zplane | zplane.development audio technology

[museg] SDK (Training) 2.6.4

by zplane.development

June 14, 2016

Contents

1 Training SDK Documentation

1.1 Introduction

The Training SDK provided by zplane generates a customized initialization and parametrization for the classification engine used by zplane's [museg] SDK. It allows to use a custom-defined training set to allow high user optimized class definition.

1.1.1 [museg] V2 SDK

The [museg] SDK segments an audio stream into two or more classes. Although it has been optimized to separate speech and music in broadcast audio streams, it is able to perform related tasks (classification, segmentation of other classes) as well if it is trained for these task.

The [museg] SDK provides three parts that can be seen and used separated from each other: the feature extraction, the training and the segmentation/classification itself. - In the feature extraction stage, meta data is extracted from audio data that should be able to describe the audio data (described more in detail in the documentation of the classification/segmentation step). This feature data can then either be used to train the algorithm for new classification tasks or to classify an unknown audio file with a previously trained classificator.

1.1.1.1 Training SDK content

The Training SDK is delivered in 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 FeatureExtraction.dll as being delivered with the [museg] SDK), and the Training-library itself with a C-interface that will be referred to as **internal API**.

The internal API can be interpreted as a raw training interface - it doesn't care what kind of training data is passed through it. The provided source code for extraction and management of the training data is the adaptation of this "raw" interface to the specific task of audio classification with the features provided by the Feature Extraction processing stage.

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 style, ...)
- 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 non-negligible amount of (subjectively) very clear-to-classify (non-noisy) data

1.3 Internal API Documentation

The SDK's training library provides a C-API which is available in the file Training-_C.h, previously named raw training interface. All required variable types 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 meta information 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.

If a two-dimensional buffer or array (e.g. afArray[i][j]) is easier to be interpreted as matrix, the first dimension (i) will be referred to as row and the second dimension (j) as column.

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

- **Train_CreateInstance (void** pphTrainingHandle, int iNumberOfClasses, Classifier_t eClassifier)**

  Creates a new instance for Training. The parameter pphTrainingHandle is written, the parameter iNumberOfClasses specifies the number of classes to be trained. The internal limit for the number of classes is 10. The parameter eClassifier specifies which classifier type should be trained (compare Classifier_t).

  Note that after one training processing, this data cannot be reused and the instance has to be destroyed and newly created.

  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 Initialization Functions

- **Train_AppendFeatureData (void$*$ phTrainingHandle, float $**$ppfFeatureMatrix, int $*$piMatrixDimensions, int iClassIdx)**

  Appends new data to the internal training set. The parameter pphTrainingHandle is the handle to the previously created instance and the parameter ppfFeatureMatrix contains the feature data of dimension [piMatrixDimensions[0]][piMatrixDimensions[1]]. The number of rows (piMatrixDimensions[0]) is the number of features and has to stay constant over the whole training set; the number of columns (piMatrixDimensions[1]) is the number of observations per feature. - The number of columns can vary among calls of Train_AppendFeatureData. The parameter iClassIdx reflects the user's index of the class this data represents. - The class index is required to have the following range: $0 <= iClassIdx < iNumberOfClasses$.

  Note that after one training processing, this data cannot be reused and the instance has to be destroyed and newly created.

  The function returns 0 in the case of no error.

1.3.5 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 previously created instance. The function Train_Process requires feature data for training; therefore the preceeding successful call of function Train_AppendFeatureData is mandatory. The call of this function is mandatory.

  Note that the internal data is modified by this function. It cannot be called twice for one instance.

  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 previously created instance, the parameter eResultIdx is the index of the result (currently, two results are possible, and the function has to be called for both of them), 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 previously 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 current training results consist of two matrices indexed by eResultIdx. These have to be stored for later usage by the classification engine.

  The function returns 0 in the case of no error.

1.3.6 Additional Functions

- **Train_GetBuildDateString ()**

  Returns a char containing the build date of the training library. This function may also be called before instance creation.

- **Train_GetTrainingSetInfo (void∗ phTrainingHandle, TrainingSetInfo_t ∗p-Info)**

  Provides information about the training data and allows to make sure that its properties are correct or to store them for later reference. This function may be called **before** Train_Process.

  The function returns 0 in the case of no error.

- **Train_GetClassifierType (void∗ phTrainingHandle)**

  Returns the index of the classifier type as selected in function Train_Create-Instance (see Classifier_t).

- **Train_GetVersion (Version_t eVersionIdx)**

  Returns an int with the (major, minor,...) version of the training library. This function may also be called before instance creation.

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 FeatureExtraction 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, int iNumber-OfClasses = 2)**

  Creates a new instance of CTraining_If. The handle of the new instance is written to parameter pCTraining_If. The parameter iNumberOfClasses has to be set if the required number of classes is larger than 2.

  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 ∗pstrPaths)**

  Sets the path to the training data for each class. The parameter pstrPaths is an array of N strings with N=iNumberOfClasses; each string contains the path to a directory that contains audio training data for one class.

  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, i.e. the training results, used for the classification process (default working directory).

  The function returns 0 in the case of no error.

- **CTraining_If::SetTrainingData (float ∗∗ppfFeatures, int iClassIdx, 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 iClassIdx is the class index of the class this data represents (see above).

  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 features, i.e. the training data, have been computed, we are able to complete the training process. This can take its time...

  The function returns 0 in the case of no error.

- **CTraining_If::WriteTrainingResults ()**

  After successful training the resulting output files can be written for later usage in classification.

  The function returns 0 in the case of no error.

1.4.1 Calling Conventions

This is a Step-by-step introducttion for the usage of the SDK. The complete code can be found in the example source file TrainingClMain.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, iNumOfClasses,
  kClassifierQDA);
```

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

```
// set input and output directories
pCTrainingInstance->SetParamDirectoryPaths (pstrDirectories);
pCTrainingInstance->SetParamOutFilePath (pstrDirectories[iNumOfClasses]);
cout << "Parameters set...\n";
```

and 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 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 Support

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

`zplane.development`

grunewaldstr. 83

d-10823 berlin

germany

fon: +49.30.854 09 15.0

fax: +49.30.854 09 15.5

@: `info@zplane.de`

2 Directory Hierarchy

2.1 Directories

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

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

5 Directory Documentation

5.1 E:/Visual Studio Projects/zplane/CMakeTestDir/museg/incl/ Directory Reference

Directory dependency graph for E:/Visual Studio Projects/zplane/CMakeTestDir/museg/incl/-
:

Files

- file Classification_C.h

  C-wrapper for the CClassification class.
- file FeatureExtraction_C.h

  C-wrapper for the CFeatureExtraction class.
- file FeatureSimilarity_C.h

  C-wrapper for the CFeatureSimilarity class.
- file Globals.h

  some global constants/types.
- file Segmentation.h

  interface of the CSegmentation class.
- file Segmentation_C.h
- file Training.h

  interface of the CTraining class.
- file Training_C.h

  C-interface wrapper for the Training.

5.2 E:/Visual Studio Projects/zplane/CMakeTestDir/museg/src/ Directory Reference

Directory dependency graph for E:/Visual Studio Projects/zplane/CMakeTestDir/museg/src/-
:

Directories

- directory TrainingCl

5.3 E:/Visual Studio Projects/zplane/CMakeTestDir/museg/src/TrainingCl/ Directory Reference

Directory dependency graph for E:/Visual Studio Projects/zplane/CMakeTestDir/museg/src/-TrainingCl/:

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 TrainingClMain.cpp

6 Data Structure Documentation

6.1 CFileList Class Reference

used internally by CTraining_If to organize file names

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

Public Member Functions

- CFileList ()
- virtual ∼CFileList ()
- int GetNumOfEntries () const

  returns the number of file name entries in the list
- zError_t Add2List (string strFileName)

  add new file name entry to the end of the list
- zError_t GetEntry (string &strFileName, int iIndex)

  retrieves one file name/entry from the list
- zError_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

6.1.1 Detailed Description

used internally by CTraining_If to organize file names

organizes a list of filenames

Definition at line 242 of file Training_If.h.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 CFileList::CFileList ( )

Definition at line 63 of file FileList.cpp.

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

```
{
    m_iMaxNumOfEntries  = kDefaultMaxNumOfEntries;

    // alloc list with default length
    m_pstrFilePaths     = new string [m_iMaxNumOfEntries];
    for (int i = 0; i < m_iMaxNumOfEntries; i++)
        m_pstrFilePaths[i].erase ();

    this->Reset ();

}
```

Here is the call graph for this function:

6.1.2.2 CFileList::∼CFileList ( ) [virtual]

Definition at line 77 of file FileList.cpp.

References m_pstrFilePaths.

```
{
    // delete allocated memory
    delete [] m_pstrFilePaths;
}
```

6.1.3 Member Function Documentation

6.1.3.1 zError_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 kMlMemAllocFailed, kMlNoError, m_iMaxNumOfEntries, m_iNumOf-Entries, and m_pstrFilePaths.

Referenced by CTraining_If::ParseDirectory().

```
{
    // check if list is full or not, if yes, realloc memory
    if (m_iNumOfEntries == m_iMaxNumOfEntries)
    {
        int     i;
        string  *pstrTmp = 0;

        // realloc memory
        pstrTmp = new string [m_iMaxNumOfEntries<<1];

        if (!pstrTmp)
        {
            m_iMaxNumOfEntries    >>= 1;
            return kMlMemAllocFailed;
        }

        for (i = 0; i < m_iMaxNumOfEntries; i++)
            pstrTmp[i].assign (m_pstrFilePaths[i]);

        for (i = m_iMaxNumOfEntries; i < (m_iMaxNumOfEntries<<1); i++)
            pstrTmp[i].erase ();

        delete [] m_pstrFilePaths;

        m_pstrFilePaths         = pstrTmp;
        m_iMaxNumOfEntries    <<= 1;
    }

    // add the new entry to the end of the list and increment number of list
       entries
    m_pstrFilePaths[m_iNumOfEntries].assign (strFileName);

    m_iNumOfEntries++;

    return kMlNoError;
}
```

6.1.3.2 zError_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 kMlInvalidArgument, kMlNoError, m_iNumOfEntries, and m_pstrFilePaths.

Referenced by CTraining_If::CalculateFeatures().

```
{
    if (iIndex >= m_iNumOfEntries)
        return kMlInvalidArgument;

    strFileName.assign (m_pstrFilePaths[iIndex]);

    return kMlNoError;
}
```

6.1.3.3 int CFileList::GetNumOfEntries ( ) const

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.

Referenced by CTraining_If::CalculateFeatures().

```
{
    return m_iNumOfEntries;
}
```

6.1.3.4 zError_t CFileList::Reset ( )

removes all entries from the list

Returns

0 if no error

Definition at line 100 of file FileList.cpp.

References kMlNoError, m_iMaxNumOfEntries, m_iNumOfEntries, and m_pstrFilePaths.

Referenced by CFileList().

```
{
    int i;

    for (i = 0; i < m_iMaxNumOfEntries; i++)
        m_pstrFilePaths[i].erase ();

    m_iNumOfEntries     = 0;

    return kMlNoError;

}
```

6.1.4 Field Documentation

6.1.4.1 int CFileList::m_iMaxNumOfEntries [private]

current available memory for entries

Definition at line 298 of file Training_If.h.

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

6.1.4.2 int CFileList::m_iNumOfEntries [private]

number of entries

Definition at line 298 of file Training_If.h.

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

6.1.4.3 string∗ CFileList::m_pstrFilePaths [private]

list of entries

Definition at line 300 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

6.2 ClassificationResult_t_tag Struct Reference

`#include <Classification_C.h>`

Data Fields

- float iStartTimeInS
  segment start
- float iStopTimeInS
  segment stop

- int iEstimatedClassIdx

    class
- float fEstimationReliability

    estimate of result reliability

6.2.1 Detailed Description

declaration of result structure

Definition at line 55 of file Classification_C.h.

6.2.2 Field Documentation

6.2.2.1 float ClassificationResult_t_tag::fEstimationReliability

estimate of result reliability

Definition at line 62 of file Classification_C.h.

6.2.2.2 int ClassificationResult_t_tag::iEstimatedClassIdx

class

Definition at line 60 of file Classification_C.h.

6.2.2.3 float ClassificationResult_t_tag::iStartTimeInS

segment start

Definition at line 57 of file Classification_C.h.

6.2.2.4 float ClassificationResult_t_tag::iStopTimeInS

segment stop

Definition at line 57 of file Classification_C.h.

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

- Classification_C.h

6.3 CSegmentation Class Reference

`#include <Segmentation.h>`

Collaboration diagram for CSegmentation:

Public Member Functions

- CSegmentation (int iSampleRate, int iNumberOfChannels)
- virtual ∼CSegmentation ()

- zERROR Initialize (int iOverallInputFileLengthInFrames)
- zERROR PreProcess (float ∗pfInputBufferInterleaved, int iNumberOfFrames)
- zERROR Process (float ∗pfInputBufferInterleaved, int iNumberOfFrames)
- zERROR FinishProcess ()
- zERROR SetTxtFilePath (char ∗pcTxtFilePath)
- zERROR PostProcess (float fTransitionWeight, float fMinimumClassTimeInS, float fAPrioriProbabilityOfMusic, float fMinimumEnergy, float ∗∗ppfRawResult=0)
- zERROR GetSizeOfIntermediateResult (zINT ∗piRows, zINT ∗piColumns)
- zERROR GetIntermediateResult (zFLOAT ∗∗pfIntermediateResult)
- zERROR SetIntermediateResult (zFLOAT ∗∗pfIntermediateResult, zINT iRows, zINT iColumns)
- int GetSizeOfResult ()
- zERROR GetResult (SegmentationResult_t ∗pstResult)

Static Public Member Functions

- static zERROR CreateInstance (CSegmentation ∗&pCSegmentation, int iSample-Rate, int iNumberOfChannels)
- static zERROR DestroyInstance (CSegmentation ∗&pCSegmentation)
- static const int GetVersion (const Version_t eVersionIdx)
- static const char ∗ GetBuildDate ()

Private Types

- enum ProcessBuffers_t_tag { k1, k2, k3, kNumProcessBuffers }
- enum FFTSizes_t_tag { kShort, kLong, kNumFFTSizes }
- enum Indices_t_tag { kStart, kStop, kNumIndices }
- enum Phases_t_tag { kPrevious, kCurrent, kNumPhaseStates }
- enum Features2Extract_t_tag { kLoudness = 0, kPeakSteadiness, kNoiseness, kNoiseness2, kNoiseness3, kNoiseness4, kRhythmicness, kCentroid, kSpread, kMidBandFlatness, kRollOff, kFlux, kMonoStrength, kMFCC, kNumFeatures = kMFCC + kNumMelCoeffs }
- enum SubFeatures_t_tag { kMean = 0, kStd, kDerivStd, kMax2Mean, kMax-Regularity, kNumSubFeatures }
- typedef enum CSegmentation::ProcessBuffers_t_tag ProcessBuffers_t
- typedef enum CSegmentation::FFTSizes_t_tag FftSizes_t
- typedef enum CSegmentation::Indices_t_tag Indices_t
- typedef enum CSegmentation::Phases_t_tag Phases_t
- typedef enum CSegmentation::Features2Extract_t_tag Features2Extract_t
- typedef enum CSegmentation::SubFeatures_t_tag SubFeatures_t

Private Member Functions

- zERROR CalcFreqTableMpeg7 ()
- zERROR CalcFlatnessFreqs ()
- zERROR CalcMFCCFilters (zINT iNumOfBands, zFLOAT fMinFreq, zFLOAT fMaxFreq)
- zERROR CalcMFCC (zFLOAT ∗pfResult)
- zFLOAT CalcMonoStrength ()
- zERROR LoadMatrixFromFile (zFLOAT ∗∗ppfMatrix, std::string acFilePath)
- zERROR CalcDTW (zFLOAT ∗∗ppfSimilarityMatrix, zFLOAT ∗∗ppfCostMatrix, zINT ∗piPathIdx, zFLOAT fTransitionCost, zINT iNumOfRows, zINT iNumOfColumns)
- zERROR RemoveLowEnergyFrames (zINT ∗piPath, zFLOAT ∗pfLoudness, zINT iLengthOPath, zFLOAT fLoudnessThreshold)
- zERROR ScaleLowEnergyFrames (zFLOAT ∗∗ppfDistances, zFLOAT ∗pfLoudness, zINT iLengthOfPath)
- zVOID CalcPitchSteadiness (zFLOAT ∗pfAudio, zFLOAT pfResult[3])
- zVOID CalcPitchBounds ()
- zVOID CalcPitchSteadinessIndices ()
- zINT PeakPicking (zFLOAT ∗pfMagSpectrum, zFLOAT ∗pfInstFreq, FftSizes_t eFftSize)
- zFLOAT RelativeNoisePower (zFLOAT ∗pfMagSpectrum, zFLOAT ∗pfInstFreq, zFLOAT ∗pfPeakSpectrum, FftSizes_t eFftSize)
- zFLOAT AbsoluteNoisePower (zFLOAT ∗pfMagSpectrum, zFLOAT ∗pfInstFreq, zFLOAT ∗pfPeakSpectrum, FftSizes_t eFftSize)
- zFLOAT CalcSpecSlope (zFLOAT ∗pfMagSpec, zINT iLength)
- zVOID CalcRhythmFeature ()
- zFLOAT CalcBackgroundNoisePower (zFLOAT ∗pfComplexSpectrum, FftSizes_t eFFTSize)

Private Attributes

- zFLOAT m_fSampleRate
- zFLOAT m_fMaxValue
- zFLOAT m_fStartTime
- zINT m_iNumberOfChannels
- zINT m_aiProcessBlockSize [kNumFFTSizes]
- zINT m_iProcessHopSizeInFrames
- zINT m_aiCurrentTimeIndex [kNumFFTSizes]
- zINT m_iFileLengthInFrames
- zINT m_iMpeg7StartIdx
- zINT m_iNumOfFeatureVectors
- zINT m_iCompleteNumOfFeatures
- zINT m_iTextureWindowLength
- zINT m_iTextureWindowHop
- zINT m_iRhythmWindowLength
- zINT m_aiFlatnessBounds [4]

- zINT m_iLengthOfResult
- zFLOAT32 ∗∗ m_ppfFeatures
- zFLOAT32 ∗ m_apfProcessBuffer [kNumProcessBuffers]
- zFLOAT32 ∗ m_apfPrevFFT [kNumFFTSizes]
- zFLOAT32 ∗ m_pfLogFreqs
- zFLOAT32 ∗ m_pfPreEmphasis
- zFLOAT32 ∗∗ m_ppfMFCCFilters
- zFLOAT32 ∗∗ m_ppfDCTCoeffs
- zFLOAT32 ∗ m_pfMFCCBuffer
- CRingBuffer< zFLOAT32 > ∗ m_pCRingBuffer
- CRingBuffer< zFLOAT32 > ∗ m_apCFeatureBuffer [kNumFeatures]
- CRingBuffer< zFLOAT32 > ∗ m_pCOnsetCurveBuffer
- CzplfFFT_If ∗ m_apFFTInstance [kNumFFTSizes]
- SegmentationResult_t ∗ m_pstResult
- std::string m_strTxtFilePath
- CParametricEqIf ∗ m_apfPreFilters [kNumPreFilters]
- zFLOAT m_fLowPassCoeff
- zFLOAT ∗ m_pfPeakSteadiness
- zFLOAT ∗ m_apfPhase [kNumPhaseStates]
- zFLOAT ∗ m_pfOmega
- zINT ∗ m_apiPitchBounds [kNumIndices]
- zINT m_aaiIndices [kNumFFTSizes][kNumIndices]
- zINT m_iNumOfPitches

6.3.1 Detailed Description

Definition at line 158 of file Segmentation.h.

6.3.2 Member Typedef Documentation

6.3.2.1 typedef enum CSegmentation::Features2Extract_t_tag CSegmentation::Features2Extract_t `[private]`

6.3.2.2 typedef enum CSegmentation::FFTSizes_t_tag CSegmentation::FftSizes_t `[private]`

6.3.2.3 typedef enum CSegmentation::Indices_t_tag CSegmentation::Indices_t `[private]`

6.3.2.4 typedef enum CSegmentation::Phases_t_tag CSegmentation::Phases_t `[private]`

6.3.2.5 typedef enum CSegmentation::ProcessBuffers_t_tag CSegmentation::ProcessBuffers_t `[private]`

6.3.2.6 typedef enum CSegmentation::SubFeatures_t_tag CSegmentation::SubFeatures_t `[private]`

6.3.3 Member Enumeration Documentation

6.3.3.1 enum CSegmentation::Features2Extract_t_tag [private]

Enumerator:

kLoudness

kPeakSteadiness

kNoiseness

kNoiseness2

kNoiseness3

kNoiseness4

kRhythmicness

kCentroid

kSpread

kMidBandFlatness

kRollOff

kFlux

kMonoStrength

kMFCC

kNumFeatures

Definition at line 251 of file Segmentation.h.

```
    {
        kLoudness   = 0,
//#ifdef V15ONLY
//        kStereoCorr,
//        kStereoMSLevel,
//#endif
        kPeakSteadiness,
#ifdef CLIP_DETECTION
        kClipping,
#endif
        kNoiseness,
        kNoiseness2,
        kNoiseness3,
        kNoiseness4,
#ifdef RHYTHM
        kRhythmicness,
#endif
        kCentroid,
        kSpread,
        kMidBandFlatness,
        kRollOff,
        kFlux,
        kMonoStrength,
        kMFCC,

        kNumFeatures    = kMFCC + kNumMelCoeffs
        //kNumFeatures
    } Features2Extract_t;
```

6.3.3.2 enum CSegmentation::FFTSizes_t_tag [private]

Enumerator:

kShort

kLong

kNumFFTSizes

Definition at line 206 of file Segmentation.h.

```
    {
        kShort,
        kLong,

        kNumFFTSizes,
    } FftSizes_t;
```

6.3.3.3 enum CSegmentation::Indices_t_tag [private]

Enumerator:

kStart

kStop

kNumIndices

Definition at line 214 of file Segmentation.h.

```
    {
        kStart,
        kStop,

        kNumIndices
    } Indices_t;
```

6.3.3.4 enum CSegmentation::Phases_t_tag [private]

Enumerator:

kPrevious

kCurrent

kNumPhaseStates

Definition at line 222 of file Segmentation.h.

```
    {
        kPrevious,
        kCurrent,

        kNumPhaseStates
    } Phases_t;
```

6.3.3.5 enum CSegmentation::ProcessBuffers_t_tag [private]

Enumerator:

k1

k2

k3

kNumProcessBuffers

Definition at line 197 of file Segmentation.h.

```
{
    k1,
    k2,
    k3,

    kNumProcessBuffers
} ProcessBuffers_t;
```

6.3.3.6 enum CSegmentation::SubFeatures_t_tag [private]

Enumerator:

kMean

kStd

kDerivStd

kMax2Mean

kMaxRegularity

kNumSubFeatures

Definition at line 281 of file Segmentation.h.

```
{
    kMean   = 0,
    kStd,
    kDerivStd,
    kMax2Mean,
    kMaxRegularity,

    kNumSubFeatures
} SubFeatures_t;
```

6.3.4 Constructor & Destructor Documentation

6.3.4.1 CSegmentation::CSegmentation (int *iSampleRate,* int *iNumberOfChannels*)

6.3.4.2 virtual CSegmentation::∼CSegmentation ( ) [virtual]

6.3.5 Member Function Documentation

6.3.5.1 zFLOAT CSegmentation::AbsoluteNoisePower (zFLOAT ∗ *pfMagSpectrum*, zFLOAT ∗ *pfInstFreq*, zFLOAT ∗ *pfPeakSpectrum*, FftSizes_t *eFftSize*) [private]

6.3.5.2 zFLOAT CSegmentation::CalcBackgroundNoisePower (zFLOAT ∗ *pfComplexSpectrum*, FftSizes_t *eFFTSize*) [private]

6.3.5.3 zERROR CSegmentation::CalcDTW (zFLOAT ∗∗ *ppfSimilarityMatrix*, zFLOAT ∗∗ *ppfCostMatrix*, zINT ∗ *piPathIdx*, zFLOAT *fTransitionCost*, zINT *iNumOfRows*, zINT *iNumOfColumns*) [private]

6.3.5.4 zERROR CSegmentation::CalcFlatnessFreqs ( ) [private]

6.3.5.5 zERROR CSegmentation::CalcFreqTableMpeg7 ( ) [private]

6.3.5.6 zERROR CSegmentation::CalcMFCC (zFLOAT ∗ *pfResult*) [private]

6.3.5.7 zERROR CSegmentation::CalcMFCCFilters (zINT *iNumOfBands*, zFLOAT *fMinFreq*, zFLOAT *fMaxFreq*) [private]

6.3.5.8 zFLOAT CSegmentation::CalcMonoStrength ( ) [private]

6.3.5.9 zVOID CSegmentation::CalcPitchBounds ( ) [private]

6.3.5.10 zVOID CSegmentation::CalcPitchSteadiness (zFLOAT ∗ *pfAudio*, zFLOAT *pfResult[3]*) [private]

6.3.5.11 zVOID CSegmentation::CalcPitchSteadinessIndices ( ) [private]

6.3.5.12 zVOID CSegmentation::CalcRhythmFeature ( ) [private]

6.3.5.13 zFLOAT CSegmentation::CalcSpecSlope (zFLOAT ∗ *pfMagSpec*, zINT *iLength*) [private]

6.3.5.14 static zERROR CSegmentation::CreateInstance (CSegmentation ∗& *pCSegmentation*, int *iSampleRate*, int *iNumberOfChannels*) [static]

6.3.5.15 static zERROR CSegmentation::DestroyInstance (CSegmentation ∗& *pCSegmentation*) [static]

6.3.5.16 zERROR CSegmentation::FinishProcess ( )

6.3.5.17 static const char∗ CSegmentation::GetBuildDate ( ) [static]

6.3.5.18 zERROR CSegmentation::GetIntermediateResult (zFLOAT ∗∗ *pfIntermediateResult*)

6.3.5.19 zERROR CSegmentation::GetResult (SegmentationResult_t ∗ *pstResult*)

6.3.5.20 zERROR CSegmentation::GetSizeOfIntermediateResult (zINT ∗ *piRows*, zINT ∗ *piColumns*)

6.3.5.21 int CSegmentation::GetSizeOfResult ( )

6.3.5.22 static const int CSegmentation::GetVersion (const Version_t *eVersionIdx*) `[static]`

6.3.5.23 zERROR CSegmentation::Initialize (int *iOverallInputFileLengthInFrames*)

6.3.5.24 zERROR CSegmentation::LoadMatrixFromFile (zFLOAT ∗∗ *ppfMatrix,* std::string *acFilePath*) `[private]`

6.3.5.25 zINT CSegmentation::PeakPicking (zFLOAT ∗ *pfMagSpectrum,* zFLOAT ∗ *pfInstFreq,* FftSizes_t *eFftSize*) `[private]`

6.3.5.26 zERROR CSegmentation::PostProcess (float *fTransitionWeight,* float *fMinimumClassTimeInS,* float *fAPrioriProbabilityOfMusic,* float *fMinimumEnergy,* float ∗∗ *ppfRawResult* = `0`)

6.3.5.27 zERROR CSegmentation::PreProcess (float ∗ *pfInputBufferInterleaved,* int *iNumberOfFrames*)

6.3.5.28 zERROR CSegmentation::Process (float ∗ *pfInputBufferInterleaved,* int *iNumberOfFrames*)

6.3.5.29 zFLOAT CSegmentation::RelativeNoisePower (zFLOAT ∗ *pfMagSpectrum,* zFLOAT ∗ *pfInstFreq,* zFLOAT ∗ *pfPeakSpectrum,* FftSizes_t *eFftSize*) `[private]`

6.3.5.30 zERROR CSegmentation::RemoveLowEnergyFrames (zINT ∗ *piPath,* zFLOAT ∗ *pfLoudness,* zINT *iLengthOPath,* zFLOAT *fLoudnessThreshold*) `[private]`

6.3.5.31 zERROR CSegmentation::ScaleLowEnergyFrames (zFLOAT ∗∗ *ppfDistances,* zFLOAT ∗ *pfLoudness,* zINT *iLengthOfPath*) `[private]`

6.3.5.32 zERROR CSegmentation::SetIntermediateResult (zFLOAT ∗∗ *pfIntermediateResult,* zINT *iRows,* zINT *iColumns*)

6.3.5.33 zERROR CSegmentation::SetTxtFilePath (char ∗ *pcTxtFilePath*)

6.3.6 Field Documentation

6.3.6.1 zINT CSegmentation::m_aaiIndices[kNumFFTSizes][kNumIndices] `[private]`

Definition at line 344 of file Segmentation.h.

6.3.6.2 zINT CSegmentation::m_aiCurrentTimeIndex[kNumFFTSizes] `[private]`

Definition at line 298 of file Segmentation.h.

6.3.6.3 zINT CSegmentation::m_aiFlatnessBounds[4] [private]

Definition at line 298 of file Segmentation.h.

6.3.6.4 zINT CSegmentation::m_aiProcessBlockSize[kNumFFTSizes] [private]

Definition at line 298 of file Segmentation.h.

6.3.6.5 CRingBuffer<zFLOAT32>* CSegmentation::m_apCFeatureBuffer[kNumFeatures] [private]

Definition at line 322 of file Segmentation.h.

6.3.6.6 CzplfFFT_If* CSegmentation::m_apFFTInstance[kNumFFTSizes] [private]

Definition at line 326 of file Segmentation.h.

6.3.6.7 zFLOAT * CSegmentation::m_apfPhase[kNumPhaseStates] [private]

Definition at line 340 of file Segmentation.h.

6.3.6.8 CParametricEqIf* CSegmentation::m_apfPreFilters[kNumPreFilters] [private]

Definition at line 333 of file Segmentation.h.

6.3.6.9 zFLOAT32 * CSegmentation::m_apfPrevFFT[kNumFFTSizes] [private]

Definition at line 313 of file Segmentation.h.

6.3.6.10 zFLOAT32 * CSegmentation::m_apfProcessBuffer[kNumProcessBuffers] [private]

Definition at line 313 of file Segmentation.h.

6.3.6.11 zINT* CSegmentation::m_apiPitchBounds[kNumIndices] [private]

Definition at line 343 of file Segmentation.h.

6.3.6.12 zFLOAT CSegmentation::m_fLowPassCoeff [private]

Definition at line 339 of file Segmentation.h.

6.3.6.13 zFLOAT CSegmentation::m_fMaxValue [private]

Definition at line 295 of file Segmentation.h.

6.3.6.14 zFLOAT CSegmentation::m_fSampleRate [private]

Definition at line 295 of file Segmentation.h.

6.3.6.15 zFLOAT CSegmentation::m_fStartTime [private]

Definition at line 295 of file Segmentation.h.

6.3.6.16 zINT CSegmentation::m_iCompleteNumOfFeatures [private]

Definition at line 298 of file Segmentation.h.

6.3.6.17 zINT CSegmentation::m_iFileLengthInFrames [private]

Definition at line 298 of file Segmentation.h.

6.3.6.18 zINT CSegmentation::m_iLengthOfResult [private]

Definition at line 298 of file Segmentation.h.

6.3.6.19 zINT CSegmentation::m_iMpeg7StartIdx [private]

Definition at line 298 of file Segmentation.h.

6.3.6.20 zINT CSegmentation::m_iNumberOfChannels [private]

Definition at line 298 of file Segmentation.h.

6.3.6.21 zINT CSegmentation::m_iNumOfFeatureVectors [private]

Definition at line 298 of file Segmentation.h.

6.3.6.22 zINT CSegmentation::m_iNumOfPitches [private]

Definition at line 344 of file Segmentation.h.

6.3.6.23 zINT CSegmentation::m_iProcessHopSizeInFrames [private]

Definition at line 298 of file Segmentation.h.

6.3.6.24 zINT CSegmentation::m_iRhythmWindowLength [private]

Definition at line 298 of file Segmentation.h.

6.3.6.25 zINT CSegmentation::m_iTextureWindowHop [private]

Definition at line 298 of file Segmentation.h.

6.3.6.26 zINT CSegmentation::m_iTextureWindowLength [private]

Definition at line 298 of file Segmentation.h.

6.3.6.27 CRingBuffer< zFLOAT32 >∗ CSegmentation::m_pCOnsetCurveBuffer [private]

Definition at line 324 of file Segmentation.h.

6.3.6.28 CRingBuffer< zFLOAT32 >∗ CSegmentation::m_pCRingBuffer [private]

Definition at line 321 of file Segmentation.h.

6.3.6.29 zFLOAT32 ∗ CSegmentation::m_pfLogFreqs [private]

Definition at line 313 of file Segmentation.h.

6.3.6.30 zFLOAT32 ∗ CSegmentation::m_pfMFCCBuffer [private]

Definition at line 313 of file Segmentation.h.

6.3.6.31 zFLOAT ∗ CSegmentation::m_pfOmega [private]

Definition at line 340 of file Segmentation.h.

6.3.6.32 zFLOAT∗ CSegmentation::m_pfPeakSteadiness [private]

Definition at line 340 of file Segmentation.h.

6.3.6.33 zFLOAT32 ∗ CSegmentation::m_pfPreEmphasis [private]

Definition at line 313 of file Segmentation.h.

6.3.6.34 zFLOAT32 ∗∗ CSegmentation::m_ppfDCTCoeffs [private]

Definition at line 313 of file Segmentation.h.

6.3.6.35 zFLOAT32∗∗ CSegmentation::m_ppfFeatures [private]

Definition at line 313 of file Segmentation.h.

6.3.6.36 zFLOAT32 ∗∗ CSegmentation::m_ppfMFCCFilters [private]

Definition at line 313 of file Segmentation.h.

6.3.6.37 SegmentationResult_t∗ CSegmentation::m_pstResult [private]

Definition at line 328 of file Segmentation.h.

6.3.6.38 std::string CSegmentation::m_strTxtFilePath [private]

Definition at line 330 of file Segmentation.h.

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

- Segmentation.h

6.4 CTraining Class Reference

```
#include <Training.h>
```

Public Member Functions

- CTraining (int iNumOfClasses, Classifier_t eClassifier)
- virtual ∼CTraining ()
- zError_t AppendFeatureData (zFLOAT ∗∗ppfFeatureMatrix, zINT ∗piMatrix-Dimensions, int iClassIdx)
- zError_t GetTrainingSetInfo (TrainingSetInfo_t ∗pInfo)
- Classifier_t GetClassifierType ()
- zError_t Process ()
- zError_t GetResultDimension (TrainingResults_t eResultIdx, int ∗piResultDimensions)
- zError_t GetResult (TrainingResults_t eResultIdx, float ∗∗ppfResult, const int ∗piResultDimensions)

Static Public Member Functions

- static zError_t CreateInstance (CTraining ∗&pCTraining, int iNumOfClasses=2, Classifier_t eClassifier=kClassifierLDA)
- static zError_t DestroyInstance (CTraining ∗&pCTraining)
- static const int GetVersion (const Version_t eVersionIdx)
- static const char ∗ GetBuildDate ()

Private Member Functions

- zError_t ProcessLda ()
- zError_t ProcessQda ()

Private Attributes

- Classifier_t m_eClassifier
- CMatrix ∗∗ m_ppCFeatureMatrix
- CMatrix ∗ m_apCResults [kNumOfTrainResults]
- zINT m_iNumOfClasses

6.4.1 Detailed Description

Definition at line 63 of file Training.h.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 CTraining::CTraining (int *iNumOfClasses*, Classifier_t *eClassifier*)

6.4.2.2 virtual CTraining::∼CTraining ( ) [virtual]

6.4.3 Member Function Documentation

6.4.3.1 zError_t CTraining::AppendFeatureData (zFLOAT ∗∗ *ppfFeatureMatrix*, zINT ∗ *piMatrixDimensions*, int *iClassIdx*)

6.4.3.2 static zError_t CTraining::CreateInstance (CTraining ∗& *pCTraining*, int *iNumOfClasses* = 2, Classifier_t *eClassifier* = kClassifierLDA) [static]

6.4.3.3 static zError_t CTraining::DestroyInstance (CTraining ∗& *pCTraining*) [static]

6.4.3.4 static const char∗ CTraining::GetBuildDate ( ) [static]

6.4.3.5 Classifier_t CTraining::GetClassifierType ( ) [inline]

Definition at line 80 of file Training.h.

References m_eClassifier.

```
{return m_eClassifier;};
```

6.4.3.6 zError_t CTraining::GetResult (TrainingResults_t *eResultIdx*, float ∗∗ *ppfResult*, const int ∗ *piResultDimensions*)

6.4.3.7 zError_t CTraining::GetResultDimension (TrainingResults_t *eResultIdx*, int ∗ *piResultDimensions*)

6.4.3.8 zError_t CTraining::GetTrainingSetInfo (TrainingSetInfo_t ∗ *pInfo*)

6.4.3.9 static const int CTraining::GetVersion (const Version_t *eVersionIdx*) [static]

6.4.3.10 zError_t CTraining::Process ( )

6.4.3.11 zError_t CTraining::ProcessLda ( ) [private]

6.4.3.12 zError_t CTraining::ProcessQda ( ) [private]

6.4.4 Field Documentation

6.4.4.1 CMatrix∗ CTraining::m_apCResults[kNumOfTrainResults] [private]

Definition at line 95 of file Training.h.

6.4.4.2 Classifier_t CTraining::m_eClassifier `[private]`

Definition at line 92 of file Training.h.

Referenced by GetClassifierType().

6.4.4.3 zINT CTraining::m_iNumOfClasses `[private]`

Definition at line 97 of file Training.h.

6.4.4.4 CMatrix∗∗ CTraining::m_ppCFeatureMatrix `[private]`

Definition at line 93 of file Training.h.

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

- Training.h

6.5 CTraining_If Class Reference

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

`#include <Training_If.h>`

Collaboration diagram for CTraining_If:

Public Member Functions

- CTraining_If (int iNumberOfClasses, Classifier_t eClassifier)
- virtual ∼CTraining_If ()
- zError_t SetParamDirectoryPaths (string ∗pstrPaths)

  sets the directory paths to the audio files per class used for training
- zError_t SetParamAudioFileExtension (string strAudioFileExtension)

  sets the extension of the searched audio files (default: .wav)
- zError_t SetParamOutFilePath (string strPath)

  sets the path where the output files (training results) will be written (default: working directory)
- zError_t SetTrainingData (float ∗∗ppfFeatures, int iClassIdx, int iNumOfObservations, int iNumOfFeatures)

  set new classifier training data
- zError_t CalculateFeatures ()

  calculate all features for the files in the previously defined directories
- zError_t Train ()

  after the calculation of features and setting all training data, finally do the training itself
- zError_t WriteTrainingResults ()

  retrieves the training results and writes them to the output files

Static Public Member Functions

- static zError_t CreateInstance (CTraining_If ∗&pCTraining_If, int iNumberOfClasses=2, Classifier_t eClassifier=kClassifierLDA)

  creates a new instance of CTraining_If
- static zError_t DestroyInstance (CTraining_If ∗&pCTraining_If)

  destroys an instance of CTraining_If

Private Member Functions

- zError_t ParseDirectory (string strPath, CFileList ∗pCFileList)

  parses a complete directory for files with a special extension and adds the file names to a list
- zError_t CalculateFeaturesAndAppendData4Training (string strFileName, int iClassIdx)

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

Private Attributes

- string m_strDirectoryPath [kMaxNumOfClasses]

  class directory paths
- string m_strFileExtension

  audio file extension to parse for
- string m_strOutFileNames [kNumOfTrainResults]

  output file names
- CFileList ∗ m_pCFileList [kMaxNumOfClasses]

  list of all audio files per class directory
- float ∗∗ m_ppfTmpFeatureMatrix

  internal memory for feature data
- void ∗ m_pCTrainingInstance

  training instance
- int m_aiMaxDimensionsOfFeatureMatrix [2]

  size of m_ppfTmpFeatureMatrix
- int m_iNumOfClasses

  number of classification classes

6.5.1 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 60 of file Training_If.h.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 CTraining_If::CTraining_If (int *iNumberOfClasses*, Classifier_t *eClassifier*)

Definition at line 83 of file Training_If.cpp.

References kDefaultAudioFileExtension, kNumOfTrainResults, m_aiMaxDimensionsOfFeatureMatrix, m_iNumOfClasses, m_pCFileList, m_pCTrainingInstance, m_ppfTmpFeatureMatrix, m_strDirectoryPath, m_strFileExtension, m_strOutFileNames, and Train_CreateInstance().

Referenced by CreateInstance().

```
{
    int i;

    for (i = 0; i < kNumOfTrainResults; i++)
    {
        m_strOutFileNames[i].erase ();
        m_strDirectoryPath[i].erase ();
    }

    m_iNumOfClasses = iNumberOfClasses;

    // initialize members and alloc memory
    for (i = 0; i < m_iNumOfClasses; i++)
    {
        m_strDirectoryPath[i].erase ();
        m_pCFileList[i] = 0;
        m_pCFileList[i] = new CFileList ();
    }

    for ( i = 0; i < kNumOfTrainResults; i++)
        m_strOutFileNames[i].erase ();

    m_ppfTmpFeatureMatrix   = 0;
    memset (m_aiMaxDimensionsOfFeatureMatrix, 0, sizeof(
      m_aiMaxDimensionsOfFeatureMatrix));

    // set extension to default value
    m_strFileExtension.assign (kDefaultAudioFileExtension);

    // create new instance of training class
    Train_CreateInstance (&m_pCTrainingInstance, iNumberOfClasses, eClassifier)
      ;
}
```

Here is the call graph for this function:

6.5.2.2 CTraining_If::∼CTraining_If ( ) [virtual]

Definition at line 117 of file Training_If.cpp.

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

```
{
    int i;

    // free allocated memory
    for (i = 0; i < m_iNumOfClasses; i++)
        delete m_pCFileList[i];

    hlpFreeMatrix (m_ppfTmpFeatureMatrix, m_aiMaxDimensionsOfFeatureMatrix);

    // destroy instance of training class
    Train_DestroyInstance (&m_pCTrainingInstance);
}
```

Here is the call graph for this function:

6.5.3 Member Function Documentation

6.5.3.1 zError_t CTraining_If::CalculateFeatures ( )

calculate all features for the files in the previously defined directories

Returns

0 if no errors

Definition at line 272 of file Training_If.cpp.

References CalculateFeaturesAndAppendData4Training(), CFileList::GetEntry(), C-FileList::GetNumOfEntries(), kMlNoError, kMlNothingToDo, m_iNumOfClasses, and m_pCFileList.

Referenced by main().

```
{
    int             iClass;
    zError_t rErr                = kMlNoError;

    // check if we really have data to train
    for (iClass = 0; iClass < m_iNumOfClasses; iClass++)
        if ((m_pCFileList[iClass]->GetNumOfEntries () == 0))
            return kMlNothingToDo;

    for (iClass = 0; iClass < m_iNumOfClasses; iClass++)
    {
        int iNumOfEntries   = m_pCFileList[iClass]->GetNumOfEntries ();

        // do feature extraction for all files
        for (int iFile = 0; iFile < iNumOfEntries; iFile++)
        {
            string CurrentFile;
            m_pCFileList[iClass]->GetEntry (CurrentFile, iFile);

            // do the feature calculation and add the features to the training
       set
            rErr    = this->CalculateFeaturesAndAppendData4Training (
      CurrentFile, iClass);
            if (rErr != kMlNoError)
                return rErr;
        }
    }

    return kMlNoError;
}
```

Here is the call graph for this function:

6.5.3.2 zError_t CTraining_If::CalculateFeaturesAndAppendData4Training (string *strFileName,* int *iClassIdx*) [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
iClassIdx	definition of class

Returns

0 if no error

See also

Segmentation_C.h|Training_C.h

Definition at line 356 of file Training_If.cpp.

References FeatEx_CreateInstance(), FeatEx_DestroyInstance(), FeatEx_FinishProcess(), FeatEx_GetFeatureMatrix(), FeatEx_GetSizeOfFeatureMatrix(), FeatEx_Initialize(), -FeatEx_Process(), hlpReallocMatrix(), kMlFileOpenFailed, kMlNoError, kNumOfAudio-Frames2Read, m_aiMaxDimensionsOfFeatureMatrix, m_pCTrainingInstance, m_ppf-TmpFeatureMatrix, and Train_AppendFeatureData().

Referenced by CalculateFeatures().

```
{
    zError_t rErr            = kMlNoError;
    CzplAudioFile   *pCInputFile                    = 0;            // input
       file handle
    void            *pCFeatureExtract               = 0;            // handle
       to segmentation instance
    bool            bReadNextFrame                  = true;
    int             aiFeatureMatrixDimensions[2]    = {0, 0};

    // open sound file
    pCInputFile = new CzplAudioFile();

    pCInputFile->OpenReadFile(strFileName.c_str (), kNumOfAudioFrames2Read);

    if (!pCInputFile->IsFileOpen())
    {
        cout << "Input file could not be opened!" << endl;
        delete pCInputFile;
        return kMlFileOpenFailed;
    }

    // create class instance and initialize it
    FeatEx_CreateInstance (&pCFeatureExtract, static_cast<int>(pCInputFile->
      GetSampleRate ()+.5), pCInputFile->GetNumOfChannels ());
    FeatEx_Initialize (pCFeatureExtract, pCInputFile->GetFileSize (), false);

    // optionally do pre-processing first to enable file normalization here...

    // now do the feature calculation
    while(bReadNextFrame)
    {
        float   afAudioData[kNumOfAudioFrames2Read<<1],
                aafAudioData[2][kNumOfAudioFrames2Read];

        // read data from file
        int iNumSamplesRead = pCInputFile->Read (&aafAudioData[0][0],
      kNumOfAudioFrames2Read);

        if(iNumSamplesRead < kNumOfAudioFrames2Read)
        {
            //memset (&afAudioData[iNumSamplesRead], 0,
       (kNumOfAudioSamples2Read - iNumSamplesRead) * sizeof(float));
            bReadNextFrame = false;
        }

        // copy data to interleaved
        int iNumChannels = pCInputFile->GetNumOfChannels ();
        for (int i = 0; i < kNumOfAudioFrames2Read; i++)
            for (int j = 0; j < iNumChannels; j++)
                afAudioData[(i*iNumChannels)+j]    = aafAudioData[j][i];

        //processing
        FeatEx_Process (pCFeatureExtract, afAudioData, iNumSamplesRead);

    } // process loop

    // finish processing
    FeatEx_FinishProcess (pCFeatureExtract);

    // now get the feature data

    // get required buffer size
    FeatEx_GetSizeOfFeatureMatrix (pCFeatureExtract, &aiFeatureMatrixDimensions
      [0], &aiFeatureMatrixDimensions[1]);

    // allocate temp memory if required
    rErr    = hlpReallocMatrix (m_ppfTmpFeatureMatrix,
      aiFeatureMatrixDimensions, m_aiMaxDimensionsOfFeatureMatrix);
    if (rErr != kMlNoError)
        return rErr;

    // get result
    FeatEx_GetFeatureMatrix (pCFeatureExtract, m_ppfTmpFeatureMatrix);

    // append result to training data
    rErr    = Train_AppendFeatureData (m_pCTrainingInstance,
      m_ppfTmpFeatureMatrix, aiFeatureMatrixDimensions, iClassIdx);
    if (rErr != kMlNoError)
        return rErr;

    // destroy feature calculation class
```

```
    FeatEx_DestroyInstance (&pCFeatureExtract);

    // close audio file
    pCInputFile->CloseFile ();
    delete pCInputFile;
    pCInputFile = 0;

    return kMlNoError;
}
```

Here is the call graph for this function:

6.5.3.3 zError_t CTraining_If::CreateInstance (CTraining_If ∗& *pCTraining_If*, int *iNumberOfClasses* = 2, Classifier_t *eClassifier* = kClassifierLDA) [static]

creates a new instance of CTraining_If

Parameters

pCTraining-_If	instance handle is written to here
iNumberOf-Classes	number of classes to train
eClassifier	index of classifier type to use

Returns

0 if no error

Definition at line 131 of file Training_If.cpp.

References CTraining_If(), kMlMemAllocFailed, and kMlNoError.

Referenced by main().

```
{
//    zError_t rErr    = kMlNoError;
    pCTraining_If = 0;

    // create class instance
    pCTraining_If = new CTraining_If (iNumberOfClasses, eClassifier);

    if (pCTraining_If == 0)
        return kMlMemAllocFailed;


    return kMlNoError;
}
```

Here is the call graph for this function:

6.5.3.4 zError_t CTraining_If::DestroyInstance (CTraining_If ∗& *pCTraining_If*) [static]

destroys an instance of CTraining_If

Parameters

pCTraining-_If	instance handle to be destroyed

Returns

0 if no error

Definition at line 147 of file Training_If.cpp.

References kMlInvalidPointer, and kMlNoError.

Referenced by main().

```
{
    if (pCTraining_If == 0)
        return kMlInvalidPointer;

    // destroy class instance
    delete pCTraining_If;
    pCTraining_If = 0;

    return kMlNoError;

}
```

6.5.3.5 zError_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 308 of file Training_If.cpp.

References CFileList::Add2List(), kMlInternalError, kMlNoError, and m_strFileExtension.

Referenced by SetParamDirectoryPaths().

```
{
#if defined (ZPLANE_WIN32) || defined (ZPLANE_WIN64)
    struct _finddata_t          CurrentFile;
    long                        hFile;
    string                      strWildCard;

    strWildCard.assign (strPath + "*" + m_strFileExtension);
    // find first file
    if( (hFile = _findfirst( strWildCard.c_str (), &CurrentFile )) == -1L )
        return kMlNoError;
    else
    {
        string CurrentFileName;

        // file found, add it to the list
        CurrentFileName.assign (CurrentFile.name);
        pCFileList->Add2List (strPath + CurrentFileName);

        // Find the rest of the files
        while( _findnext( hFile, &CurrentFile ) == 0 )
        {

            // file found, add it to the list
            CurrentFileName.assign (CurrentFile.name);
            pCFileList->Add2List (strPath + CurrentFileName);
        }

        // has to be called at the end
        _findclose( hFile );
    }

#else

    /* MM
     * need to implement this for gcc/clang
     */
    #warning "NEEDS TO BE IMPLEMENTED!"

    return kMlInternalError;

#endif

    return kMlNoError;
}
```

Here is the call graph for this function:

6.5.3.6 zError_t CTraining_If::SetParamAudioFileExtension (string *strAudioFileExtension*)

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

Parameters

strAudio-File-Extension	extension

Returns

0 if no error

Definition at line 187 of file Training_If.cpp.

References kMlNoError, and m_strFileExtension.

```
{
    m_strFileExtension.assign (strAudioFileExtension);

    return kMlNoError;
}
```

6.5.3.7 zError_t CTraining_If::SetParamDirectoryPaths (string * *pstrPaths*)

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

Parameters

pstrPaths	points to an array of N strings with N=iNumberOfClasses; each string contains the path to a directory that contains audio training data for one class

Remarks

the call of this function is mandatory

Returns

0 if no error

Definition at line 160 of file Training_If.cpp.

References kMlNoError, kSlash, m_iNumOfClasses, m_pCFileList, m_strDirectory-Path, and ParseDirectory().

Referenced by main().

```
{
    int i;

    zError_t rErr = kMlNoError;

    // set directory paths
    //m_strDirectoryPath[kClassMusic].assign (strPathToMusicDir);
    //m_strDirectoryPath[kClassNoMusic].assign (strPathToNonMusicDir);

    // generate file lists
    for (i = 0; i < m_iNumOfClasses; i++)
    {
        m_strDirectoryPath[i].assign (pstrPaths[i]);
        // check if we have a slash
        if (m_strDirectoryPath[i].compare (m_strDirectoryPath[i].length ()-1, 1
      , kSlash) != 0)
            m_strDirectoryPath[i]  += kSlash;

        // now list all audio files
        rErr    = this->ParseDirectory (m_strDirectoryPath[i], m_pCFileList[i])
      ;
        if (rErr != kMlNoError)
            return rErr;
    }
    return rErr;
}
```

Here is the call graph for this function:

6.5.3.8 zError_t CTraining_If::SetParamOutFilePath (string *strPath*)

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

Parameters

strPath	output file path (directory)

Returns

0 if no error

Definition at line 194 of file Training_If.cpp.

References kClassifierLDA, kClassifierQDA, kMlNoError, kNumOfTrainResults, k-Slash, kTrainResultDet, m_pCTrainingInstance, m_strOutFileNames, pkTrainClass-Prefixes, pkTrainResultsFileNames, and Train_GetClassifierType().

Referenced by main().

```
{
    Classifier_t    eClassifierType = Train_GetClassifierType (
      m_pCTrainingInstance);

    // assign directory names
    for (int i = 0; i < kNumOfTrainResults; i++)
    {
        if ((eClassifierType == kClassifierLDA) && (i >= kTrainResultDet))
            continue;

        m_strOutFileNames[i].assign (strPath);

        // check if we have a slash
        if (m_strOutFileNames[i].compare (m_strOutFileNames[i].length ()-1, 1,
      kSlash) != 0)
            m_strOutFileNames[i]  += kSlash;

        // now set the file names
        m_strOutFileNames[i].append ((eClassifierType == kClassifierLDA)?
      pkTrainClassPrefixes[kClassifierLDA] : pkTrainClassPrefixes[kClassifierQDA]);
        m_strOutFileNames[i].append (pkTrainResultsFileNames[i]);
    }

    return kMlNoError;
}
```

Here is the call graph for this function:

6.5.3.9 zError_t CTraining_If::SetTrainingData (float ** *ppfFeatures*, int *iClassIdx*, int *iNumOfObservations*, int *iNumOfFeatures*)

set new classifier training data

Parameters

ppfFeatures	feature data (dimensions: [iNumOfFeatures]x[iNumOfObservations])
iClassIdx	class index
iNumOf-Observations	number of observations
iNumOf-Features	number of features

Returns

0 if no error

Definition at line 441 of file Training_If.cpp.

References m_pCTrainingInstance, and Train_AppendFeatureData().

Referenced by main().

```
{
    int aiDimension[2]  = {iNumOfFeatures, iNumOfObservations};

    return Train_AppendFeatureData (m_pCTrainingInstance, ppfFeatures,
      aiDimension, iClassIdx);

}
```

Here is the call graph for this function:

6.5.3.10 zError_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 302 of file Training_If.cpp.

References m_pCTrainingInstance, and Train_Process().

Referenced by main().

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

Here is the call graph for this function:

6.5.3.11 zError_t CTraining_If::WriteTrainingResults ( )

retrieves the training results and writes them to the output files

Returns

0 if now error

Definition at line 218 of file Training_If.cpp.

References kMlFileOpenFailed, kMlNoError, kNumOfTrainResults, m_pCTraining-Instance, m_strOutFileNames, Train_GetResult(), and Train_GetResultDimension().

Referenced by main().

```
{

    // do for all results
    for (int r = 0; r < kNumOfTrainResults; r++)
    {
        float           **ppfOutData        = 0;
        int              i,
                         aiOutDataDimensions[2] = {0,0};
        std::ofstream   aFFileHandle;

        if (m_strOutFileNames[r].empty ())
            return kMlFileOpenFailed;

        // get output dimensions
        Train_GetResultDimension (m_pCTrainingInstance, (TrainingResults_t)r,
      aiOutDataDimensions);

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

        // get output data
        Train_GetResult (m_pCTrainingInstance, (TrainingResults_t)r, ppfOutData
      , aiOutDataDimensions);

        // open output file
        aFFileHandle.open (m_strOutFileNames[r].c_str (), std::ios::out);

        if (aFFileHandle.is_open ())
        {
            // write file
            for (i = 0; i < aiOutDataDimensions[0]; i++)   // rows
            {
                for (int j = 0; j < aiOutDataDimensions[1]; j++)   // cols
                    aFFileHandle << std::scientific << std::setprecision(10) <<
       ppfOutData[i][j] << "\t";
```

```
                aFFileHandle << std::endl;
            }
        }

        // close file
        if (aFFileHandle.is_open ())
            aFFileHandle.close ();

        // free memory
        for (i = 0; i < aiOutDataDimensions[0]; i++)
            delete [] ppfOutData[i];
        delete [] ppfOutData;
        ppfOutData = 0;
    }

    return kMlNoError;
}
```

Here is the call graph for this function:

6.5.4 Field Documentation

6.5.4.1 int CTraining_If::m_aiMaxDimensionsOfFeatureMatrix[2] [private]

size of m_ppfTmpFeatureMatrix

Definition at line 232 of file Training_If.h.

Referenced by CalculateFeaturesAndAppendData4Training(), CTraining_If(), and ∼-CTraining_If().

6.5.4.2 int CTraining_If::m_iNumOfClasses [private]

number of classification classes

Definition at line 233 of file Training_If.h.

Referenced by CalculateFeatures(), CTraining_If(), SetParamDirectoryPaths(), and ∼-CTraining_If().

6.5.4.3 CFileList∗ CTraining_If::m_pCFileList[kMaxNumOfClasses] [private]

list of all audio files per class directory

Definition at line 227 of file Training_If.h.

Referenced by CalculateFeatures(), CTraining_If(), SetParamDirectoryPaths(), and ∼-CTraining_If().

6.5.4.4 void∗ CTraining_If::m_pCTrainingInstance [private]

training instance

Definition at line 230 of file Training_If.h.

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

6.5.4.5 float∗∗ CTraining_If::m_ppfTmpFeatureMatrix `[private]`

internal memory for feature data

Definition at line 229 of file Training_If.h.

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

6.5.4.6 string CTraining_If::m_strDirectoryPath[kMaxNumOfClasses] `[private]`

class directory paths

Definition at line 224 of file Training_If.h.

Referenced by CTraining_If(), and SetParamDirectoryPaths().

6.5.4.7 string CTraining_If::m_strFileExtension `[private]`

audio file extension to parse for

Definition at line 224 of file Training_If.h.

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

6.5.4.8 string CTraining_If::m_strOutFileNames[kNumOfTrainResults] `[private]`

output file names

Definition at line 224 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

6.6 FeatureSimilarityResult_t_tag Struct Reference

`#include <FeatureSimilarity_C.h>`

Data Fields

- float fMaxLikelihood
- int iFeatureMatrixIndex

6.6.1 Detailed Description

Definition at line 49 of file FeatureSimilarity_C.h.

6.6.2 Field Documentation

6.6.2.1 float FeatureSimilarityResult_t_tag::fMaxLikelihood

Definition at line 51 of file FeatureSimilarity_C.h.

6.6.2.2 int FeatureSimilarityResult_t_tag::iFeatureMatrixIndex

Definition at line 52 of file FeatureSimilarity_C.h.

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

- FeatureSimilarity_C.h

6.7 SegmentationResult_t_tag Struct Reference

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

Data Fields

- float iStartTimeInS

  segment start
- float iStopTimeInS

  segment stop
- SegmentationClasses_t eEstimatedClass

  class
- float fEstimationReliability

  estimate of result reliability

6.7.1 Detailed Description

declaration of result structure

Definition at line 24 of file Segmentation_C.h.

6.7.2 Field Documentation

6.7.2.1 SegmentationClasses_t SegmentationResult_t_tag::eEstimatedClass

class

Definition at line 29 of file Segmentation_C.h.

6.7.2.2 float SegmentationResult_t_tag::fEstimationReliability

estimate of result reliability

Definition at line 31 of file Segmentation_C.h.

6.7.2.3 float SegmentationResult_t_tag::iStartTimeInS

segment start

Definition at line 26 of file Segmentation_C.h.

6.7.2.4 float SegmentationResult_t_tag::iStopTimeInS

segment stop

Definition at line 26 of file Segmentation_C.h.

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

- Segmentation_C.h

6.8 TrainingSetInfo_t_tag Struct Reference

info structure on the training data

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

Data Fields

- float afPrior [kMaxNumOfClasses]

  probability of class in the training set
- int iNumOfFeatures

  number of features
- int iOverallNumOfObservations

  number of observations for all classes

6.8.1 Detailed Description

info structure on the training data

Definition at line 59 of file Training_C.h.

6.8.2 Field Documentation

6.8.2.1 float TrainingSetInfo_t_tag::afPrior[kMaxNumOfClasses]

probability of class in the training set

Definition at line 61 of file Training_C.h.

6.8.2.2 int TrainingSetInfo_t_tag::iNumOfFeatures

number of features

Definition at line 62 of file Training_C.h.

6.8.2.3 int TrainingSetInfo_t_tag::iOverallNumOfObservations

number of observations for all classes

Definition at line 62 of file Training_C.h.

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

- Training_C.h

7 File Documentation

7.1 Classification_C.h File Reference

C-wrapper for the CClassification class.

`#include "Globals.h"` Include dependency graph for Classification_C.h:

Data Structures

- struct ClassificationResult_t_tag

Typedefs

- typedef struct ClassificationResult_t_tag ClassificationResult_t

Functions

- zError_t Class_CreateInstance (void ∗∗pphClassificationHandle, Classifier_t eClassifier=kClassifierLDA, float fHopSizeInS=.5F, float fWindowSizeInS=2.0F)
- zError_t Class_DestroyInstance (void ∗∗pphClassificationHandle)
- const int Class_GetVersion (const Version_t eVersionIdx)
- const char ∗ Class_GetBuildDateString ()
- zError_t Class_SetParamPath2TrainResults (void ∗phClassificationHandle, char ∗pcTxtFilePath)
- zError_t Class_SetParamTrainResults (void ∗phClassificationHandle, float ∗∗ppfTrainResult, int iNumRows, int iNumCols, TrainingResults_t eTrainResult)
- zError_t Class_CalcSegments (void ∗phClassificationHandle, bool bUseAPosterioriProbability=false)
- zError_t Class_RawClassification (void ∗phClassificationHandle, float ∗pfUnNormedResults, float ∗pfObservation, int iLengthOfObservation, bool bUseAPosterioriProbability=false)

- zError_t Class_RawPostProcess (void ∗phClassificationHandle, float ∗∗ppfProbabilities, int iNumOfProbabilities, bool bUseAPosterioriProbability=false)
- int Class_GetSizeOfResult (void ∗phClassificationHandle)
- zError_t Class_GetResult (void ∗phClassificationHandle, ClassificationResult_t ∗pstResult)
- zError_t Class_SetFeatureMatrix (void ∗phClassificationHandle, float ∗∗ppfFeatureMatrix, int iRows, int iColumns, float fFirstTimeStampInS=0, bool bDontAllocInternalMemory=false)
- zError_t Class_SetParamAPrioriProbabilities (void ∗phClassificationHandle, float ∗pfAPrioriProbabilities, int iNumOfClasses)
- zError_t Class_SetParamTransitionWeight (void ∗phClassificationHandle, float fTransitionWeight=.6F)
- zError_t Class_SetParamMinClassTime (void ∗phClassificationHandle, float fClassTimeInS=1.0F)
- zError_t Class_SetParamSetClassificationByFeatureLThresh (void ∗phClassificationHandle, int iFeatureIdx=0, float fFeatureThresh=.1F, int iClassIdx=0, float fClassProb=.8F)

7.1.1 Detailed Description

C-wrapper for the CClassification class. :

Definition in file Classification_C.h.

7.1.2 Typedef Documentation

7.1.2.1 typedef struct ClassificationResult_t_tag ClassificationResult_t

declaration of result structure

7.1.3 Function Documentation

7.1.3.1 zError_t Class_CalcSegments (void ∗ *phClassificationHandle*, bool *bUseAPosterioriProbability* = `false`)

calculates result (combined call of Class_RawClassification and Class_CalcPostProcess)

Parameters

phClassificationHandle	: handle to the instance
bUseAPosterioriProbability	: bool indicating if for internal processing the a posteriori probability should be used (true) or the likelihood should be used (false)

Returns

int : 0 if no error

7.1.3.2 zError_t Class_CreateInstance (void ** *pphClassificationHandle*, Classifier_t *eClassifier* = kClassifierLDA, float *fHopSizeInS* = `.5F`, float *fWindowSizeInS* = `2.0F`)

Creates a new instance of the segmentation lib

Parameters

pph-Classification-Handle	: handle to the new instance
eClassifier	: classifier used (see Classifier_t)
fHopSizeInS	: hop size in seconds used by feature calculation
fWindow-SizeInS	: texture window size in seconds used by feature calculation

Returns

int : 0 if no error

7.1.3.3 zError_t Class_DestroyInstance (void ** *pphClassificationHandle*)

destroys a previously created instance of the segmentation lib

Parameters

pph-Classification-Handle	: handle to the instance

Returns

int : 0 if no error

7.1.3.4 const char* Class_GetBuildDateString ( )

returns a string containing the build date of this library

Returns

char* : string

7.1.3.5 zError_t Class_GetResult (void * *phClassificationHandle*, ClassificationResult_t * *pstResult*)

copies the result to pstResult

Parameters

ph-Classification-Handle	: handle to the instance
∗pstResult	: pointer to allocated memory where the result can be copied into

Returns

int : 0 if no error

7.1.3.6 int Class_GetSizeOfResult (void ∗ *phClassificationHandle*)

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

Parameters

ph-Classification-Handle	: handle to the instance

Returns

int : number of results

7.1.3.7 const int Class_GetVersion (const Version_t *eVersionIdx*)

returns a string containing the version number of this library

Returns

char∗ : string

7.1.3.8 zError_t Class_RawClassification (void ∗ *phClassificationHandle,* float ∗ *pfUnNormedResults,* float ∗ *pfObservation,* int *iLengthOfObservation,* bool *bUseAPosterioriProbability* = `false`)

calculates result for one observation (no normalization, no thresholding, etc...)

Parameters

ph-Classification-Handle	: handle to the instance
pfUn-Normed-Results	: result of dimension kNumClasses is written to this buffer (note that the result is "flipped" --> the lower the value, the higher the probability)
pf-Observation	: input feature vector of length kNumFeatures

iLengthOf-Observation	: has to equal kNumFeatures
bUseA-Posteriori-Probability	: bool indicating if for internal processing the a posteriori probability should be used (true) or the likelihood should be used (false)

Returns

int : 0 if no error

7.1.3.9 zError_t Class_RawPostProcess (void * *phClassificationHandle,* float ** *ppfProbabilities,* int *iNumOfProbabilities,* bool *bUseAPosterioriProbability =* `false`)

calculates the final classification after the probabilities have been computed (e.g. by Class_RawClassification)

Parameters

ph-Classification-Handle	: handle to the instance
ppf-Probabilities	: pointer to matrix that contains the results of Class_RawClassification (dimension: classes x observations)
iNumOf-Probabilities	:number of columns in matrix ppfProbabilities (equals number of observations)
bUseA-Posteriori-Probability	: bool indicating if for internal processing the a posteriori probability has been used

Returns

int : 0 if no error

7.1.3.10 zError_t Class_SetFeatureMatrix (void * *phClassificationHandle,* float ** *ppfFeatureMatrix,* int *iRows,* int *iColumns,* float *fFirstTimeStampInS =* `0`, bool *bDontAllocInternalMemory =* `false`)

sets the internal feature data

Parameters

ph-Classification-Handle	: handle to the instance
***ppf-Feature-Matrix*	: two dimensional matrix with intermediate results

iRows	: number of rows of matrix
iColumns	: number of columns of matrix
fFirstTime-StampInS	: time stamp indicating the start time of the block the first feature has been calculated from
bDontAlloc-Internal-Memory	: flag (should the processing be done on the externally allocated memory, or should the matrix be copied

Returns

int : 0 if no error

7.1.3.11 zError_t Class_SetParamAPrioriProbabilities (void * *phClassificationHandle,* float * *pfAPrioriProbabilities,* int *iNumOfClasses*)

sets the a priori probabilities of each class for the classification process

Parameters

ph-Classification-Handle	: handle to the instance
**pfAPriori-Probabilities*	: a priori probabilities of all classes (sum has to equal 1.0F)
iNumOf-Classes	: length of buffer pfAPrioriProbabilities

Returns

int : 0 if no error

7.1.3.12 zError_t Class_SetParamMinClassTime (void * *phClassificationHandle,* float *fClassTimeInS* = `1.0F`)

sets the minimum time to be spend in one class (for succeeding blocks of features)

Parameters

ph-Classification-Handle	: handle to the instance
**fClass-TimeInS*	: discard class decisions that lead to a class time in seconds that is lower than this value

Returns

int : 0 if no error

7.1.3.13 zError_t Class_SetParamPath2TrainResults (void * *phClassificationHandle*, char * *pcTxtFilePath*)

set path to input txt files (means.txt and scale.txt); call alternative to Class_SetParam-TrainResults

Parameters

ph-Classification-Handle	: handle to the instance
pcTxtFile-Path	: path

Returns

int : 0 if no error

7.1.3.14 zError_t Class_SetParamSetClassificationByFeatureLThresh (void * *phClassificationHandle*, int *iFeatureIdx* = 0, float *fFeatureThresh* = .1F, int *iClassIdx* = 0, float *fClassProb* = .8F)

sets the feature and values that force a classification result for all feature values below this threshold

Parameters

ph-Classification-Handle	: handle to the instance
iFeatureIdx	: the index of the feature to be used
fFeature-Thresh	: set the class probability for all blocks where the feature is lower than this threshold
iClassIdx	: set this class to this special result
fClassProb	: probability the class with iClassIdx is set to, the other classes are set to (1-fClassProb)/NumOfClasses

Returns

int : 0 if no error

7.1.3.15 zError_t Class_SetParamTrainResults (void * *phClassificationHandle*, float ** *ppfTrainResult*, int *iNumRows*, int *iNumCols*, TrainingResults_t *eTrainResult*)

set train result matrixes (means and scale); call alternative to Class_SetParamPath2-TrainResults; it is imparative to call this function with TrainingResults_t::kTrainResult-Means first to ensure correct internal initialization

Parameters

ph-Classification-Handle	: handle to the instance
ppfTrain-Result	: matrix containing the training result as specified in eTrainResult, matrix dimensions ([rows][cols])
iNumRows	: number of matrix rows
iNumCols	: number of matrix cols
eTrainResult	: which result is currently handed over (see TrainingResults_t)

Returns

int : 0 if no error

7.1.3.16 zError_t Class_SetParamTransitionWeight (void ∗ *phClassificationHandle,* float *fTransitionWeight* = `.6F`)

sets the internal cost of jumping from one class to another

Parameters

ph-Classification-Handle	: handle to the instance
∗*f-Transition-Weight*	: make the transition between classes more difficult by applying this weight

Returns

int : 0 if no error

7.2 CMakeLists.txt File Reference

Functions

- set (APP TrainingCl) include(general) include(general-options) include(includes) file(GLOB $

7.2.1 Function Documentation

7.2.1.1 set (APP *TrainingCl*)

Definition at line 1 of file CMakeLists.txt.

```
{APP}_SOURCES RELATIVE ${CMAKE_SOURCE_DIR}/${APP} *.cpp)
```

7.3 FeatureExtraction_C.h File Reference

C-wrapper for the CFeatureExtraction class.

`#include "Globals.h"` Include dependency graph for FeatureExtraction_C.h: This graph shows which files directly or indirectly include this file:

Functions

- zError_t FeatEx_CreateInstance (void ∗∗pphFeatureExtractionHandle, int iSampleRate, int iNumberOfChannels, float fWindowLengthInS=2.0F, float fHopLengthInS=0.5F)
- zError_t FeatEx_DestroyInstance (void ∗∗pphFeatureExtractionHandle)
- const int FeatEx_GetVersion (const Version_t eVersionIdx)
- const char ∗ FeatEx_GetBuildDateString ()
- float FeatEx_GetFeatureHopSizeInS (void ∗phFeatureExtractionHandle)
- float FeatEx_GetFeatureWindowSizeInS (void ∗phFeatureExtractionHandle)
- int FeatEx_GetNumOfFeatures (void ∗phFeatureExtractionHandle)
- const char ∗ FeatEx_GetMainFeatureName (void ∗phFeatureExtractionHandle, int iFeatureIdx)
- const char ∗ FeatEx_GetSubFeatureName (void ∗phFeatureExtractionHandle, int iFeatureIdx)
- zError_t FeatEx_Initialize (void ∗phFeatureExtractionHandle, int iOverallInputFileLengthInFrames, bool bStoreIntermediateResults)
- zError_t FeatEx_PreProcess (void ∗phFeatureExtractionHandle, float ∗pfInputBufferInterleaved, int iNumberOfFrames)
- zError_t FeatEx_Process (void ∗phFeatureExtractionHandle, float ∗pfInputBufferInterleaved, int iNumberOfFrames)
- zError_t FeatEx_FinishProcess (void ∗phFeatureExtractionHandle)
- zError_t FeatEx_GetSizeOfFeatureMatrix (void ∗phFeatureExtractionHandle, int ∗piRows, int ∗piColumns)
- zError_t FeatEx_GetFeatureMatrix (void ∗phFeatureExtractionHandle, float ∗∗ppfFeatureMatrix)
- float ∗∗ FeatEx_GetFeatureMatrixPointer (void ∗phFeatureExtractionHandle)
- zError_t FeatEx_GetSizeOfIntermediateResult (void ∗phFeatureExtractionHandle, int ∗piRows, int ∗piColumns)
- zError_t FeatEx_GetIntermediateResult (void ∗phFeatureExtractionHandle, float ∗∗ppfFeatureMatrix)
- float ∗∗ FeatEx_GetIntermediateResultPointer (void ∗phFeatureExtractionHandle)
- float FeatEx_GetFirstTimeStamp (void ∗phFeatureExtractionHandle)

7.3.1 Detailed Description

C-wrapper for the CFeatureExtraction class. :

Definition in file FeatureExtraction_C.h.

7.3.2 Function Documentation

7.3.2.1 zError_t FeatEx_CreateInstance (void ** *pphFeatureExtractionHandle*, int *iSampleRate*, int *iNumberOfChannels*, float *fWindowLengthInS* = `2.0F`, float *fHopLengthInS* = `0.5F`)

Creates a new instance of the segmentation lib

Parameters

pphFeature-Extraction-Handle	: handle to the new instance
iSampleRate	: sample rate of audio file
iNumberOf-Channels	: number of audio channels
fWindow-LengthInS	: length of window in seconds for each observation (must be multiple of 0.01s, must be longer than fHopLengthInS)
fHopLength-InS	: length of window hop in seconds (must be multiple of 0.01s)

Returns

int : 0 if no error

Referenced by CTraining_If::CalculateFeaturesAndAppendData4Training().

7.3.2.2 zError_t FeatEx_DestroyInstance (void ** *pphFeatureExtractionHandle*)

destroys a previously created instance of the segmentation lib

Parameters

pphFeature-Extraction-Handle	: handle to the instance

Returns

int : 0 if no error

Referenced by CTraining_If::CalculateFeaturesAndAppendData4Training().

7.3.2.3 zError_t FeatEx_FinishProcess (void * *phFeatureExtractionHandle*)

called to signal there is no more audio data available

Parameters

phFeature-Extraction-Handle	: handle to the instance

Returns

int : 0 if no error

Referenced by CTraining_If::CalculateFeaturesAndAppendData4Training().

7.3.2.4 const char∗ FeatEx_GetBuildDateString ( )

returns a string containing the build date of this library

Returns

char∗ : string

7.3.2.5 float FeatEx_GetFeatureHopSizeInS (void ∗ *phFeatureExtractionHandle*)

returns the feature hopsize in seconds

Parameters

phFeature-Extraction-Handle	: handle to the instance

Returns

float : feature hopsize in seconds

7.3.2.6 zError_t FeatEx_GetFeatureMatrix (void ∗ *phFeatureExtractionHandle,* float ∗∗ *ppfFeatureMatrix*)

copies the internal intermediate feature data

Parameters

phFeature-Extraction-Handle	: handle to the instance
∗∗*ppf-Feature-Matrix*	: two-dimensional matrix with the size from FeatEx_GetSizeOf-FeatureMatrix

Returns

int : 0 if no error

Referenced by CTraining_If::CalculateFeaturesAndAppendData4Training().

7.3.2.7 float** FeatEx_GetFeatureMatrixPointer (void * *phFeatureExtractionHandle*)

returns the pointer to the internal intermediate feature data

Parameters

phFeature-Extraction-Handle	: handle to the instance

Returns

float** : pointer to result (do *not* free!)

7.3.2.8 float FeatEx_GetFeatureWindowSizeInS (void * *phFeatureExtractionHandle*)

returns the feature windowsize in seconds

Parameters

phFeature-Extraction-Handle	: handle to the instance

Returns

float : feature windowsize in seconds

7.3.2.9 float FeatEx_GetFirstTimeStamp (void * *phFeatureExtractionHandle*)

returns the timestamp for the first observation in seconds

Parameters

phFeature-Extraction-Handle	: handle to the instance

Returns

float : first timestamp

7.3.2.10 zError_t FeatEx_GetIntermediateResult (void * *phFeatureExtractionHandle,* float ** *ppfFeatureMatrix*)

returns the internal feature data

Parameters

phFeature-Extraction-Handle	: handle to the instance
***ppf-Feature-Matrix*	: two-dimensional matrix with the size from FeatEx_GetSizeOf-FeatureMatrix

Returns

int : 0 if no error

7.3.2.11 float** FeatEx_GetIntermediateResultPointer (void * *phFeatureExtractionHandle*)

returns the pointer to the internal intermediate feature data

Parameters

phFeature-Extraction-Handle	: handle to the instance

Returns

float** : pointer to result (do *not* free!)

7.3.2.12 const char* FeatEx_GetMainFeatureName (void * *phFeatureExtractionHandle,* int *iFeatureIdx*)

returns the name of a main feature

Parameters

phFeature-Extraction-Handle	: handle to the instance
iFeatureIdx	: feature index

Returns

const char : main feature name

7.3.2.13 int FeatEx_GetNumOfFeatures (void ∗ *phFeatureExtractionHandle*)

returns the overall number of features per observation

Parameters

phFeature-Extraction-Handle	: handle to the instance

Returns

int : main feature name

7.3.2.14 zError_t FeatEx_GetSizeOfFeatureMatrix (void ∗ *phFeatureExtractionHandle,* int ∗ *piRows,* int ∗ *piColumns*)

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

Parameters

phFeature-Extraction-Handle	: 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

Referenced by CTraining_If::CalculateFeaturesAndAppendData4Training().

7.3.2.15 zError_t FeatEx_GetSizeOfIntermediateResult (void ∗ *phFeatureExtractionHandle,* int ∗ *piRows,* int ∗ *piColumns*)

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

Parameters

phFeature-Extraction-Handle	: 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

7.3.2.16 const char∗ FeatEx_GetSubFeatureName (void ∗ *phFeatureExtractionHandle,* int *iFeatureIdx*)

returns the name of a sub feature

Parameters

phFeature-Extraction-Handle	: handle to the instance
iFeatureIdx	: feature index

Returns

const char : sub feature name

7.3.2.17 const int FeatEx_GetVersion (const Version_t *eVersionIdx*)

returns a string containing the version number of this library

Returns

char∗ : string

7.3.2.18 zError_t FeatEx_Initialize (void ∗ *phFeatureExtractionHandle,* int *iOverallInputFileLengthInFrames,* bool *bStoreIntermediateResults*)

initializes an instance of the segmentation lib

Parameters

phFeature-Extraction-Handle	: handle to the instance
iOverall-InputFile-LengthIn-Frames	: number of audio frames
bStore-Intermediate-Results	: bool to indicate access is required to the intermediate results (keep at false if you don't know what this means)

Returns

int : 0 if no error

Referenced by CTraining_If::CalculateFeaturesAndAppendData4Training().

7.3.2.19 zError_t FeatEx_PreProcess (void ∗ *phFeatureExtractionHandle,* float ∗ *pfInputBufferInterleaved,* int *iNumberOfFrames*)

preprocessing loop of the audio data

Parameters

phFeature-Extraction-Handle	: handle to the instance
∗pfInput-Buffer-Interleaved	: audio input data in pcm interleaved format
iNumberOf-Frames	: number of audio frames in buffer

Returns

int : 0 if no error

7.3.2.20 zError_t FeatEx_Process (void ∗ *phFeatureExtractionHandle,* float ∗ *pfInputBufferInterleaved,* int *iNumberOfFrames*)

processes the audio data

Parameters

phFeature-Extraction-Handle	: handle to the instance
∗pfInput-Buffer-Interleaved	: audio input data in pcm interleaved format
iNumberOf-Frames	: number of audio frames in buffer

Returns

int : 0 if no error

Referenced by CTraining_If::CalculateFeaturesAndAppendData4Training().

7.4 FeatureSimilarity_C.h File Reference

C-wrapper for the CFeatureSimilarity class.

`#include "Globals.h"` Include dependency graph for FeatureSimilarity_C.h:

Data Structures

- struct FeatureSimilarityResult_t_tag

Typedefs

- typedef struct FeatureSimilarityResult_t_tag FeatureSimilarityResult_t

Functions

- zError_t FeatSim_CreateInstance (void ∗∗pphFeatureSimilarityHandle)
- zError_t FeatSim_DestroyInstance (void ∗∗pphFeatureSimilarityHandle)
- const int FeatSim_GetVersion (const Version_t eVersionIdx)
- const char ∗ FeatSim_GetBuildDateString ()
- zError_t FeatSim_Initialize (void ∗phFeatureSimilarityHandle, float ∗pfMeans, int aiDim1[2], int aiDim2[2])
- zError_t FeatSim_Process (void ∗phFeatureSimilarityHandle, float ∗∗ppfFeature-Matrix1, float ∗∗ppfFeatureMatrix2)
- zError_t FeatSim_GetResult (void ∗phFeatureSimilarityHandle, FeatureSimilarity-Result_t ∗pstResult)

7.4.1 Detailed Description

C-wrapper for the CFeatureSimilarity class. :

Definition in file FeatureSimilarity_C.h.

7.4.2 Typedef Documentation

7.4.2.1 typedef struct FeatureSimilarityResult_t_tag FeatureSimilarityResult_t

7.4.3 Function Documentation

7.4.3.1 zError_t FeatSim_CreateInstance (void ∗∗ *pphFeatureSimilarityHandle*)

Creates a new instance of the segmentation lib

Parameters

pphFeature-Similarity-Handle	: handle to the new instance

Returns

int : 0 if no error

7.4.3.2 zError_t FeatSim_DestroyInstance (void ∗∗ *pphFeatureSimilarityHandle*)

destroys a previously created instance of the segmentation lib

Parameters

pphFeature-Similarity-Handle	: handle to the instance

Returns

int : 0 if no error

7.4.3.3 const char∗ FeatSim_GetBuildDateString ( )

returns a string containing the build date of this library

Returns

char∗ : string

7.4.3.4 zError_t FeatSim_GetResult (void ∗ *phFeatureSimilarityHandle*, FeatureSimilarityResult_t ∗ *pstResult*)

processes the audio data

Parameters

phFeature-Similarity-Handle	: handle to the instance
∗*pstResult*	: structure with the results (to be written)

Returns

int : 0 if no error

7.4.3.5 const int FeatSim_GetVersion (const Version_t *eVersionIdx*)

returns a string containing the version number of this library

Returns

char∗ : string

7.4.3.6 zError_t FeatSim_Initialize (void ∗ *phFeatureSimilarityHandle*, float ∗ *pfMeans*, int *aiDim1[2]*, int *aiDim2[2]*)

initializes an instance of the segmentation lib

Parameters

phFeature-Similarity-Handle	: handle to the instance
pfMeans	: vector containing the means of each individual feature (from the training set, length is aiDim1[0])
aiDim1	: dimensions of the first matrix, rows:features, columns: observations
aiDim2	: dimensions of the first matrix, rows:features, columns: observations (note that aiDim1[0] == aiDim2[0] and aiDim1[1] > aiDim2[1])

Returns

int : 0 if no error

7.4.3.7 zError_t FeatSim_Process (void ∗ *phFeatureSimilarityHandle*, float ∗∗ *ppfFeatureMatrix1*, float ∗∗ *ppfFeatureMatrix2*)

processes the audio data

Parameters

phFeature-Similarity-Handle	: handle to the instance
∗*ppfFeature-Matrix1*	: big feature matrix [features x observations]
ppfFeature-Matrix2	: small feature matrix [features x observations]

Returns

int : 0 if no error

7.5 FileList.cpp File Reference

implementation of the CFileList class.

`#include "Training_C.h"` `#include "Training_If.h"` Include dependency graph for FileList.cpp:

Defines

- #define kDefaultMaxNumOfEntries 32

default length of list at init

7.5.1 Detailed Description

implementation of the CFileList class. :

Definition in file FileList.cpp.

7.5.2 Define Documentation

7.5.2.1 #define kDefaultMaxNumOfEntries 32

default length of list at init

Definition at line 61 of file FileList.cpp.

Referenced by CFileList::CFileList().

7.6 Globals.h File Reference

some global constants/types.

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

Defines

- #define kMaxNumOfClasses 10

Typedefs

- typedef enum Version_t_tag Version_t
- typedef enum Classifier_t_tag Classifier_t

  indices of available classifiers
- typedef enum TrainingResults_t_tag TrainingResults_t

  indices of training results
- typedef enum zError_t_tag zError_t

  definition of error types

Enumerations

- enum Version_t_tag { kMajor, kMinor, kPatch, kRevision, kNumVersionInts }
- enum Classifier_t_tag { kClassifierLDA, kClassifierQDA, kNumOfClassifiers }

  indices of available classifiers
- enum TrainingResults_t_tag { kTrainResultMeans, kTrainResultScale, kTrain-ResultDet, kNumOfTrainResults }

indices of training results

- enum zError_t_tag { kMlNoError, kMlNothingToDo, kMlNotInitialized, kMl-MatrixDimensionMismatch, kMlInvalidPointer, kMlInvalidArgument, kMlInvalid-TrainingData, kMlInsufficientTrainingData, kMlInvalidSampleRate, kMlInvalid-NumOfChannels, kMlFileOpenFailed, kMlMemAllocFailed, kMlInternalError }

  definition of error types

7.6.1 Detailed Description

some global constants/types. :

Definition in file Globals.h.

7.6.2 Define Documentation

7.6.2.1 #define kMaxNumOfClasses 10

Definition at line 51 of file Globals.h.

7.6.3 Typedef Documentation

7.6.3.1 typedef enum Classifier_t_tag Classifier_t

indices of available classifiers

7.6.3.2 typedef enum TrainingResults_t_tag TrainingResults_t

indices of training results

7.6.3.3 typedef enum Version_t_tag Version_t

7.6.3.4 typedef enum zError_t_tag zError_t

definition of error types

7.6.4 Enumeration Type Documentation

7.6.4.1 enum Classifier_t_tag

indices of available classifiers

Enumerator:

kClassifierLDA	linear discriminant analysis (default)
kClassifierQDA	quadratic discriminant analysis
kNumOfClassifiers	

Definition at line 68 of file Globals.h.

```
    {
        kClassifierLDA,
        kClassifierQDA,

        kNumOfClassifiers
    } Classifier_t;
```

7.6.4.2 enum TrainingResults_t_tag

indices of training results

Enumerator:

kTrainResultMeans index of the means matrix

kTrainResultScale index of the scale matrix

kTrainResultDet index of log determinant vector for each covariance matrix (only for kClassifierQDA)

kNumOfTrainResults

Definition at line 80 of file Globals.h.

```
    {
        kTrainResultMeans,
        kTrainResultScale,
        kTrainResultDet,

        kNumOfTrainResults
    } TrainingResults_t;
```

7.6.4.3 enum Version_t_tag

Enumerator:

kMajor

kMinor

kPatch

kRevision

kNumVersionInts

Definition at line 53 of file Globals.h.

```
    {
        kMajor,
        kMinor,
        kPatch,
        kRevision,

        kNumVersionInts
    } Version_t;
```

7.6.4.4 enum zError_t_tag

definition of error types

Enumerator:

kMlNoError no error occurred

kMlNothingToDo either the parameter has already been set before, or the function call at this time was unnecessary

kMlNotInitialized the instance has not been initialized properly

kMlMatrixDimensionMismatch the matrix does not have the expected dimensions

kMlInvalidPointer one pointer in the function parameters points to an invalid address

kMlInvalidArgument one function parameter has a value that is not allowed (out of range)

kMlInvalidTrainingData something's wrong with the training data

kMlInsufficientTrainingData not enough training data

kMlInvalidSampleRate the sample rate does not meet the internal requirements

kMlInvalidNumOfChannels the number of audio channels does not meet the internal requirements

kMlFileOpenFailed a file could not be opened

kMlMemAllocFailed internal memory allocation failed

kMlInternalError a different error - please contact support

Definition at line 93 of file Globals.h.

```
{
    kMlNoError,

    kMlNothingToDo,
    kMlNotInitialized,

    kMlMatrixDimensionMismatch,

    kMlInvalidPointer,
    kMlInvalidArgument,

    kMlInvalidTrainingData,
    kMlInsufficientTrainingData,

    kMlInvalidSampleRate,
    kMlInvalidNumOfChannels,

    kMlFileOpenFailed,

    kMlMemAllocFailed,

    kMlInternalError

} zError_t;
```

7.7 HelperFunctions.cpp File Reference

helper functions

`#include <memory>` `#include <string>` `#include <fstream>` `#include <iostream>` `#include <iomanip>` `#include "Globals.h"` `#include "Training_C.h"` `#include "HelperFunctions.h"` Include dependency graph for HelperFunctions.cpp:

Functions

- zError_t hlpReallocMatrix (float ∗∗&ppfMatrix, int ∗piNewDimensions, int ∗piOldDimensions)
- zError_t hlpFreeMatrix (float ∗∗&ppfMatrix, int ∗piDimensions)
- zError_t hlpCopyMatrix (float ∗∗ppfDest, float ∗∗ppfSrc, int ∗piDestDimensions, int ∗piSrcDimensions, int ∗piDestStartIndices)
- int hlpGetNumRows (std::ifstream &FFile)
- int hlpGetNumCols (std::ifstream &FFile)
- zError_t hlpLoadMatrixFromFile (float ∗∗ppfMatrix, std::ifstream &FFile, int iNumRows, int iNumCols)

7.7.1 Detailed Description

helper functions :

Definition in file HelperFunctions.cpp.

7.7.2 Function Documentation

7.7.2.1 zError_t hlpCopyMatrix (float ∗∗ *ppfDest,* float ∗∗ *ppfSrc,* int ∗ *piDestDimensions,* int ∗ *piSrcDimensions,* int ∗ *piDestStartIndices*)

Definition at line 144 of file HelperFunctions.cpp.

References kMlMatrixDimensionMismatch, and kMlNoError.

```
{
    int i,
        iStart   = piDestStartIndices[0];

    if (piDestDimensions[0] - piDestStartIndices[0] != piSrcDimensions[0])
        return kMlMatrixDimensionMismatch;
    if (piDestDimensions[1] - piDestStartIndices[1] != piSrcDimensions[1])
        return kMlMatrixDimensionMismatch;

    for (i = iStart; i < iStart + piSrcDimensions[0]; i++)
        memcpy (&ppfDest[iStart][piDestStartIndices[1]], &ppfSrc[i - iStart][0]
      , (piSrcDimensions[1]) * sizeof(float));

    return kMlNoError;
}
```

7.7.2.2 zError_t hlpFreeMatrix (float $**$& *ppfMatrix,* int $*$ *piDimensions*)

Definition at line 128 of file HelperFunctions.cpp.

References kMlInvalidPointer, and kMlNoError.

Referenced by CTraining_If::∼CTraining_If().

```
{
    if (!ppfMatrix)
        return kMlInvalidPointer;

    // free columns
    for (int i = 0; i < piDimensions[0]; i++)
        free (ppfMatrix[i]);

    // free rows
    free (ppfMatrix);
    ppfMatrix   = 0;

    return kMlNoError;
}
```

7.7.2.3 int hlpGetNumCols (std::ifstream & *FFile*)

Definition at line 177 of file HelperFunctions.cpp.

Referenced by main().

```
{
    int iNumOfRows      = 0,
        iNumOfCols      = 0;
    float fDummy;

    // check num of input file elements
    while (!FFile.eof ())
    {
        FFile.ignore (INT_MAX, '\n');
        iNumOfRows++;
    }
    iNumOfRows--;
    FFile.clear ();
    FFile.seekg(0, std::ios::beg);

    while (!FFile.eof ())
    {
        FFile >> fDummy;
        iNumOfCols++;
    }
    iNumOfCols--;
    FFile.clear ();
    FFile.seekg(0, std::ios::beg);

    return iNumOfCols/iNumOfRows;
}
```

7.7.2.4 int hlpGetNumRows (std::ifstream & *FFile*)

Definition at line 160 of file HelperFunctions.cpp.

Referenced by main().

```
{
    int iNumOfRows      = 0;

    // check num of input file elements
    while (!FFile.eof ())
    {
        FFile.ignore (INT_MAX, '\n');
        iNumOfRows++;
    }
    iNumOfRows--;
    FFile.clear ();
    FFile.seekg(0, std::ios::beg);

    return iNumOfRows;
}
```

7.7.2.5 zError_t hlpLoadMatrixFromFile (float ** *ppfMatrix*, std::ifstream & *FFile*, int *iNumRows*, int *iNumCols*)

Definition at line 206 of file HelperFunctions.cpp.

References kMlNoError.

Referenced by main().

```
{
    int i,j;

    // read file
    for (i = 0; i < iNumRows; i++)
    {
        for (j = 0; j < iNumCols; j++)
            FFile >> ppfMatrix[i][j];
    }
    FFile.clear ();
    FFile.seekg(0, std::ios::beg);

    return kMlNoError;
}
```

7.7.2.6 zError_t hlpReallocMatrix (float **& *ppfMatrix*, int * *piNewDimensions*, int * *piOldDimensions*)

Definition at line 63 of file HelperFunctions.cpp.

References kMlMemAllocFailed, and kMlNoError.

Referenced by CTraining_If::CalculateFeaturesAndAppendData4Training().

```
{
    int     i;
    void    *pTmp;

    // we do not need to reallocate if the previous matrix size is equal or
       greater
    if (piNewDimensions[0] <= piOldDimensions[0] && piNewDimensions[1] <=
      piOldDimensions[1])
        return kMlNoError;

    // new alloc of whole matrix
    if (!ppfMatrix)
    {
        // alloc rows
        ppfMatrix = (float**)malloc (sizeof(float*)*piNewDimensions[0]);
        if (!ppfMatrix)
            return kMlMemAllocFailed;

        // alloc columns
        for (i = 0; i < piNewDimensions[0]; i++)
        {
            ppfMatrix[i]    = (float*)malloc (sizeof(float)*piNewDimensions[1])
      ;
            if (!ppfMatrix[i])
                return kMlMemAllocFailed;
        }
    }
    else
    {
        // alloc more rows
        if (piNewDimensions[0] > piOldDimensions[0])
        {
            pTmp                    = realloc(ppfMatrix, sizeof(float*)*
      piNewDimensions[0]);
            if (!pTmp)
                return kMlMemAllocFailed;
            ppfMatrix   = (float**)pTmp;
        }

        // alloc more columns
        if (piNewDimensions[1] > piOldDimensions[1])
        {
            for (i = 0; i < piOldDimensions[0]; i++)
            {
                pTmp                        = realloc (ppfMatrix[i], sizeof(
      float)*piNewDimensions[1]);
                if (!pTmp)
                    return kMlMemAllocFailed;
                ppfMatrix[i]    = (float*)pTmp;
            }
        }
        piNewDimensions[1] = (piNewDimensions[1] > piOldDimensions[1])?
      piNewDimensions[1] : piOldDimensions[1];

        // alloc columns for new rows
        for (i = piOldDimensions[0]; i < piNewDimensions[0]; i++)
        {
            pTmp                        = malloc (sizeof(float)*piNewDimensions
      [1]);
            if (!pTmp)
                return kMlMemAllocFailed;
            ppfMatrix[i]    = (float*)pTmp;
        }
    }
    // update dimensions
    piOldDimensions[0] = (piNewDimensions[0] > piOldDimensions[0])?
      piNewDimensions[0] : piOldDimensions[0];
    piOldDimensions[1] = (piNewDimensions[1] > piOldDimensions[1])?
      piNewDimensions[1] : piOldDimensions[1];
    return kMlNoError;
}
```

7.8 Segmentation.h File Reference

interface of the CSegmentation class.

`#include "Segmentation_C.h"` `#include "zplVecLib.h"` `#include <string>` Include dependency graph for Segmentation.h:

Data Structures

- class CSegmentation

Defines

- #define PREFILTER
- #define RHYTHM
- #define kNumMelCoeffs (4)
- #define kNumNotches (3)
- #define kNumHPFilters (1)
- #define kNumPreFilters (kNumHPFilters+kNumNotches)
- #define kCompleteNumOfFeatures (kNumFeatures $*$ kNumSubFeatures)

7.8.1 Detailed Description

interface of the CSegmentation class. :

Definition in file Segmentation.h.

7.8.2 Define Documentation

7.8.2.1 #define kCompleteNumOfFeatures (kNumFeatures $*$ kNumSubFeatures)

Definition at line 292 of file Segmentation.h.

7.8.2.2 #define kNumHPFilters (1)

Definition at line 193 of file Segmentation.h.

7.8.2.3 #define kNumMelCoeffs (4)

Definition at line 191 of file Segmentation.h.

7.8.2.4 #define kNumNotches (3)

Definition at line 192 of file Segmentation.h.

7.8.2.5 #define kNumPreFilters (kNumHPFilters+kNumNotches)

Definition at line 194 of file Segmentation.h.

7.8.2.6 #define PREFILTER

Definition at line 152 of file Segmentation.h.

7.8.2.7 #define RHYTHM

Definition at line 154 of file Segmentation.h.

7.9 Segmentation_C.h File Reference

`#include "Globals.h"` Include dependency graph for Segmentation_C.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct SegmentationResult_t_tag

Typedefs

- typedef enum SegmentationClasses_t_tag SegmentationClasses_t
- typedef struct 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)
- const int Segment_GetVersion (const Version_t eVersionIdx)
- const char ∗ Segment_GetBuildDateString ()
- int Segment_Initialize (void ∗phSegmentationHandle, int iOverallInputFileLength-InFrames)
- int Segment_PreProcess (void ∗phSegmentationHandle, float ∗pfInputBuffer-Interleaved, 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.-95F, float fMinimumEnergy=0.1F, float ∗∗ppfRawResult=0)
- int Segment_GetSizeOfResult (void ∗phSegmentationHandle)
- int Segment_GetResult (void ∗phSegmentationHandle, SegmentationResult_t ∗pst-Result)
- int Segment_GetSizeOfIntermediateResult (void ∗phSegmentationHandle, int ∗pi-Rows, int ∗piColumns)
- int Segment_GetIntermediateResult (void ∗phSegmentationHandle, float ∗∗ppf-IntermediateResult)
- int Segment_SetIntermediateResult (void ∗phSegmentationHandle, float ∗∗ppf-IntermediateResult, int iRows, int iColumns)

7.9.1 Typedef Documentation

7.9.1.1 typedef enum SegmentationClasses_t_tag SegmentationClasses_t

list of the valid classes

7.9.1.2 typedef struct SegmentationResult_t_tag SegmentationResult_t

declaration of result structure

7.9.2 Enumeration Type Documentation

7.9.2.1 enum SegmentationClasses_t_tag

list of the valid classes

Enumerator:

kDoesNotContainMusic	there is no music
kContainsMusic	there was music detected
kNumOfSegmentationClasses	

Definition at line 14 of file Segmentation_C.h.

```
{
    kDoesNotContainMusic    = 0,
    kContainsMusic          = 1,

    kNumOfSegmentationClasses
} SegmentationClasses_t;
```

7.9.3 Function Documentation

7.9.3.1 int Segment_CreateInstance (void ∗∗ *pphSegmentationHandle,* int *iSampleRate,* int *iNumberOfChannels*)

Creates a new instance of the segmentation lib

Parameters

pph-Segmentation-Handle	: handle to the new instance
iSampleRate	: sample rate of audio file
iNumberOf-Channels	: number of audio channels

Returns

int : 0 if no error

7.9.3.2 int Segment_DestroyInstance (void ∗∗ *pphSegmentationHandle*)

destroys a previously created instance of the segmentation lib

Parameters

pph-Segmentation-Handle	: handle to the instance

Returns

int : 0 if no error

7.9.3.3 int Segment_FinishProcess (void ∗ *phSegmentationHandle*)

called to signal there is no more audio data available

Parameters

ph-Segmentation-Handle	: handle to the instance

Returns

int : 0 if no error

7.9.3.4 const char∗ Segment_GetBuildDateString ( )

returns a string containing the build date of this library

Returns

char∗ : string

7.9.3.5 int Segment_GetIntermediateResult (void $*$ *phSegmentationHandle,* float $**$ *ppfIntermediateResult*)

returns the internal feature data

Parameters

ph-Segmentation-Handle	: handle to the instance
$**$*ppf-Intermediate-Result*	: two-dimensional matrix with the size from GetSizeOfIntermediate-Result

Returns

int : 0 if no error

7.9.3.6 int Segment_GetResult (void $*$ *phSegmentationHandle,* SegmentationResult_t $*$ *pstResult*)

copies the result to pstResult

Parameters

ph-Segmentation-Handle	: handle to the instance
$*$*pstResult*	: pointer to allocated memory where the result can be copied into

Returns

int : 0 if no error

7.9.3.7 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

ph-Segmentation-Handle	: 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

7.9.3.8 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

7.9.3.9 const int Segment_GetVersion (const Version_t *eVersionIdx*)

returns a string containing the version number of this library

Returns

char∗ : string

7.9.3.10 int Segment_Initialize (void ∗ *phSegmentationHandle*, int *iOverallInputFileLengthInFrames*)

initializes an instance of the segmentation lib

Parameters

phSegmentationHandle	: handle to the instance
iOverallInputFileLengthInFrames	: number of audio frames

Returns

int : 0 if no error

7.9.3.11 int Segment_PostProcess (void ∗ *phSegmentationHandle,* float *fTransitionWeight =* `0.6F`*,* float *fMinimumClassTimeInS =* `1.0F`*,* float *fAPrioriProbabilityOfMusic =* `0.95F`*,* float *fMinimumEnergy =* `0.1F`*,* float ∗∗ *ppfRawResult =* `0`)

postprocesses the extracted 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 postive
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)
ppfRawResult	: get raw classification result for each feature vector (memory has to be allocated, dimensions [kNumOfSegmentationClasses][piColumns (from Segment_GetSizeOfIntermediateResult)]

Returns

int : 0 if no error

7.9.3.12 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

7.9.3.13 int Segment_Process (void ∗ *phSegmentationHandle,* float ∗ *pfInputBufferInterleaved,* int *iNumberOfFrames*)

processes the audio data

Parameters

ph-Segmentation-Handle	: handle to the instance
∗*pfInput-Buffer-Interleaved*	: audio input data in pcm interleaved format
iNumberOf-Frames	: number of audio frames in buffer

Returns

int : 0 if no error

7.9.3.14 int Segment_SetIntermediateResult (void ∗ *phSegmentationHandle,* float ∗∗ *ppfIntermediateResult,* int *iRows,* int *iColumns*)

sets the internal feature data (no Segment_Process is required then)

Parameters

ph-Segmentation-Handle	: handle to the instance
∗∗*ppf-Intermediate-Result*	: two dimensional matrix with intermediate results
iRows	: number of rows of matrix
iColumns	: number of columns of matrix

Returns

int : 0 if no error

7.9.3.15 int Segment_SetTxtFilePath (void ∗ *phSegmentationHandle,* char ∗ *pcTxtFilePath*)

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

Parameters

ph-Segmentation-Handle	: handle to the instance
pcTxtFile-Path	: path

Returns

int : 0 if no error

7.10 Training.h File Reference

interface of the CTraining class.

`#include "zplVecLib.h"` Include dependency graph for Training.h:

Data Structures

- class CTraining

7.10.1 Detailed Description

interface of the CTraining class. :

Definition in file Training.h.

7.11 training.txt File Reference

7.12 Training_C.h File Reference

C-interface wrapper for the Training.

`#include "Globals.h"` Include dependency graph for Training_C.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct TrainingSetInfo_t_tag

  info structure on the training data

Typedefs

- typedef struct TrainingSetInfo_t_tag TrainingSetInfo_t

  info structure on the training data

Functions

- zError_t Train_CreateInstance (void ∗∗pphTrainingHandle, int iNumberOfClasses, Classifier_t eClassifier)

  creates a new instance of training
- zError_t Train_DestroyInstance (void ∗∗pphTrainingHandle)

  destroys an instance of training
- const int Train_GetVersion (const Version_t eVersionIdx)
- const char ∗ Train_GetBuildDateString ()
- zError_t Train_AppendFeatureData (void ∗phTrainingHandle, float ∗∗ppfFeature-Matrix, int ∗piMatrixDimensions, int iClassIdx)

  append new feature data to the training set
- zError_t Train_GetTrainingSetInfo (void ∗phTrainingHandle, TrainingSetInfo_t ∗pInfo)

  returns some information about the training data
- Classifier_t Train_GetClassifierType (void ∗phTrainingHandle)

  returns the selected classifier of this instance
- zError_t Train_Process (void ∗phTrainingHandle)

  do the training, the internal data is deleted after process
- zError_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
- zError_t Train_GetResult (void ∗phTrainingHandle, TrainingResults_t eResult-Idx, float ∗∗ppfResult, const int ∗piResultDimensions)

  returns the result with the specific index

7.12.1 Detailed Description

C-interface wrapper for the Training. :

Definition in file Training_C.h.

7.12.2 Typedef Documentation

7.12.2.1 typedef struct TrainingSetInfo_t_tag TrainingSetInfo_t

info structure on the training data

7.12.3 Function Documentation

7.12.3.1 zError_t Train_AppendFeatureData (void ∗ *phTrainingHandle,* float ∗∗ *ppfFeatureMatrix,* int ∗ *piMatrixDimensions,* int *iClassIdx*)

append new feature data to the training set

Parameters

phTraining-Handle	handle to training instance
ppfFeature-Matrix	new data (dimensions ([iNumOfFeatures]x[iNumOfObservations])
piMatrix-Dimensions	dimensions of ppfFeatureMatrix [iRows][iColumns]
iClassIdx	class index (e.g. speech = 0, music = 1)

Returns

0 if no error

Referenced by CTraining_If::CalculateFeaturesAndAppendData4Training(), and CTraining-_If::SetTrainingData().

7.12.3.2 zError_t Train_CreateInstance (void ** *pphTrainingHandle*, int *iNumberOfClasses*, Classifier_t *eClassifier*)

creates a new instance of training

Parameters

pph-Training-Handle	handle to new instance
iNumberOf-Classes	number of classes to be trained for separation (2...10)
eClassifier	classifier to be trained (see Classifier_t)

Returns

0 if no error

Referenced by CTraining_If::CTraining_If().

7.12.3.3 zError_t Train_DestroyInstance (void ** *pphTrainingHandle*)

destroys an instance of training

Parameters

pph-Training-Handle	handle to instance to be destroyed

Returns

0 if no error

Referenced by CTraining_If::∼CTraining_If().

7.12.3.4 const char∗ Train_GetBuildDateString ( )

returns a string containing the build date of this library

Returns

char∗ : string

Referenced by CLShowProgInfo().

7.12.3.5 Classifier_t Train_GetClassifierType (void ∗ *phTrainingHandle*)

returns the selected classifier of this instance

Parameters

phTraining-Handle	handle to training instance

Returns

see Classifier_t

Referenced by CTraining_If::SetParamOutFilePath().

7.12.3.6 zError_t Train_GetResult (void ∗ *phTrainingHandle*, TrainingResults_t *eResultIdx*, float ∗∗ *ppfResult*, const int ∗ *piResultDimensions*)

returns the result with the specific index

Parameters

phTraining-Handle	handle to training instance
eResultIdx	index of result we are interested in
ppfResult	matrix where the result values are copied to
piResult-Dimensions	dimensions of ppfResult, has to equal the parameter from Train_Get-ResultDimension

Returns

0 if no error

See also

Train_GetResultDimension

Referenced by CTraining_If::WriteTrainingResults().

7.12.3.7 zError_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

phTraining-Handle	handle to training instance
eResultIdx	index of result we are interested in
piResult-Dimensions	the result dimensions are written here ([iNumOfRows]x[iNumOf-Columns])

Returns

0 if no error

Referenced by CTraining_If::WriteTrainingResults().

7.12.3.8 zError_t Train_GetTrainingSetInfo (void ∗ *phTrainingHandle*, TrainingSetInfo_t ∗ *pInfo*)

returns some information about the training data

Parameters

phTraining-Handle	handle to training instance
pInfo	pointer to structure where the information is copied into

Returns

0 if no error

7.12.3.9 const int Train_GetVersion (const Version_t *eVersionIdx*)

returns a string containing the version number of this library

Returns

char∗ : string

Referenced by CLShowProgInfo().

7.12.3.10 zError_t Train_Process (void ∗ *phTrainingHandle*)

do the training, the internal data is deleted after process

Parameters

phTraining-Handle	handle to training instance

Returns

0 if no error

Referenced by CTraining_If::Train().

7.13 Training_If.cpp File Reference

implementation of the CTraining_If class.

`#include "zplAudioFile.h"` `#include "FeatureExtraction_C.h"` × `#include "Training_C.h"` `#include "Training_If.h"` `#include "-HelperFunctions.h"` `#include "musegConfig.h"` `#include <fstream>` × `#include <iostream>` `#include <iomanip>` 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 kNumOfAudioFrames2Read 4096

  for audio file parsing, this defines the input block size

Variables

- static const char ∗ pkTrainClassPrefixes [kNumOfClassifiers]

  these are parts of the names of the output files
- static const char ∗ pkTrainResultsFileNames [kNumOfTrainResults]

7.13.1 Detailed Description

implementation of the CTraining_If class. :

Definition in file Training_If.cpp.

7.13.2 Define Documentation

7.13.2.1 #define kDefaultAudioFileExtension ".wav"

initialization value of the audio file extension to parse for

Definition at line 59 of file Training_If.cpp.

Referenced by CTraining_If::CTraining_If().

7.13.2.2 #define kNumOfAudioFrames2Read 4096

for audio file parsing, this defines the input block size

Definition at line 81 of file Training_If.cpp.

Referenced by CTraining_If::CalculateFeaturesAndAppendData4Training().

7.13.2.3 #define kSlash "\\"

just a slash...

Definition at line 65 of file Training_If.cpp.

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

7.13.3 Variable Documentation

7.13.3.1 const char∗ pkTrainClassPrefixes[kNumOfClassifiers] `[static]`

Initial value:

```
  {{"lda_"},
{"qda_"}}
```

these are parts of the names of the output files

Definition at line 71 of file Training_If.cpp.

Referenced by CTraining_If::SetParamOutFilePath().

7.13.3.2 const char∗ pkTrainResultsFileNames[kNumOfTrainResults] `[static]`

Initial value:

```
{{"means.txt"},
{"scale.txt"},
{"det.txt"}}
```

Definition at line 73 of file Training_If.cpp.

Referenced by CTraining_If::SetParamOutFilePath().

7.14 Training_If.h File Reference

interface of the CTraining_If class.

`#include <string>` Include dependency graph for Training_If.h: This graph shows which files directly or indirectly include this file:

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

7.14.1 Detailed Description

interface of the CTraining_If class. :

Definition in file Training_If.h.

7.15 TrainingClMain.cpp File Reference

`#include <time.h>` `#include <string>` `#include <fstream>` `#include <iostream>` `#include <iomanip>` `#include "Training_C.h"` `#include "Training_If.h"` `#include "HelperFunctions.h"` `#include <stdlib.h>` Include dependency graph for TrainingClMain.cpp:

Defines

- #define DBG_READ_FEATURES_AND_CLASSES_FROM_MATLAB 0
- #define kNumMinCLArgs 4

Functions

- static void CLShowProgInfo ()
- static void CLReadArgs (string ∗pstrDirectories, int argc, char ∗argv[ ])
- static void CLShowProcessedTime (clock_t clTime)
- int main (int argc, char ∗argv[ ])

7.15.1 Define Documentation

7.15.1.1 #define DBG_READ_FEATURES_AND_CLASSES_FROM_MATLAB 0

Definition at line 38 of file TrainingClMain.cpp.

7.15.1.2 #define kNumMinCLArgs 4

Definition at line 76 of file TrainingClMain.cpp.

Referenced by main().

7.15.2 Function Documentation

7.15.2.1 static void CLReadArgs (string * *pstrDirectories*, int *argc*, char * *argv[]*) [static]

Definition at line 243 of file TrainingClMain.cpp.

Referenced by main().

```
{
    for (int i = 1; i < argc; i++)
        pstrDirectories[i-1]   = argv[i];

    return;
}
```

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

Definition at line 251 of file TrainingClMain.cpp.

Referenced by main().

```
{
    cout << "\nTime elapsed:\t" << ((float)(clock () - clTime) / CLOCKS_PER_SEC
      ) << " sec" << endl;

    return;
}
```

7.15.2.3 static void CLShowProgInfo ( ) [static]

Definition at line 227 of file TrainingClMain.cpp.

References kMajor, kMinor, kPatch, kRevision, Train_GetBuildDateString(), and -Train_GetVersion().

Referenced by main().

```
{
    cout << "zplane.development  Classification Training Commandline
       Application" << endl;
    cout << "(c) 2002-2016 by zplane" << endl;
    cout << "V"
        << Train_GetVersion (kMajor) << "."
        << Train_GetVersion (kMinor) << "."
        << Train_GetVersion (kPatch) << " build: "
        << Train_GetVersion (kRevision) << ", date: "
        << Train_GetBuildDateString () << endl;
    cout << "Synopsis TrainingTestCl Class1AudioDir Class2AudioDir ...
       OutFileDir" << endl;
    cout << "Press Escape to cancel..." << endl << endl;

    return;
}
```

Here is the call graph for this function:

7.15.2.4 int main (int *argc*, char ∗ *argv[]*)

< local variable for time measurement

< number of classes for classification

< instance handle for training

< array of strings for the directories defined in the command line

Definition at line 85 of file TrainingClMain.cpp.

References CTraining_If::CalculateFeatures(), CLReadArgs(), CLShowProcessedTime(), CLShowProgInfo(), CTraining_If::CreateInstance(), CTraining_If::DestroyInstance(), hlpGetNumCols(), hlpGetNumRows(), hlpLoadMatrixFromFile(), kClassifierQDA, k-NumMinCLArgs, CTraining_If::SetParamDirectoryPaths(), CTraining_If::SetParam-OutFilePath(), CTraining_If::SetTrainingData(), CTraining_If::Train(), and CTraining-_If::WriteTrainingResults().

```
{
    clock_t         clTotalTime;

    int             iNumOfClasses;

    CTraining_If    *pCTrainingInstance = 0;

    string          *pstrDirectories;

#if (!defined(WITHOUT_MEMORY_CHECK) && defined(_DEBUG) && defined (WIN32))
    // set memory checking flags
    int iDbgFlag = _CrtSetDbgFlag(_CRTDBG_REPORT_FLAG);
    iDbgFlag       |= _CRTDBG_CHECK_ALWAYS_DF;
    iDbgFlag       |= _CRTDBG_LEAK_CHECK_DF;
    _CrtSetDbgFlag( iDbgFlag );
#endif
#if (!defined(WITHOUT_EXCEPTIONS) && defined(_DEBUG) && defined (WIN32))
        // enable check for exceptions (don't forget to enable stop in MSVC!)
        _controlfp(~(_EM_INVALID | _EM_ZERODIVIDE | _EM_OVERFLOW |
      _EM_UNDERFLOW | _EM_DENORMAL), _MCW_EM) ;
#endif // #ifndef WITHOUT_EXCEPTIONS

    //check for correct number of command line arguments
    if (argc < kNumMinCLArgs)
    {
        cout << "Wrong number of command line arguments!\n";
        return -1;
    }

    iNumOfClasses   = argc-2;
    pstrDirectories = new string[iNumOfClasses+1];

    // show application info
    CLShowProgInfo ();

    // read command line arguments
    CLReadArgs (pstrDirectories, argc, argv);

    // create class instance
    CTraining_If::CreateInstance (pCTrainingInstance, iNumOfClasses,
      kClassifierQDA);
    cout << "Created Training Instance...\n";
    // set input and output directories
    pCTrainingInstance->SetParamDirectoryPaths (pstrDirectories);
    pCTrainingInstance->SetParamOutFilePath (pstrDirectories[iNumOfClasses]);
    cout << "Parameters set...\n";

    clTotalTime = clock ();

#if DBG_READ_FEATURES_AND_CLASSES_FROM_MATLAB
    {
        float   **ppfFeatures   = 0,
                **ppfObservation= 0,
                *apfClassIdx[2] = {0,0};
        int     i,
                aiMatrixDimensions[2];
        std::ifstream   FFeatureFile,
                        FClassFile;

        // define feature and class file paths
        static const char *pcFeatureFilePath    = "
      H:/temp/soundaware/input/Y.wav.Subfeatures.txt";
        static const char *pcClassFilePath      = "
      H:/temp/soundaware/input/Y.wav.k.txt";

        // open files
        FFeatureFile.open (pcFeatureFilePath, std::ios_base::in);
        FClassFile.open (pcClassFilePath, std::ios_base::in);

        if (!FFeatureFile.is_open () || !FClassFile.is_open ())
        {
            return -1;
        }

        // get matrix dimensions
        aiMatrixDimensions[0]   = hlpGetNumRows (FFeatureFile); // number of
       features
        aiMatrixDimensions[1]   = hlpGetNumCols (FFeatureFile); // number of
       observations

        // alloc memory
        apfClassIdx[0]  = new float [aiMatrixDimensions[1]];
        ppfObservation  = new float* [aiMatrixDimensions[0]];
        ppfFeatures     = new float* [aiMatrixDimensions[0]];
        for (i = 0; i < aiMatrixDimensions[0]; i++)
        {
            ppfFeatures[i]      = new float [aiMatrixDimensions[1]];
            ppfObservation[i]   = new float [1];
        }

        // read in files
        hlpLoadMatrixFromFile (ppfFeatures, FFeatureFile, aiMatrixDimensions[0]
      , aiMatrixDimensions[1]);
        hlpLoadMatrixFromFile (apfClassIdx, FClassFile, 1, aiMatrixDimensions[1
      ]);

        // set training data
        for (i = 0; i < aiMatrixDimensions[1]; i++)
        {
            for (int j = 0; j < aiMatrixDimensions[0]; j++)
                ppfObservation[j][0]    = ppfFeatures[j][i];
            pCTrainingInstance->SetTrainingData (ppfObservation, (int)(
      apfClassIdx[0][i]-.9F), aiMatrixDimensions[0], 1);
```

```
        }

        // free memory
        for (i = 0; i < aiMatrixDimensions[0]; i++)
        {
            delete [] ppfObservation[i];
            delete [] ppfFeatures[i];
        }
        delete [] ppfFeatures;
        delete [] ppfObservation;
        delete [] apfClassIdx[0];

        // close files
        FFeatureFile.close ();
        FClassFile.close ();

        cout << "Trainingset read...\n";
    }
#else
    // do feature calculation
    pCTrainingInstance->CalculateFeatures ();
    cout << "Features calculated...\n";
#endif

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

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

    CLShowProcessedTime (clTotalTime);


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

    delete [] pstrDirectories;

    return 0;

}
```

Here is the call graph for this function: