainNoError
kTrainNothingToDo
kTrainMatrixDimensionMismatch
kTrainInvalidPointerError
kTrainInvalidArgumentError
kTrainInvalidTrainingData
kTrainFileOpenError
kTrainMemoryAllocError
kInternalError

Definition at line 38 of file Training_C.h.

```
39     {  
40         kTrainNoError,  
41  
42         kTrainNothingToDo,  
43  
44         kTrainMatrixDimensionMismatch,  
45  
46         kTrainInvalidPointerError,  
47         kTrainInvalidArgumentError,  
48  
49         kTrainInvalidTrainingData,  
50  
51         kTrainFileOpenError,  
52  
53         kTrainMemoryAllocError,  
54  
55         kInternalError  
56     } TrainingError_t;
```

9.4.3.3 enum TrainingResults_t_tag

indices of training results

Enumerator:

kResultMeans
kResultScale
kNumOfResults

Definition at line 26 of file Training_C.h.

```

27     {
28         kResultMeans,
29         kResultScale,
30
31         kNumOfResults
32     } TrainingResults_t;
```

9.4.4 Function Documentation**9.4.4.1 TrainingError_t Train_AppendFeatureData (void * *phTrainingHandle*, float ** *ppfFeatureMatrix*, int * *piMatrixDimensions*, TrainingClasses_t *eClass*)**

append new feature data to the training set

Parameters:

phTrainingHandle handle to training instance
ppfFeatureMatrix new data (dimensions ([iNumOfFeatures]x[iNumOfObservations])
piMatrixDimensions dimensions of *ppfFeatureMatrix* [iRows][iColumns]
eClass class label

Returns:

0 if no error

Referenced by CTraining_If::CalculateFeaturesAndAppendData4Training(), and CTraining_If::SetTrainingData().

9.4.4.2 TrainingError_t Train_CreateInstance (void ** *pphTrainingHandle*)

creates a new instance of training

Parameters:

pphTrainingHandle handle to new instance

Returns:

0 if no error

Referenced by CTraining_If::CTraining_If().

9.4.4.3 TrainingError_t Train_DestroyInstance (void ** *phTrainingHandle*)

destroys an instance of training

Parameters:

phTrainingHandle handle to instance to be destroyed

Returns:

0 if no error

Referenced by CTraining_If::~~CTraining_If().

9.4.4.4 TrainingError_t Train_GetClassifierInfo (void * *phTrainingHandle*, ClassifierInfo_t * *pInfo*)

returns some information about the training data

Parameters:

phTrainingHandle handle to training instance

pInfo pointer to structure where the information is copied into

Returns:

0 if no error

9.4.4.5 TrainingError_t Train_GetResult (void * *phTrainingHandle*, TrainingResults_t *eResultIdx*, float ** *ppfResult*, const int * *piResultDimensions*)

returns the result with the specific index

Parameters:

phTrainingHandle handle to training instance

eResultIdx index of result we are interested in

ppfResult matrix where the result values are copied to

piResultDimensions dimensions of *ppfResult*, has to equal the parameter from [Train_GetResultDimension](#)

Returns:

0 if no error

See also:

[Train_GetResultDimension](#)

Referenced by CTraining_If::WriteTrainingResults().

9.4.4.6 TrainingError_t Train_GetResultDimension (void * *phTrainingHandle*, TrainingResults_t *eResultIdx*, int * *piResultDimensions*)

returns the dimension, i.e. number of columns and number of rows of the result with the given index

Parameters:

phTrainingHandle handle to training instance

eResultIdx index of result we are interested in

piResultDimensions the result dimensions are written here ([iNumOfRows]x[i-NumOfColumns])

Returns:

0 if no error

Referenced by CTraining_If::WriteTrainingResults().

9.4.4.7 TrainingError_t Train_Process (void * *phTrainingHandle*)

do the training

Parameters:

phTrainingHandle handle to training instance

Returns:

0 if no error

Referenced by CTraining_If::Train().

9.5 Training_If.cpp File Reference**9.5.1 Detailed Description**

implementation of the CTraining_If class.

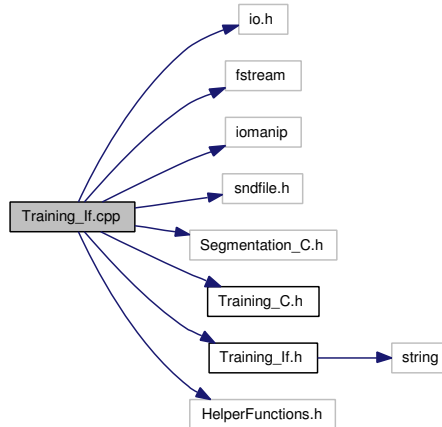
:

Definition in file Training_If.cpp.

```
#include <io.h>
#include <fstream>
#include <iomanip>
#include "sndfile.h"
#include "Segmentation_C.h"
#include "Training_C.h"
```

```
#include "Training_If.h"
#include "HelperFunctions.h"
```

Include dependency graph for Training_If.cpp:



Defines

- `#define kDefaultAudioFileExtension ".wav"`
initialization value of the audio file extension to parse for
- `#define kSlash "\\\"`
just a slash...
- `#define kNumOfAudioSamples2Read 4096`
for audio file parsing, this defines the input block size

Variables

- `static const char * pcOutFileNames [2]`
these are the names of the output files

9.5.2 Define Documentation

9.5.2.1 `#define kDefaultAudioFileExtension ".wav"`

initialization value of the audio file extension to parse for

Definition at line 72 of file Training_If.cpp.

Referenced by CTraining_If::CTraining_If().

9.5.2.2 #define kNumOfAudioSamples2Read 4096

for audio file parsing, this defines the input block size

Definition at line 94 of file Training_If.cpp.

Referenced by CTraining_If::CalculateFeaturesAndAppendData4Training().

9.5.2.3 #define kSlash "\\\"

just a slash...

Definition at line 78 of file Training_If.cpp.

Referenced by CTraining_If::SetParamDirectoryPaths(), and CTraining_If::SetParamOutFilePath().

9.5.3 Variable Documentation**9.5.3.1 const char* pcOutFileNames[2] [static]**

Initial value:

```
{
    {"Means.txt"},
    {"Scale.txt"},
}
```

these are the names of the output files

Definition at line 84 of file Training_If.cpp.

Referenced by CTraining_If::SetParamOutFilePath().

9.6 Training_If.h File Reference**9.6.1 Detailed Description**

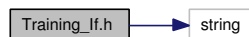
interface of the [CTraining_If](#) class.

:

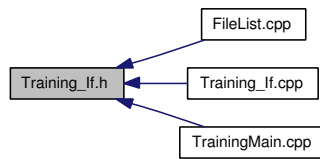
Definition in file [Training_If.h](#).

```
#include <string>
```

Include dependency graph for Training_If.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace `std`

Data Structures

- class `CTraining_If`
this class provides an interface and some additional functionality in the context of the for the training library
- class `CFileList`
used internally by `CTraining_If` to organize file names

Defines

- `#define __libTraining_If_HEADER_INCLUDED__`

9.6.2 Define Documentation

9.6.2.1 `#define __libTraining_If_HEADER_INCLUDED__`

Definition at line 60 of file `Training_If.h`.

9.7 TrainingMain.cpp File Reference

9.7.1 Detailed Description

implementation of the training command line application.

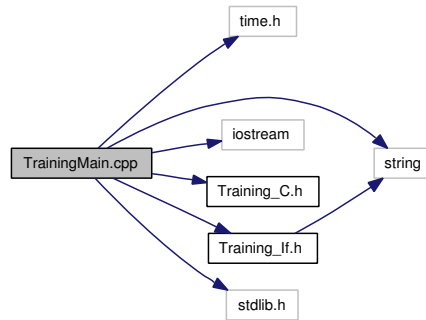
:

Definition in file `TrainingMain.cpp`.

```
#include <time.h>
#include <string>
#include <iostream>
#include "Training_C.h"
#include "Training_If.h"
```

```
#include <stdlib.h>
```

Include dependency graph for TrainingMain.cpp:



Defines

- #define **kNumMinCLArgs** (kNumDirectories+1)

Enumerations

- enum **Directories_t** { **kDirNoMusic**, **kDirMusic**, **kDirOutfile**, **kNumDirectories** }

Functions

- static void **CLShowProgInfo** ()
- static void **CLReadArgs** (string strDirectories[kNumDirectories], int argc, char *argv[])
- static void **CLShowProcessedTime** (clock_t cTime)
- int **main** (__int32 argc, char *argv[])

9.7.2 Define Documentation

9.7.2.1 #define **kNumMinCLArgs** (kNumDirectories+1)

Definition at line 75 of file TrainingMain.cpp.

Referenced by main().

9.7.3 Enumeration Type Documentation

9.7.3.1 enum **Directories_t**

Enumerator:

kDirNoMusic

kDirMusic
kDirOutfile
kNumDirectories

Definition at line 65 of file TrainingMain.cpp.

```
66 {  
67     kDirNoMusic,  
68     kDirMusic,  
69     kDirOutfile,  
70  
71     kNumDirectories  
72 };
```

9.7.4 Function Documentation

9.7.4.1 static void CLReadArgs (string *strDirectories*[kNumDirectories], int *argc*, char * *argv*[]) [static]

Definition at line 161 of file TrainingMain.cpp.

Referenced by main().

```
162 {  
163     for (int i = 1; i < kNumDirectories+1; i++)  
164         strDirectories[i-1] = argv[i];  
165  
166     return;  
167 }
```

9.7.4.2 static void CLShowProcessedTime (clock_t *clTime*) [static]

Definition at line 169 of file TrainingMain.cpp.

Referenced by main().

```
170 {  
171     cout << "\nTime elapsed:\t" << ((float)(clock () - clTime) / CLOCKS_PER_SEC) << " sec" << endl;  
172  
173     return;  
174 }
```

9.7.4.3 static void CLShowProgInfo () [static]

Definition at line 152 of file TrainingMain.cpp.

Referenced by main().

```
153 {  
154     cout << "zplane.development Classification Training Commandline Application\n";  
155     cout << "(c) 2002-2006 by zplane\n";  
156     cout << "TrainingCL DirectoryPathNoMusic DirectoryPathMusic DirectoryPathOutFiles";  
157  
158     return;  
159 }
```

9.7.4.4 int main (__int32 argc, char * argv[])

Definition at line 84 of file TrainingMain.cpp.

References CTraining_If::CalculateFeatures(), CLReadArgs(), CLShowProcessedTime(), CLShowProgInfo(), CTraining_If::CreateInstance(), CTraining_If::DestroyInstance(), kDirMusic, kDirNoMusic, kDirOutfile, kNumDirectories, kNumMinCLArgs, CTraining_If::SetParamDirectoryPaths(), CTraining_If::SetParamOutFilePath(), CTraining_If::Train(), and CTraining_If::WriteTrainingResults().

```

85 {
86     clock_t          clTotalTime;
87
88     CTraining_If     *pCTrainingInstance = 0;
89
90     string           strDirectories[kNumDirectories];
91
92 #if (!defined(WITHOUT_MEMORY_CHECK) && defined(_DEBUG) && defined (WIN32))
93     // set memory checking flags
94     int iDbgFlag = _CrtSetDbgFlag(_CRTDBG_REPORT_FLAG);
95     iDbgFlag      |= _CRTDBG_CHECK_ALWAYS_DF;
96     iDbgFlag      |= _CRTDBG_LEAK_CHECK_DF;
97     _CrtSetDbgFlag( iDbgFlag );
98 #endif
99 #if (!defined(WITHOUT_EXCEPTIONS) && defined(_DEBUG) && defined (WIN32))
100     // enable check for exceptions (don't forget to enable stop in MSVC!)
101     _controlfp(~(_EM_INVALID | _EM_ZERODIVIDE | _EM_OVERFLOW | _EM_UNDERFLOW | _EM_DENORMAL), _MCW);
102 #endif // #ifndef WITHOUT_EXCEPTIONS
103
104     //check for correct number of command line arguments
105     if (argc < kNumMinCLArgs)
106     {
107         cout << "Wrong number of command line arguments!\n";
108         return -1;
109     }
110
111     // show application info
112     CLShowProgInfo ();
113
114     // read command line arguments
115     CLReadArgs (strDirectories, argc, argv);
116
117     // create class instance
118     CTraining_If::CreateInstance (pCTrainingInstance);
119     cout << "Created Training Instance...\n";
120
121     // set input and output directories
122     pCTrainingInstance->SetParamDirectoryPaths (strDirectories[kDirMusic], strDirectories[kDirNoMusic], strDirectories[kDirOutfile]);
123     pCTrainingInstance->SetParamOutFilePath (strDirectories[kDirOutfile]);
124     cout << "Parameters set...\n";
125
126     clTotalTime = clock ();
127
128     // do feature calculation
129     pCTrainingInstance->CalculateFeatures ();
130     cout << "Features calculated...\n";
131
132     // after we have our training data available, we are able to train
133     pCTrainingInstance->Train ();
134     cout << "Training done...\n";
135

```

```
136     // now write the data to the txt files
137     pCTrainingInstance->WriteTrainingResults ();
138     cout << "Output files written...\n";
139
140     CLShowProcessedTime (clTotalTime);
141
142
143     // destroy instance
144     CTraining_If::DestroyInstance (pCTrainingInstance);
145     cout << "Destroyed Training Instance...\n";
146
147     return 0;
148
149 }
```