

Implementation of the digital watermarking method Reference Manual
0.1

Generated by Doxygen 1.3.8

Wed Sep 1 14:09:22 2004

Contents

1	Implementation of the digital watermarking method Hierarchical Index	1
1.1	Implementation of the digital watermarking method Class Hierarchy	1
2	Implementation of the digital watermarking method Class Index	3
2.1	Implementation of the digital watermarking method Class List	3
3	Implementation of the digital watermarking method Class Documentation	5
3.1	DaubechiesD4 Class Reference	5
3.2	DWT Class Reference	7
3.3	DWTDSSSWatermarker Class Reference	13
3.4	FingerprintsMatcher Class Reference	15
3.5	Haar Class Reference	18
3.6	ImageIO Class Reference	20
3.7	Matrix Class Reference	23
3.8	PRNoiseGenerator Class Reference	30
3.9	Watermarker Class Reference	32
3.10	WatermarkMessage Class Reference	33

Chapter 1

Implementation of the digital watermarking method Hierarchical Index

1.1 Implementation of the digital watermarking method Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

DWT	7
DaubechiesD4	5
Haar	18
FingerprintsMatcher	15
ImageIO	20
Matrix	23
PRNoiseGenerator	30
Watermarker	32
DWTDSSSWatermarker	13
WatermarkMessage	33

Chapter 2

Implementation of the digital watermarking method Class Index

2.1 Implementation of the digital watermarking method Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

DaubechiesD4	5
DWT	7
DWTDSSSWatermarker	13
FingerprintsMatcher	15
Haar	18
ImageIO	20
Matrix	23
PRNoiseGenerator	30
Watermarker	32
WatermarkMessage	33

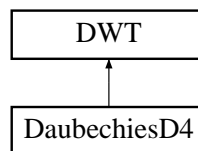
Chapter 3

Implementation of the digital watermarking method Class Documentation

3.1 DaubechiesD4 Class Reference

```
#include <DaubechiesD4.h>
```

Inheritance diagram for DaubechiesD4:



Public Member Functions

- **DaubechiesD4** (**Matrix** *original)
- virtual **~DaubechiesD4** (void)
- void **arrayTransform** (double *signal, const int signalSize)
- void **arrayInvTransform** (double *signal, const int signalSize)

3.1.1 Detailed Description

DaubechiesD4 inherits **DWT**(p. 7) and performs the signal processing relevant to the Daubechies D4 wavelet

3.1.2 Constructor & Destructor Documentation

3.1.2.1 DaubechiesD4::DaubechiesD4 (Matrix * *original*)

Constructor: constructor of the class. The **Matrix**(p.23) passed as parameter is the signal that is to be transformed

Parameters:

original the signal to transform

3.1.2.2 DaubechiesD4::~~DaubechiesD4 (void) [virtual]

destructor

3.1.3 Member Function Documentation

3.1.3.1 void DaubechiesD4::arrayInvTransform (double * *signal*, const int *signalSize*) [virtual]

arrayInvTransform: inverse transforms the given array according to the Daubechies D4 filter.

Parameters:

signal the signal to inverse transform

signalSize the size of the signal

Reimplemented from **DWT** (p.8).

3.1.3.2 void DaubechiesD4::arrayTransform (double * *signal*, const int *signalSize*) [virtual]

arrayTransform: transforms the given array according to the Daubechies D4 filter.

Parameters:

signal the signal to transform

signalSize the size of the signal

Reimplemented from **DWT** (p.8).

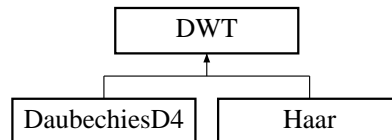
The documentation for this class was generated from the following files:

- DaubechiesD4.h
- DaubechiesD4.cpp

3.2 DWT Class Reference

```
#include <dwt.h>
```

Inheritance diagram for DWT::



Public Member Functions

- **DWT** (**Matrix** *original)
- **~DWT** (void)
- **Matrix** * **getOriginal** (void)
- **Matrix** * **getTransformed** (void)
- **Matrix** * **getInverseTransformed** (void)
- **Matrix** * **getLL** ()
- **Matrix** * **getLH** ()
- **Matrix** * **getHL** ()
- **Matrix** * **getHH** ()
- void **setLL** (**Matrix** *ll)
- void **setLH** (**Matrix** *lh)
- void **setHL** (**Matrix** *hl)
- void **setHH** (**Matrix** *hh)
- void **transform** (void)
- void **invTransform** (void)
- virtual void **arrayTransform** (double *signal, const int signalSize)
- virtual void **arrayInvTransform** (double *signal, const int signalSize)

Protected Member Functions

- void **horizontalTransform** (**Matrix** *source, **Matrix** *destination)
- void **verticalTransform** (**Matrix** *source, **Matrix** *destination)
- void **invHorizontalTransform** (**Matrix** *source, **Matrix** *destination)
- void **invVerticalTransform** (**Matrix** *source, **Matrix** *destination)
- **Matrix** * **getPass** (int vLocator, int hLocator)
- void **setPass** (**Matrix** *band, **Matrix** *destination, int vLocator, int hLocator)

3.2.1 Detailed Description

DWT is a basis class for holding attributes and methods that are common to all 2-D discrete wavelet transform. The code specific to the used filters is to be developed in the subclasses. That code in the subclasses only has to implement the tranform and inverse transformn of 1-D signal

3.2.2 Constructor & Destructor Documentation

3.2.2.1 DWT::DWT (Matrix * *original*)

Constructor: constructor of the basis class

Parameters:

original the matrix we wish to transform

3.2.2.2 DWT::~~DWT (void)

Destructor: destructor of the class instances

3.2.3 Member Function Documentation

3.2.3.1 void DWT::arrayInvTransform (double * *signal*, const int *signalSize*) [virtual]

arrayInvTransform: perform the inverse transform on the given array

Parameters:

signal the array containing the signal to inverse transform

signalSize the size of the array

Reimplemented in **DaubechiesD4** (p. 6), and **Haar** (p. 18).

3.2.3.2 void DWT::arrayTransform (double * *signal*, const int *signalSize*) [virtual]

arrayTransform: perform a first level transform on the given array

Parameters:

signal the array containing the signal to transform

signalSize the size of the array

Reimplemented in **DaubechiesD4** (p. 6), and **Haar** (p. 19).

3.2.3.3 Matrix * DWT::getHH ()

getHH: get the HH band pass of the transformed matrix The original matrix needs to have been transformed by calling the method **transform()**(p. 12) prior to use this method

Returns:

the HH band

3.2.3.4 Matrix * DWT::getHL ()

getHL: get the HL band pass of the transformed matrix The original matrix needs to have been transformed by calling the method **transform()**(p. 12) prior to use this method

Returns:

the HL band

3.2.3.5 Matrix * DWT::getInverseTransformed (void)

getInverseTransformed: get the inverse transformed matrix. The original matrix needs to have been transformed with the **transform()**(p. 12) method prior and the transformed to have been inverse transformed with the **invTransform** prior to call this method

Returns:

the inverse transformed matrix

3.2.3.6 Matrix * DWT::getLH ()

getLH: get the LH band pass of the transformed matrix The original matrix needs to have been transformed by calling the method **transform()**(p. 12) prior to use this method

Returns:

the LH band

3.2.3.7 Matrix * DWT::getLL ()

getLL: get the LL band pass of the transformed matrix The original matrix needs to have been transformed by calling the method **transform()**(p. 12) prior to use this method

Returns:

the LL band

3.2.3.8 Matrix * DWT::getOriginal (void)

getOriginal: get the original matrix, i.e the one that was set to to construct the instance

Returns:

the original matrix

3.2.3.9 Matrix * DWT::getPass (int *vLocator*, int *hLocator*) [protected]

getPass: returns the band pass corresponding to the given horizontal and vertical locators. Meaning of the locators:

(1,1) -> LL ; (1,0) -> LH ; (0,1) -> HL ; (0,0) -> HH

Parameters:

vLocator the vertical locator

hLocator the horizontal locator

3.2.3.10 Matrix * DWT::getTransformed (void)

getTransformed: get the transformed matrix. The original matrix needs to have been transformed with the **transform()**(p. 12) method prior to call this method

Returns:

the transformed matrix

3.2.3.11 void DWT::horizontalTransform (Matrix * *source*, Matrix * *destination*)
[protected]

horizontalTransform: 2-D transform horizontally the data of the given source matrix and store the transformed data in the destination matrix

Parameters:

source the matrix to transform the data of

destination the matrix where to store the transformed data

3.2.3.12 void DWT::invHorizontalTransform (Matrix * *source*, Matrix * *destination*)
[protected]

invHorizontalTranform: 2-D inverse transform horizontally the data of the given source matrix and store the inverse transformed data in the destination matrix

Parameters:

source the matrix to inverse transform the data of

destination the matrix where to store the inverse transformed data

3.2.3.13 void DWT::invTransform (void)

transform: transform the 'original' **Matrix**(p. 23) and set the transformed in the 'transformed' **Matrix**(p. 23)

3.2.3.14 void DWT::invVerticalTransform (Matrix * *source*, Matrix * *destination*)
[protected]

invVerticalTranform: 2-D inverse transform vertical the data of the given source matrix and store the inverse transformed data in the destination matrix

Parameters:

source the matrix to inverse transform the data of

destination the matrix where to store the inverse transformed data

3.2.3.15 void DWT::setHH (Matrix * hh)

setHH: set the given **Matrix**(p. 23) as the HH band pass in the transformed **Matrix**(p. 23) This method is useful in conjunction with **getHH**()(p. 8) when a band pass needs further alteration such as a normalisation

Parameters:

hh the matrix to set as the HH band pass

3.2.3.16 void DWT::setHL (Matrix * hl)

setHL: set the given **Matrix**(p. 23) as the HL band pass in the transformed **Matrix**(p. 23) This method is useful in conjunction with **getHL**()(p. 9) when a band pass needs further alteration such as a normalisation

Parameters:

hl the matrix to set as the HL band pass

3.2.3.17 void DWT::setLH (Matrix * lh)

setLH: set the given **Matrix**(p. 23) as the LH band pass in the transformed **Matrix**(p. 23) This method is useful in conjunction with **getLH**()(p. 9) when a band pass needs further alteration such as a normalisation

Parameters:

lh the matrix to set as the LH band pass

3.2.3.18 void DWT::setLL (Matrix * ll)

setLL: set the given **Matrix**(p. 23) as the LL band pass in the transformed **Matrix**(p. 23) This method is useful in conjunction with **getLL**()(p. 9) when a band pass needs further alteration such as a normalisation

Parameters:

ll the matrix to set as the LL band pass

3.2.3.19 void DWT::setPass (Matrix * band, Matrix * destination, int vLocator, int hLocator) [protected]

setPass: the given band pass at the corresponding location (determined by the given horizontal and vertical locators) into the specified destination **Matrix**(p. 23). Meaning of the locators:

(1,1) -> LL ; (1,0) -> LH ; (0,1) -> HL ; (0,0) -> HH

Parameters:

band the **Matrix**(p. 23) containing the band pass to set

destination the **Matrix**(p. 23) onto which the band pass will be set

vLocator the vertical locator

hLocator the horizontal locator

3.2.3.20 void DWT::transform (void)

transform: transform the 'original' **Matrix**(p.23) and set the transformed in the 'transformed' **Matrix**(p.23)

3.2.3.21 void DWT::verticalTransform (Matrix * *source*, Matrix * *destination*)
[protected]

verticalTranform: 2-D transform vertically the data of the given source matrix and store the transformed data in the destination matrix

Parameters:

source the matrix to transform the data of

destination the matrix where to store the transformed data

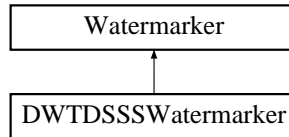
The documentation for this class was generated from the following files:

- dwt.h
- dwt.cpp

3.3 DWTDSSSWatermarker Class Reference

```
#include <DWTDSSSWatermarker.h>
```

Inheritance diagram for DWTDSSSWatermarker::



Public Member Functions

- **DWTDSSSWatermarker** (**ImageIO** *image, int **type**)
- virtual **~DWTDSSSWatermarker** ()
- void **embed** (**WatermarkMessage** *messageToEmbed, int seed, int coefficient)
- bool * **recover** (int binaryMessageSize, int seed)

3.3.1 Detailed Description

DWTDSSSWatermarker is a subclass of **Watermarker**(p.32) that implements a watermarking scheme based on the direct sequence spread spectrum (DSSS) in the discrete wavelet transform (DWT(p.7)) domain

3.3.2 Constructor & Destructor Documentation

3.3.2.1 DWTDSSSWatermarker::DWTDSSSWatermarker (**ImageIO** * *image*, int *type*)

Constructor: construct the watermarker instance for the given image. The specified type determines the **DWT**(p.7) to use. As yet, the **Haar**(p.18) and Daubechies D4 **DWT**(p.7) have been implemented The watermarker can do both embedding an recovering the hidden data

Parameters:

- image* the image to/from which the watermark will be embedded/recovered
- type* the **DWT**(p.7) to be used

3.3.2.2 DWTDSSSWatermarker::~~DWTDSSSWatermarker () [virtual]

Destructor: destruction of the instance

3.3.3 Member Function Documentation

3.3.3.1 void DWTDSSSWatermarker::embed (**WatermarkMessage** * *messageToEmbed*, int *seed*, int *coefficient*)

embed: embeds the given WatermarkerMessage into the current image. The PN generator will be seeded with the given value and noise added to the signal will be scaled by the given factor.

Parameters:

messageToEmbed the watermark message to embed
seed the seed for the PN generator
coefficient the coefficient for scaling the noise

3.3.3.2 **bool * DWTDSSSWatermarker::recover (int *binaryMessageSize*, int *seed*)**

recover: recover the hidden data, given their length and the seed for the PN generator.

Parameters:

binaryMessageSize size of the message in binary
seed the seed for the PN generator

Returns:

an array containing the bits of the recovered message

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

- DWTDSSSWatermarker.h
- DWTDSSSWatermarker.cpp

3.4 FingerprintsMatcher Class Reference

```
#include <FingerprintsMatcher.h>
```

Public Member Functions

- **FingerprintsMatcher** (int SDKCode)
- virtual **~FingerprintsMatcher** ()
- int **getFirstQuality** ()
- int **getSecondQuality** ()
- int **getFirstNbMinutiae** ()
- int **getSecondNbMinutiae** ()
- int **getMatchingScore** ()
- BYTE ** **createBinaryImage** (int width, int height)
- void **setFirstFingerprint** (ImageIO *fpImage)
- void **setSecondFingerprint** (ImageIO *fpImage)
- bool **match** (void)
- void **freeFirstImage** (void)
- void **freeSecondImage** (void)
- void **printOutMatchingInfo** ()

3.4.1 Detailed Description

FingerprintsMatcher allows to match two fingerprints images and get information such as the quality, number of minutiae detected for the two fingerprint images as well as their matching score

3.4.2 Constructor & Destructor Documentation

3.4.2.1 FingerprintsMatcher::FingerprintsMatcher (int *SDKCode*)

Constructor: the constructor of the class. The SDK code passed as argument is meant to determine which SDK the matcher is to make use of. As yet, only the 'BioKeySDK' of Idencom has been used for the different methods. The defined BIOKEY_SDK can be used for the SDK code

Parameters:

SDKCode code of the SDK

3.4.2.2 FingerprintsMatcher::~FingerprintsMatcher () [virtual]

Destructor: destruction of the class instance

3.4.3 Member Function Documentation

3.4.3.1 BYTE ** FingerprintsMatcher::createBinaryImage (int *width*, int *height*)

createBinaryImage: create an empty binary image into which the encoded binary image will be stored. This method is actually the binary image container creator.

Parameters:

width the width of the image

height the height of the image

Returns:

the created binary image

3.4.3.2 void FingerprintsMatcher::freeFirstImage (void)

freeFirstImage: free the memory from the first image. This method is very useful when the FingerprintsMatcher instance has to load and match series of images (for statistical purposes, e.g, FRR/FAR computation).

3.4.3.3 void FingerprintsMatcher::freeSecondImage (void)

freeSecondImage: free the memory from the second image.

3.4.3.4 int FingerprintsMatcher::getFirstNbMinutiae ()

getFirstNbMinutiae: get the number of minutiae detected on the first loaded fingerprint image

Returns:

the number of minutiae

3.4.3.5 int FingerprintsMatcher::getFirstQuality ()

getFirstQuality: get the quality coefficient of the first loaded fingerprint image

Returns:

the coefficient of quality

3.4.3.6 int FingerprintsMatcher::getMatchingScore ()

getMatchingScore: get the computed matching score between the two loaded fingerprint images

Returns:

the matching score

3.4.3.7 int FingerprintsMatcher::getSecondNbMinutiae ()

getSecondNbMinutiae: get the number of minutiae detected on the second loaded fingerprint image

Returns:

the number of minutiae

3.4.3.8 int FingerprintsMatcher::getSecondQuality ()

getSecondQuality: get the quality coefficient of the second loaded fingerprint image

Returns:

the coefficient of quality

3.4.3.9 bool FingerprintsMatcher::match (void)

match: match the loaded fingerprint images. If you call this method prior to load the fingerprint images, it won't crash the application but the matching score will be set to -1 and this function return false

Returns:

whether the matching process could run properly

3.4.3.10 void FingerprintsMatcher::printOutMatchingInfo ()

printOutMatchingInfo: a nice text printout of the information related to the match. It will show the quality, number minutiae found for both matched images as well as their score.

3.4.3.11 void FingerprintsMatcher::setFirstFingerprint (ImageIO * *fpImage*)

setFirstFingerprint: set the first fingerprint by loading its image

Parameters:

fpImage first fingerprint image

3.4.3.12 void FingerprintsMatcher::setSecondFingerprint (ImageIO * *fpImage*)

setSecondFingerprint: set the second fingerprint by loading its image

Parameters:

fpImage second fingerprint image

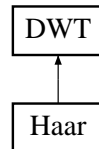
The documentation for this class was generated from the following files:

- FingerprintsMatcher.h
- FingerprintsMatcher.cpp

3.5 Haar Class Reference

```
#include <Haar.h>
```

Inheritance diagram for Haar::



Public Member Functions

- **Haar** (**Matrix** *original)
- void **arrayTransform** (double *signal, const int signalSize)
- void **arrayInvTransform** (double *signal, const int signalSize)
- virtual **~Haar** (void)

3.5.1 Detailed Description

Haar inherits **DWT**(p. 7) and performs the signal processing relevant to the Haar wavelet

3.5.2 Constructor & Destructor Documentation

3.5.2.1 Haar::Haar (**Matrix** * *original*)

Constructor: construct the class with the given **Matrix**(p. 23) as the signal to be processed

Parameters:

original the original signal to process

3.5.2.2 Haar::~~Haar (void) [virtual]

Destructor: destruction of the class instance

3.5.3 Member Function Documentation

3.5.3.1 void Haar::arrayInvTransform (double * *signal*, const int *signalSize*) [virtual]

arrayInvTransform: inverse transform the given array according to the Haar **DWT**(p. 7)

Parameters:

signal to inverse transform

signalSize size of the signal to inverse transform

Reimplemented from **DWT** (p. 8).

3.5.3.2 void Haar::arrayTransform (double * *signal*, const int *signalSize*) [virtual]

arrayTransform: transform the given array according to the Haar DWT(p. 7)

Parameters:

signal to transform

signalSize size of the signal to transform

Reimplemented from DWT (p. 8).

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

- Haar.h
- Haar.cpp

3.6 ImageIO Class Reference

```
#include <ImageIO.h>
```

Public Member Functions

- **ImageIO** (char *filePath)
- **ImageIO** (int width, int height)
- **~ImageIO** (void)
- int **getWidth** (void)
- int **getHeight** (void)
- void **setImageMatrix** (**Matrix** *data)
- **BOOL readFromDisk** (char *filePath)
- **BOOL writeToDisk** (char *filepath)
- **readFromSensor** (void)
- **BYTE ** getByteImageMatrixData** (void)
- **double ** getDoubleImageMatrixData** (void)
- **Matrix * getImageMatrix** (void)
- **double getPSNRWith** (**ImageIO** *otherImage)

3.6.1 Detailed Description

ImageIO is the implementation of an image input/output allowing to read and write images file from and to disk. An extension would be to read image from a sensor.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 ImageIO::ImageIO (char * *filePath*)

Constructor: constructs an image from the file path given as parameter.

Parameters:

filePath the path to file image to construct from

3.6.2.2 ImageIO::ImageIO (int *width*, int *height*)

Constructor: constructs an empty image with the given width and height

Parameters:

width width of the image

height height of the image

3.6.2.3 ImageIO::~ImageIO (void)

Destructor: destructor of the ImageIO's components

3.6.3 Member Function Documentation

3.6.3.1 `BYTE ** ImageIO::getBytesMatrixData (void)`

`getBytesMatrix`: generates a 2-D array of bytes containing the pixels of this matrix Note that values will rounded by casting them to integer

Returns:

the 2-D array of bytes

3.6.3.2 `double ** ImageIO::getDoubleMatrixData (void)`

`getDoubleMatrix`: generates a 2-D array of doubles containing the pixels of this matrix

Returns:

the 2-D array of double

3.6.3.3 `int ImageIO::getHeight (void)`

`getWidth`: get the width of this image

Returns:

the width of this image

3.6.3.4 `Matrix * ImageIO::getImageMatrix (void)`

`getImageMatrix`: get the **Matrix**(p. 23) representation of this image pixels

Returns:

the **Matrix**(p. 23) of pixels

3.6.3.5 `double ImageIO::getPSNRWith (ImageIO * otherImage)`

`getPSNRWith`: calculates the PSNR between this image and the given image

Parameters:

otherImage the image to calculate the PSNR with

Returns:

the PSNR value

3.6.3.6 `int ImageIO::getWidth (void)`

`getWidth`: get the width of this image

Returns:

the width of this image

3.6.3.7 BOOL ImageIO::readFromDisk (char * *filePath*)

readFromDisk: read an image from disk

Parameters:

filePath path to the image file to open

Returns:

wether the image could be read from the file on disk

3.6.3.8 ImageIO::readFromSensor (void)

readFromSensor: read from sensor This method has not yet be implemented but would allow to read an image directly from a sensor

3.6.3.9 void ImageIO::setImageMatrix (Matrix * *data*)

setImageMatrix: set the image pixels to the coefficients in the given **Matrix**(p. 23)

Parameters:

data the **Matrix**(p. 23) containing the pixels to set

3.6.3.10 BOOL ImageIO::writeToDisk (char * *filepath*)

writeToDisk: write the current image to disk

Parameters:

filepath path to the file the image will be written to

Returns:

wether the image could be written to the specified file on disk

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

- ImageIO.h
- ImageIO.cpp

3.7 Matrix Class Reference

```
#include <Matrix.h>
```

Public Member Functions

- **Matrix** (int width, int height)
- **Matrix** (double **data, int width, int height)
- **~Matrix** (void)
- int **getWidth** (void)
- int **getHeight** (void)
- double **getMax** (void)
- double **getMin** (void)
- double ** **getDoubleData** (void)
- BYTE ** **getByteData** (void)
- double **correlationCoeffWith** (**Matrix** *otherMatrix)
- double * **getRow** (int position)
- double * **getColumn** (int position)
- double **getPixelIndex** (int x, int y)
- **Matrix** * **getDistribution** ()
- double **mean** (void)
- double **getSum** ()
- double **getPSNRWith** (**Matrix** *otherMatrix)
- void **normalise** (int treshold)
- void **specialNormalise** (double min, double max)
- bool **isBeyond** (int min, int max)
- void **add** (**Matrix** *matrixData)
- void **subtract** (**Matrix** *matrixData)
- void **setData** (double **data)
- void **setPixelIndex** (int x, int y, double value)
- void **setRow** (double *rowData, int position)
- void **setColumn** (double *columnData, int position)
- void **printToFile** (char *fileName, char separator)
- double **mean** (double **matrixData)
- double **sumVariances** (double **otherMatrixData)
- double **sumDeviations** (double **matrixData)

3.7.1 Detailed Description

Matrix is the implementation of 2-D array. It can be used to store and manipulate data represented in the form a matrix such as the pixels of an image. This class is aimed at making the typical manipulation of the contained data in a fairly easy way by offering high-level methods.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 Matrix::Matrix (int *width*, int *height*)

Constructor: construct a matrix of the given width and height Note that also this constructor will create the data container it won't set any value and you should not call accessors prior to fill the container with relevant methods.

Parameters:

width the width of the matrix

height the height of the matrix

3.7.2.2 Matrix::Matrix (double ** *data*, int *width*, int *height*)

Constructor: construct a matrix of the given width and height and fed by the given 2-D array of doubles.

Parameters:

data a 2-D array of double to create the matrix from

width the width of the matrix

height the height of the matrix

3.7.2.3 Matrix::~~Matrix (void)

Destructor: destructor of the class.

3.7.3 Member Function Documentation

3.7.3.1 void Matrix::add (Matrix * *matrixData*)

add: add the given matrix to this matrix Although the added matrix may have smaller width and height of this matrix, the addition will happen as mask starting at coordinates [0,0] for both matrices

Parameters:

matrixData the matrix to add to this matrix

3.7.3.2 double Matrix::correlationCoeffWith (Matrix * *otherMatrix*)

correlationCoeffWith: returns the correlation coefficient of this matrix with the given matrix

Parameters:

otherMatrix the other Matrix with which the correlation will be calculated

Returns:

the correlation coefficient

3.7.3.3 BYTE ** Matrix::getBytesData (void)

getBytesData: returns the content of Matrix as a 2-D array of bytes.

Returns:

a 2-D array of bytes containing the matrix data

3.7.3.4 double * Matrix::getColumn (int position)

getColumn: abstraction of a matrix column. Returns an array containing the coefficients at the width specified in the parameters

Parameters:

position horizontal position (width) of the row

Returns:

the specified row

3.7.3.5 Matrix * Matrix::getDistribution ()

getDistribution: create a matrix containing the distribution of the contained coefficients. This is quite useful looking at the shape of the distribution. By using the **printToFile()**(p.28) method, that matrix can be printed out to a file a specific format (e.g, CSV) to then be plotted in a proper application

Returns:

the distribution of the matrix coefficients

3.7.3.6 double ** Matrix::getDoubleData (void)

getDoubleData: returns the content of Matrix as a 2-D array of doubles.

Returns:

a 2-D array of doubles containing the matrix data

3.7.3.7 int Matrix::getHeight (void)

getHeight: returns the height of the matrix

Returns:

the height of the matrix

3.7.3.8 double Matrix::getMax (void)

getMax: returns, as a double, the biggest matrix coefficient

Returns:

value of the highest coefficient in the matrix

3.7.3.9 double Matrix::getMin (void)

getMin: returns, as a double, the smallest matrix coefficient

Returns:

value of the smallest coefficient in the matrix

3.7.3.10 double Matrix::getPixelIndex (int *x*, int *y*)

getPixelIndex: returns the value of the pixel at the given coordinate

Parameters:

x horizontal position of the pixel

y vertical position of the pixel

Returns:

the value of the pixel

3.7.3.11 double Matrix::getPSNRWith (Matrix * *otherMatrix*)

getPSNRWith: calculate the peak signal-to-noise ratio (PSNR) of this matrix with the matrix given in parameters.

Parameters:

otherMatrix the matrix to calculate the PSNR with

Returns:

the PSNR of this matrix with the given matrix

3.7.3.12 double * Matrix::getRow (int *position*)

getRow: abstraction of a matrix row. Returns an array containing the coefficients at the height specified in the parameters

Parameters:

position vertical position (height) of the row

Returns:

the specified row

3.7.3.13 double Matrix::getSum ()

getSum: calculates the sum of all the matrix coefficients

Returns:

the sum of all the coefficients

3.7.3.14 int Matrix::getWidth (void)

getWidth: returns the width of the matrix

Returns:

the width of the matrix

3.7.3.15 bool Matrix::isBeyond (int *min*, int *max*)

isBeyond: tell if any coefficients of this matrix are beyond the given minimum and maximum boundaries

Parameters:

max the maximum boundary

min the minimum boundary

Returns:

wether any coefficients are beyond the given boundaries

3.7.3.16 double Matrix::mean (double ** *matrixData*)

mean: calculates the mean of all coefficients in the given 2-D array of double This method can of course be applied to this matrix by passing the content as a 2-D array of double

Parameters:

matrixData the 2-D array to get the mean of

Returns:

the mean of the coefficients in the given 2-D array

3.7.3.17 double Matrix::mean (void)

mean: calculates the mean of hte matrix coefficients

Returns:

the mean of the matrix coefficients

3.7.3.18 void Matrix::normalise (int *threshold*)

normalise: normalisation the matrix. Since the distribution will be {0..1} by default, the treshold parameter is meant to determine by which value the resulting normalisation will be scaled

Parameters:

threshold treshold value for scaling the normalisation

3.7.3.19 void Matrix::printToFile (char * *fileName*, char *separator*)

printToFile: print this matrix in a file which path and separator are given as parameters. The separator could for instance be a ',' for CSV formatting

Parameters:

fileName the path to the file the matrix is to be written to

separator the separator

3.7.3.20 void Matrix::setColumn (double * *columnData*, int *position*)

setColumn: abstraction of a column. Set the given array of double as a column in the matrix at the horizontal position given as parameter

Parameters:

columnData the array of double to set

position the vertical (width) position of the column to set

3.7.3.21 void Matrix::setData (double ** *data*)

setData: set the data of the matrix with the 2-D array of double given as parameter

Parameters:

data the 2-D array of data to add

3.7.3.22 void Matrix::setPixelIndex (int *x*, int *y*, double *value*)

setPixelIndex: set the index at the given coordinates to the given value

Parameters:

x x coordinate of the value in this matrix

y y coordinate of the value in this matrix

value the value to set

3.7.3.23 void Matrix::setRow (double * *rowData*, int *position*)

setRow: abstraction of a row. Set the given array of double as a row in the matrix at the vertical position given as parameter

Parameters:

rowData the array of double to set

position the vertical (height) position of the row to set

3.7.3.24 void Matrix::specialNormalise (double *min*, double *max*)

specialNormalise: normalises the matrix and scale to distribute between the maximum and minimum given as parameters

Parameters:

max the minimum bound for scaling the normalisation

min the minimum bound for scaling the normalisation

3.7.3.25 void Matrix::subtract (Matrix * *matrixData*)

subtract: subtract the given matrix to this matrix The subtraction is done in the same manner as the addition and thus has the same properties

Parameters:

matrixData the matrix to subtract to this matrix

3.7.3.26 double Matrix::sumDeviations (double ** *matrixData*)

sumDeviations: calculates the sum of the deviations of this matrix with the matrix given as parameter

Parameters:

matrixData the matrix to calculate the sum of deviations with

Returns:

the sum of the deviations

3.7.3.27 double Matrix::sumVariances (double ** *otherMatrixData*)

sumVariances: calculates the sum of the variance of this matrix with the matrix given as parameter

Parameters:

otherMatrixData the matrix to calculate the sum of variances with

Returns:

the sum of the variances

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

- Matrix.h
- Matrix.cpp

3.8 PRNoiseGenerator Class Reference

```
#include <PRNoiseGenerator.h>
```

Public Member Functions

- **PRNoiseGenerator** (int *seed*, int *coefficient*, int *width*, int *height*)
- virtual **~PRNoiseGenerator** (void)
- **Matrix * getNoiseMatrix** (void)
- **Matrix * getNoiseMatrix** (int *step*)
- void **setSeed** (int *seed*)
- void **reset** (void)

3.8.1 Detailed Description

PRNoise Generator is the implementation of a pseudorandom number generator. It can be used to generated matrices of noise that can be reproduced if seeded with the same value

3.8.2 Constructor & Destructor Documentation

3.8.2.1 PRNoiseGenerator::PRNoiseGenerator (int *seed*, int *coefficient*, int *width*, int *height*)

Constructor: constructs a pseudorandom noise generator which will be seed with the given value, scale the noise by given coefficient and produce noise matrices of the given width and height

Parameters:

- seed* the seed for the generator
- coefficient* for scaling the noise
- width* width of the generated noise matrices
- height* height of the generated noise matrices

3.8.2.2 PRNoiseGenerator::~~PRNoiseGenerator (void) [virtual]

destructor: destroys the objects of the instance

3.8.3 Member Function Documentation

3.8.3.1 Matrix * PRNoiseGenerator::getNoiseMatrix (int *step*)

getNoiseMatrix(step): get the noise matrix for the specified state By giving the step argument, it is possible at any state of the generator to get the noise matrix relevant to the specified step

Parameters:

- step* the step for the desired state

Returns:

- the noise matrix

3.8.3.2 Matrix * PRNoiseGenerator::getNoiseMatrix (void)

getNoiseMatrix(void)(p. 31): get the current noise matrix

Returns:

the generated noise matrix for the current state

3.8.3.3 void PRNoiseGenerator::reset (void)

reset: reset the pseudorandom generator to the current seed value

3.8.3.4 void PRNoiseGenerator::setSeed (int *seed*)

setSeed: seed the generator with the given value

Parameters:

seed the value to seed the generator with

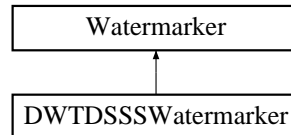
The documentation for this class was generated from the following files:

- PRNoiseGenerator.h
- PRNoiseGenerator.cpp

3.9 Watermarker Class Reference

```
#include <Watermarker.h>
```

Inheritance diagram for Watermarker::



Public Member Functions

- **Watermarker** (**ImageIO** *image)

Protected Attributes

- **int** type

3.9.1 Detailed Description

Watermarker is a basis class that regroups common methods and attributes for different watermarking schemes that will inherit from it.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 Watermarker::Watermarker (**ImageIO** * *image*)

Constructor: constructs the watermarker with the given image to embed/recover message to/from

Parameters:

image the image for embedding recovering

3.9.3 Member Data Documentation

3.9.3.1 **int** Watermarker::type [protected]

DWT(p. 7) type, **Haar**(p. 18): 0; **DaubD4**: 1

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

- Watermarker.h
- Watermarker.cpp

3.10 WatermarkMessage Class Reference

```
#include <WatermarkMessage.h>
```

Public Member Functions

- **WatermarkMessage** (char *message, bool useTimestamp, bool encrypted=false)
- **WatermarkMessage** (bool *binaryMessage, int binaryMessageSize, bool useTimestamp, bool encrypted)
- char * **getStringMessage** (void)
- int **getBinaryMessageSize** ()
- bool * **getBinaryMessage** (void)
- int **getMessageTimestamp** (void)
- bool **isBinaryBitSetAt** (int index)
- bool **isEncrypted** (void)
- void **setEncrypted** (bool encrypted)
- void **setBinaryMessage** (void)
- void **setStringMessage** (void)
- bool **encrypt** ()
- bool **decrypt** ()

3.10.1 Detailed Description

WatermarkMessage is the implementation of message object that is meant to be embedded. It handles the tasks of converting the message to and from binary sequences. It may (optional) add the current timestamp in the message.

3.10.2 Constructor & Destructor Documentation

3.10.2.1 WatermarkMessage::WatermarkMessage (char * message, bool useTimestamp, bool encrypted = false)

Constructor: construction of a watermark message. This constructor is to be used when embedding. It takes as parameter a string to embed, offers to optionally embed the current timestamp and to encrypt the message. The encryption has not yet been implemented

Parameters:

message the string message to embed

useTimestamp whether we want to embed the current timestamp

encrypted whether we want to encrypt the message

3.10.2.2 WatermarkMessage::WatermarkMessage (bool * binaryMessage, int binaryMessageSize, bool useTimestamp, bool encrypted)

Constructor: construction of a watermark message. This constructor is to be used when recovering as it take a binary sequence as parameter for the message. The size of the message needs to be specified (in its binary format). We also specify if a timestamp is to be recovered and if the recovered message will be encrypted.

Parameters:

binaryMessage the recovered message in binary

binaryMessageSize size of the binary message

useTimestamp whether the message contains a timestamp

encrypted whether the message is encrypted

3.10.3 Member Function Documentation

3.10.3.1 `bool WatermarkMessage::decrypt ()`

decrypt: decrypts the current message

This function is not implemented yet

3.10.3.2 `bool WatermarkMessage::encrypt ()`

encrypt: encrypts the current message

This function is not implemented yet

3.10.3.3 `bool* WatermarkMessage::getBinaryMessage (void)`

getBinaryMessage: get the message in its binary format. This is what is going to be embedded or has been recovered and it contains the timestamp (if it has been enabled).

Returns:

the message in its binary format

3.10.3.4 `int WatermarkMessage::getBinaryMessageSize ()`

getBinaryMessageSize: get the size of the current binary message. Note that the binary message contains the timestamp (if enabled)

Returns:

the size of the binary message

3.10.3.5 `WatermarkMessage::getMessageTimestamp (void)`

getMessageTimeStamp: get the message timestamp. The timestamp is an integer of 4Bytes (32 bits) and is in the Unix format, i.e, the number of seconds since the Epoch (1/1/1970)

Returns:

the timestamp in Unix format

3.10.3.6 char * WatermarkMessage::getStringMessage (void)

getStringMessage: get the message in its string representation. Note that even if the timestamp has been set, it is not contained in the string representation of the message but can be called with the `getMessageTimestamp()` (p. 34) method.

Returns:

the string representation of the message

3.10.3.7 bool WatermarkMessage::isBinaryBitSetAt (int *index*)

isBinaryBitSetAt: whether the binary at the given index is set to one

Parameters:

index of the bit in the sequence

Returns:

whether the asked bit is set to 1 (true) or 0 (false)

3.10.3.8 bool WatermarkMessage::isEncrypted (void)

isEncrypted:

Returns:

whether the message is encrypted

3.10.3.9 void WatermarkMessage::setBinaryMessage (void)

setBinaryMessage: set the binary message from the current string message and optionally timestamp by converting them into a binary sequence

3.10.3.10 void WatermarkMessage::setEncrypted (bool *encrypted*)

setEncrypted:

Parameters:

encrypted set whether the message is encrypted

3.10.3.11 void WatermarkMessage::setStringMessage (void)

setStringMessage: set the string message and timestamp from the current binary sequence by doing the relevant format conversion

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

- WatermarkMessage.h
- WatermarkMessage.cpp

Index

- ~DWT
 - DWT, 8
- ~DWTDSSSWatermarker
 - DWTDSSSWatermarker, 13
- ~DaubechiesD4
 - DaubechiesD4, 6
- ~FingerprintsMatcher
 - FingerprintsMatcher, 15
- ~Haar
 - Haar, 18
- ~ImageIO
 - ImageIO, 20
- ~Matrix
 - Matrix, 24
- ~PRNoiseGenerator
 - PRNoiseGenerator, 30
- add
 - Matrix, 24
- arrayInvTransform
 - DaubechiesD4, 6
 - DWT, 8
 - Haar, 18
- arrayTransform
 - DaubechiesD4, 6
 - DWT, 8
 - Haar, 18
- correlationCoeffWith
 - Matrix, 24
- createBinaryImage
 - FingerprintsMatcher, 15
- DaubechiesD4, 5
 - DaubechiesD4, 6
- DaubechiesD4
 - ~DaubechiesD4, 6
 - arrayInvTransform, 6
 - arrayTransform, 6
 - DaubechiesD4, 6
- decrypt
 - WatermarkMessage, 34
- DWT, 7
 - ~DWT, 8
 - arrayInvTransform, 8
 - arrayTransform, 8
 - DWT, 8
 - getHH, 8
 - getHL, 8
 - getInverseTransformed, 9
 - getLH, 9
 - getLL, 9
 - getOriginal, 9
 - getPass, 9
 - getTransformed, 9
 - horizontalTransform, 10
 - invHorizontalTransform, 10
 - invTransform, 10
 - invVerticalTransform, 10
 - setHH, 10
 - setHL, 11
 - setLH, 11
 - setLL, 11
 - setPass, 11
 - transform, 11
 - verticalTransform, 12
- DWTDSSSWatermarker, 13
 - ~DWTDSSSWatermarker, 13
 - DWTDSSSWatermarker, 13
 - embed, 13
 - recover, 14
- embed
 - DWTDSSSWatermarker, 13
- encrypt
 - WatermarkMessage, 34
- FingerprintsMatcher, 15
 - FingerprintsMatcher, 15
- FingerprintsMatcher
 - ~FingerprintsMatcher, 15
 - createBinaryImage, 15
 - FingerprintsMatcher, 15
 - freeFirstImage, 16
 - freeSecondImage, 16
 - getFirstNbMinutiae, 16
 - getFirstQuality, 16
 - getMatchingScore, 16
 - getSecondNbMinutiae, 16
 - getSecondQuality, 16

- match, 17
- printOutMatchingInfo, 17
- setFirstFingerprint, 17
- setSecondFingerprint, 17
- freeFirstImage
 - FingerprintsMatcher, 16
- freeSecondImage
 - FingerprintsMatcher, 16
- getBinaryMessage
 - WatermarkMessage, 34
- getBinaryMessageSize
 - WatermarkMessage, 34
- getByteData
 - Matrix, 24
- getByteImageMatrixData
 - ImageIO, 21
- getColumn
 - Matrix, 25
- getDistribution
 - Matrix, 25
- getDoubleData
 - Matrix, 25
- getDoubleImageMatrixData
 - ImageIO, 21
- getFirstNbMinutiae
 - FingerprintsMatcher, 16
- getFirstQuality
 - FingerprintsMatcher, 16
- getHeight
 - ImageIO, 21
 - Matrix, 25
- getHH
 - DWT, 8
- getHL
 - DWT, 8
- getImageMatrix
 - ImageIO, 21
- getInverseTransformed
 - DWT, 9
- getLH
 - DWT, 9
- getLL
 - DWT, 9
- getMatchingScore
 - FingerprintsMatcher, 16
- getMax
 - Matrix, 25
- getMessageTimestamp
 - WatermarkMessage, 34
- getMin
 - Matrix, 25
- getNoiseMatrix
 - PRNoiseGenerator, 30
- getOriginal
 - DWT, 9
- getPass
 - DWT, 9
- getPixelIndex
 - Matrix, 26
- getPSNRWith
 - ImageIO, 21
 - Matrix, 26
- getRow
 - Matrix, 26
- getSecondNbMinutiae
 - FingerprintsMatcher, 16
- getSecondQuality
 - FingerprintsMatcher, 16
- getStringMessage
 - WatermarkMessage, 34
- getSum
 - Matrix, 26
- getTransformed
 - DWT, 9
- getWidth
 - ImageIO, 21
 - Matrix, 26
- Haar, 18
 - ~Haar, 18
 - arrayInvTransform, 18
 - arrayTransform, 18
 - Haar, 18
- horizontalTransform
 - DWT, 10
- ImageIO, 20
 - ImageIO, 20
- ImageIO
 - ~ImageIO, 20
 - getByteImageMatrixData, 21
 - getDoubleImageMatrixData, 21
 - getHeight, 21
 - getImageMatrix, 21
 - getPSNRWith, 21
 - getWidth, 21
 - ImageIO, 20
 - readFromDisk, 21
 - readFromSensor, 22
 - setImageMatrix, 22
 - writeToDisk, 22
- invHorizontalTransform
 - DWT, 10
- invTransform
 - DWT, 10
- invVerticalTransform
 - DWT, 10

- isBeyond
 - Matrix, 27
- isBinaryBitSetAt
 - WatermarkMessage, 35
- isEncrypted
 - WatermarkMessage, 35
- match
 - FingerprintsMatcher, 17
- Matrix, 23
 - ~Matrix, 24
 - add, 24
 - correlationCoeffWith, 24
 - getByteData, 24
 - getColumn, 25
 - getDistribution, 25
 - getDoubleData, 25
 - getHeight, 25
 - getMax, 25
 - getMin, 25
 - getPixelIndex, 26
 - getPSNRWith, 26
 - getRow, 26
 - getSum, 26
 - getWidth, 26
 - isBeyond, 27
 - Matrix, 24
 - mean, 27
 - normalise, 27
 - printToFile, 27
 - setColumn, 28
 - setData, 28
 - setPixelIndex, 28
 - setRow, 28
 - specialNormalise, 28
 - subtract, 29
 - sumDeviations, 29
 - sumVariances, 29
- mean
 - Matrix, 27
- normalise
 - Matrix, 27
- printOutMatchingInfo
 - FingerprintsMatcher, 17
- printToFile
 - Matrix, 27
- PRNoiseGenerator, 30
 - PRNoiseGenerator, 30
- PRNoiseGenerator
 - ~PRNoiseGenerator, 30
 - getNoiseMatrix, 30
 - PRNoiseGenerator, 30
 - reset, 31
 - setSeed, 31
- readFromDisk
 - ImageIO, 21
- readFromSensor
 - ImageIO, 22
- recover
 - DWTDSSSWatermarker, 14
- reset
 - PRNoiseGenerator, 31
- setBinaryMessage
 - WatermarkMessage, 35
- setColumn
 - Matrix, 28
- setData
 - Matrix, 28
- setEncrypted
 - WatermarkMessage, 35
- setFirstFingerprint
 - FingerprintsMatcher, 17
- setHH
 - DWT, 10
- setHL
 - DWT, 11
- setImageMatrix
 - ImageIO, 22
- setLH
 - DWT, 11
- setLL
 - DWT, 11
- setPass
 - DWT, 11
- setPixelIndex
 - Matrix, 28
- setRow
 - Matrix, 28
- setSecondFingerprint
 - FingerprintsMatcher, 17
- setSeed
 - PRNoiseGenerator, 31
- setStringMessage
 - WatermarkMessage, 35
- specialNormalise
 - Matrix, 28
- subtract
 - Matrix, 29
- sumDeviations
 - Matrix, 29
- sumVariances
 - Matrix, 29
- transform

- DWT, 11
- type
 - Watermarker, 32
- verticalTransform
 - DWT, 12
- Watermarker, 32
 - type, 32
 - Watermarker, 32
- WatermarkMessage, 33
 - WatermarkMessage, 33
- WatermarkMessage
 - decrypt, 34
 - encrypt, 34
 - getBinaryMessage, 34
 - getBinaryMessageSize, 34
 - getMessageTimestamp, 34
 - getStringMessage, 34
 - isBinaryBitSetAt, 35
 - isEncrypted, 35
 - setBinaryMessage, 35
 - setEncrypted, 35
 - setStringMessage, 35
 - WatermarkMessage, 33
- writeToDisk
 - ImageIO, 22