FaceCell 1.1 EDK

FaceCell 1.1 EDK

Published November 5, 2007. Version 1.1.0.0

Table of Contents

Chapter 1. About

1.1. Introduction

FaceCell EDK consists of FccExtractor, FccMatcher main libraries. NCore and NImages are auxiliary libraries that provides infrastructure and functionality for working with images. The samples programs were developed to show how to use FaceCell EDK components.

1.2. Platforms Supported

FaceCell EDK supports platforms based on ARM processor architecture. Libraries for Windows Mobile and ARM Linux operating systems are provided.

1.3. System Requirements

- ARM based 400 MHz processor is recommended for face enrollment in less than two seconds. Supported ARM processor core families are: XScale, StrongArm, ARM11, ARM10, ARM9.
- At least 8 Mb of memory for FaceCell code and data arrays.
- Embedded camera with at least 320 x 240 pixels physical resolution (640 x 480 pixels recommended).
- Linux or MS Windows Mobile 2003 (or later).

Please note, that FaceCell source code EDK could be easily ported to most other platforms and processors.

1.4. Licensing

Customers should sign the FaceCell 1.0 source code EDK Software Licensing Agreement before purchasing FaceCell 1.0 source code EDK.

10,000 FaceCell 1.0 installation licenses are already included with the FaceCell 1.0 source code EDK license. Additional FaceCell 1.0 installation licenses may be purchased anytime.

Chapter 2. What's New

2.1. Version 1.1.0.0

- Improved face and eyes detection precision.
- Improved enrollment from sequence of images (generates more reliable template).
- Added quality measure to facial features extraction. It can be controlled by face quality threshold.
- Added NLRecord library that is required by FccExtractor and FccMatcher libraries.

2.2. Version 1.0.1.0

- Improved eye detection.
- Updated components to newer versions.

2.3. Version 1.0.0.2

- Updated components to newer versions.

2.4. Version 1.0.0.1

- JPEG image format support was added.

2.5. Version 1.0.0.0

- Initial release.

Chapter 3. Overview

3.1. Image Support

Image support in the FaceCell EDK can be divided into the following three parts:

- Image. The base of all image support. Developers should start using this part and take advantage of other parts if it is required.
- Image Format. Declares the supported image formats. Shows how to load and save images in a format-neutral way.
- Low-Level Image Input-Output. Should be used to have more control on how images are loaded and saved in particular format.

3.1.1. Image

Image is a rectangular area of pixels (image elements), defined by width, height and pixel format.

Pixel format describes type of color information contained in the image like monochrome, grayscale, true color or palette-based (indexed) and describes pixels storage in memory (how many bits are required to store one pixel).

Image in the FaceCell EDK is defined by HNImage handle in NImage module. It is an encapsulation of a memory block that stores image pixels. The memory block is organized as rows that follow each other in top-to-bottom order. The number of rows is equal to height of image. Each row is organized as pixels that follow each other in left-to-right order. The number of pixels in a row is equal to width of image. A pixel format describes how image pixels are stored. See `NImageGetWidth`, `NImageGetHeight`, `NImageGetStride`, `NImageGetPixelFormat` and `NImageGetPixels` functions for more information.

An image can have horizontal and vertical resolution attributes assigned to it if they are applicable (they are required for image, and do not make sense for face image). See `NImageGetHorzResolution` and `NImageGetVertResolution` functions for more information.

An image can be created either as empty or from existing memory block. See `NImageCreate`, `NImageCreateFromData` and `NImageCreateWrapper` functions for more information.

For each value of NPixelFormat exposed via interface a module is provided for managing according type of image (getting and setting individual pixels, etc.). See NGrayscaleImage, NMonochromeImage and NRgbImage modules for more information.

An image can be converted to different pixel format using `NImageCreateFromImage` or `NImageCreateFromImageEx` function.

Different methods should be used to display an image on different platforms:

- On Windows `BmpSaveImageToHBitmap` function can be used to receive a standard Win32 HBITMAP for the image. The reverse process is also possible using `BmpLoadImageFromHBitmap` function.
- On Linux there is no easy method implemented. However, a memory block containing pixels of image could be accessed via `NImageGetPixelFormat` function. The memory block can be used to display the image or convert it to some other representation on any platform.

An image can be stored in file in any supported image format using `NImageSaveToFile` function.

An image stored in file in any supported image format can be loaded using `NImageCreateFromFile` function.

3.1.2. Image Format

Image format is a specification of image storage in a file. The specification may require to compress/decompress image during writing/reading it to/from a file.

Image format in the FaceCell EDK is defined by HNImageFormat handle in NImageFormat module.

There is a number of image formats supported in the FaceCell EDK. Certain formats could not be read from and written to a file on all platforms. See the following table for details.

Image Format	**Can read**	**Can write**
BMP	Yes	Yes
GIF	No	No
JPEG	Yes	Yes
PNG	No	No
TIFF	Yes	No

These image formats are accessible using `NImageFormatGetBmp` and `NImageFormatGetTiff` functions.

To find out which images formats are supported in the FaceCell EDK in version-independent way `NImageFormatGetFormatCount` and `NImageFormatGetFormat` functions should be used.

Name, file name pattern (file filter) and default file extension of the image format can be retrieved using `NImageFormatGetName`, `NImageFormatGetFileFilter` and `NImageFormatGetDefaultFileExtension` functions.

To find out which image format should be used to read or write a particular file `NImage-`

`FormatSelect` function should be used.

An image can be loaded and saved from/to file or memory buffer using `NImageFormat-LoadImageFromFile`, `NImageFormatLoadImageFromMemory`, `NImageFormat-SaveImageToFile` and `NImageFormatSaveImageToMemory` functions. Note that not all image formats support both reading and writing. Use `NImageFormatCanRead` and/or `NImageFormatCanWrite` function(s) to check if the particular image format does.

3.1.3. Low-Level Image Input-Output

Low-level image I/O in the FaceCell EDK is implemented in Bmp and Tiff modules.

These modules provides functions for loading and saving images in according format (BMP and TIFF).

Those functions can take parameters that precisely control loading and saving of the image in particular formats.

3.2. Matching threshold and similarity

FaceCell features matching algorithm provides value of features collections similarity as a result. The higher is similarity, the higher is probability that features collections are obtained from the same person.

Matching threshold is linked to false acceptance rate (FAR, different subjects erroneously accepted as of the same) of matching algorithm. The higher is threshold, the lower is FAR and higher FRR (false rejection rate, same subjects erroneously accepted as different) and vice a versa.

FAR	Threshold
1%	0.625
0.1%	0.650
0.01%	0.675
0.001%	0.700
0.0001%	0.725
0.00001%	0.750

3.3. Face Image Constraints

3.3.1. Face image constraints

Face recognition is very sensitive to image quality so maximum care should be attributed to image acquisition.

3.3.1.1. Pose

The frontal pose (full-face) must be used. Rotation of the head must be less than ±5 degrees from frontal in every direction - nodded up/down, rotated left/right, tilted right/left.

3.3.1.2. Expression

The expression should be neutral (non-smiling) with both eyes open, and mouth closed. Every effort should be made to have supplied images comply with this specification. A smile with closed jaw is allowed but not recommended.

3.3.1.2.1. Examples of Non-Recommended Expressions

- A smile where the inside of the mouth is exposed (jaw open).
- Raised eyebrows.
- Closed eyes.
- Eyes looking away from the camera.
- Squinting.
- Frowning.
- Hair covering eyes.
- Rim of glasses covering part of the eye.

3.3.1.3. Face changes

Beard, moustache and other changeable face features influence face recognition quality and if frequent face changes are typical for some individual, face database should contain e.g. face with beard and cleanly shaved face enrolled with identical ID.

3.3.1.4. Lighting

Lighting must be equally distributed on each side of the face and from top to bottom. There should be no significant direction of the light or visible shadows. Care must be taken to avoid "hot spots". These artifacts are typically caused when one, high intensity, focused light source is used for illumination.

3.3.1.5. Eyeglasses

There should be no lighting artifacts on eyeglasses. This can typically be achieved by increasing the angle between the lighting, subject and camera to 45 degrees or more. If lighting reflections cannot be removed, then the glasses themselves should be removed. (However this

is not recommended as face recognition typically works best when matching people with eyeglasses against themselves wearing the same eyeglasses).

Glasses have to be of clear glass and transparent so the eyes and irises are clearly visible. Heavily tinted glasses are not acceptable.

3.3.1.6. Web cameras

Embedded camera with at least 320 x 240 pixels physical resolution (640 x 480 pixels recommended).

Images should be enrolled and matched using the same camera, as devices have different optical distortions that can influence face recognition performance.

Chapter 4. Tutorials

4.1. About

This chapter of documentation describes and in depth comments tutorial programs sources provided in the FaceCell EDK. It provides you with basic overview of workflow when performing fundamental tasks. Tutorial source code is available for the following programming languages:

1. C tutorials

4.2. C Tutorials

1. Enroll face from an image,
2. Enroll face from a stream,
3. Find all the faces in an image,
4. Find eyes of a face,
5. Verify a face from the image with a template,
6. Identify a face from the image.

4.2.1. Enroll face from an image

This tutorial describes the process of extracting face features from an image using the C programming language. The source for this tutorial can be found in /tutorials/C/EnrollImage.c .

1. FaceCell EDK is made of several parts each performing different subset of whole SDK functions. Before writting any program the dependancies for a particular task must be determined and set.

```
#include <NCore.h>
#include <NImages.h>
#include <FccExtractor.h>
```

2. Next step is to load an image from which face features will be extracted.

```
NResult result;
HNImage image;
char *fileName;
```



```
fileName = /* obtain file name */
result = NImageCreateFromFile(fileName, NULL, &image);
```

3. FaceCell EDK works only with grayscale images. So images must be converted to grayscale before being used with the FaceCell EDK functions:

```
HNImage grayscale;
result = NImageCreateFromImage(npfGrayscale, 0, image, &grayscale);
```

4. All the parameters of extraction routines are stored in an extractor object, so it must be initialized before usage.

```
HFccExtractor extractor;
result = FcceCreate(&extractor);
```

5. Before extracting face features buffer space has to be allocated for these features to be stored. The size of this buffer can be determined this way:

```
NSizeType maxSize;
result = FcceGetMaxTemplateSize(extractor, &maxSize);
```

6. The simplest way to extract face feature templates from an image is to use function that will automatically detect face in an image and return face features template and face detection results.

```
void *template;
FcceDetectionDetails details;
template = (void *) malloc(maxSize);
result = FcceExtract(extractor, grayscale, &details, template, maxSize, &size);
```

7. After extraction the details of the face detection can be accessed and checked this way:

```
if (details.FaceAvailable)
{
        printf("found face:\n");
        printf("\tlocation = (%d, %d), width = %d, height = %d,
                confidence = %.2f\n",
                details.Face.Rectangle.X, details.Face.Rectangle.Y,
                details.Face.Rectangle.Width, details.Face.Rectangle.Height,
                details.Face.Confidence);
}
if (details.EyesAvailable)
{
        printf("\tfound eyes:\n");
        printf("\t\tfirst: location = (%d, %d), confidence = %.2f\n",
                details.Eyes.First.X, details.Eyes.First.Y,
                details.Eyes.FirstConfidence);
        printf("\t\tsecond: location = (%d, %d), confidence = %.2f\n",
                details.Eyes.Second.X, details.Eyes.Second.Y,
```



```
                details.Eyes.SecondConfidence);
}
```

8. After the image has been used and isn't necessary anymore it must be freed:

```
NImageFree(grayscale);
```

9. When the extractor is not necessary, it must be freed:

```
FcceFree(extractor);
```

4.2.2. Enroll face from a stream

This tutorial describes the process of extracting face features from an image stream using the C programming language. The source for this tutorial can be found in /tutorials/C/EnrollStream.c .

1. For includes, image loading, converting to grayscale and extractor initialization, maximum template size see "Enrolling face from an image" tutorial.

2. Before enrolling a person using image stream extractor parameters must be initialized. Currently there is only one parameter to be set - the number of frames from which face features should be extracted. This initialization must be done every time a new persons face features are to be extracted.

```
Int numberOfFrames = /* get the number of frames */
result = FcceExtractStart(extractor, numberOfFrames);
```

3. After initalizing enrollement from stream the frames must be processed in frame by frame fashion, checking the results after each processing step.

```
NSizeType size;
NBool stopped;
stopped = NFalse;
size = 0;
for (i = 0; !stopped && (i < numberOfFrames); ++i)
{
        result = FcceExtractNext(extractor, images[i], &details,
                template, maxSize, &size, &stopped);
}
```

4. For detection results, image and extractor freeing see "Enrolling face from an image" tutorial.

4.2.3. Find all the faces in an image

This tutorial describes the process of finding all the face regions (bounding boxes) in an image using the C programming language. The source for this tutorial can be found in /tutorials/C/FindFaces.c .

1. For includes, image loading, converting to grayscale and extractor initialization, maximum template size see "Enrolling face from an image" tutorial.

2. FaceCell EDK has several functions related to face and eyes finding. First of all all face region have to be found in an image:

```
int faceCount;
FcceFace *faces;
result = FcceDetectFaces(extractor, grayscale, &faceCount, &faces);
```

3. In FaceCell EDK the C function that detects all the faces allocates buffer space for the found faces structures, but this buffer isn't freed automatically, so it must be freed manually:

```
Nfree(faces);
```

4. For detection results, image and extractor freeing see "Enrolling face from an image" tutorial.

4.2.4. Find eyes of a face

This tutorial describes the process of finding face eyes in an image using the C programming language. The source for this tutorial can be found in /tutorials/C/FindEyes.c .

1. For includes, image loading, converting to grayscale and extractor initialization, maximum template size see "Enrolling face from an image" tutorial.

2. For face region detection and remarks see "Finding all the faces in an image" tutorial.

3. After all the face regions have been found in the image, these region can be further searched for face eyes:

```
FcceEyes *eyes;
eyes = new FcceEyes[faceCount];
for (int i = 0; i < faceCount; ++i)
{
        result = FcceDetectEyes(extractor, grayscale, &faces[i], &eyes);
}
```

4. In FaceCell EDK the C function that detects all the faces allocates buffer space for the

found faces structures, but this buffer isn't freed automatically, so it must be freed manually:

```
Nfree(faces);
```

5. For detection results, image and extractor freeing see "Enrolling face from an image" tutorial.

4.2.5. Identify a face from the image

This tutorial describes the process of identifying face features against a database of collected face features using the C programming language. The source for this tutorial can be found in /tutorials/C/Identify.c .

1. For includes, image loading, converting to grayscale and extractor initialization, maximum template size, extracting face features from an image" see "Enrolling face from an image" tutorial.

2. All the parameters of matching routines are stored in a matcher object, so it must be initialized before usage:

```
HFccMatcher matcher;
result = FccmCreate(&matcher);
```

3. FaceCell EDK has several functions for tempalate matching. If one template should be checked against all the database it is advised to use identification functions. To start identification procedure, you must first initialize matcher for this task by telling which template will be matched agains all the database.

```
result = FccmIdentifyStart(matcher, template, size);
```

4. After the matcher has been initialized the templates from the database should be matched one by one with the template that was used for initialization.

```
Int templateCount = /* obtain template count */
void **templates;
Int *sizes;
templates = (void **) malloc(sizeOfTemplate * templateCount);
/* Obtain templates */
for (i = 0; i < templateCount; ++i)
{
        result = FccmIdentifyNext(matcher, templates[i], sizes[i], &score);
}
```

5. After template identification is finished, the memory, taken by the identification routines must be freed:


```
result = FccmIdentifyEnd(matcher);
```

6. When the matcher is not necessary, it must be destroyed:

```
FccmFree(matcher);
```

7. For detection results, image and extractor freeing see "Enrolling face from an image" tutorial.

4.2.6. Verify a face from the image with a template

This tutorial describes the process of verifying face features with another face features using the C programming language. The source for this tutorial can be found in /tutorials/C/Verify.c .

1. For includes, image loading, converting to grayscale and extractor initialization, maximum template size, extracting face features from an image" see "Enrolling face from an image" tutorial. Matcher initalization can be found in "Identifying a template" tutorial.

2. When just two templates have to be matched against each other, verification function of FaceCell EDK should be used:

```
NDouble score;
result = FccmVerify(matcher, template, size, enrolledTemplate,
        enrolledSize, &score);
```

3. For matcher freeing see "Identifying a template" tutorial.

4. For detection results, image and extractor freeing see "Enrolling face from an image" tutorial.

Chapter 5. Samples

Before running samples read Face Image Constraints chapter.

5.1. Windows CE

5.1.1. Pocket PC 2003

This sample application demonstrates the use of FaceCell EDK functions for simple tasks: enrolling face from an image and identifying it against a database. This sample application works only with pictures (no camera support).

5.1.2. Windows Mobile 5.0

This sample application demonstrates the use of FaceCell EDK functions for simple tasks: enrolling face from an image or a camera and identifying it against a database.

Chapter 6. Reference (C/C++)

Libraries

FccExtractor	FccExtractor is a library for finding faces in images and extracting their features into templates.
FccMatcher	Provides functionality to verify and match VeriLook face feature templates.
NCore	Provides infrastructure for Neurotechnologija components.
NImages	Provides functionality for loading, saving and converting images in various formats.
NLRecord	Provides functionality for packing, unpacking and editing Neurotechnologija Face Records.

6.1. FccExtractor Library

FccExtractor is a library for finding faces in images and extracting their features into templates.

Windows

Import library: `FccExtractor.dll.lib`.

DLL: `FccExtractor.dll`.

Requirements:

- `NCore.dll`.
- `NImage.dll`.

Linux

Shared object: `libFccExtractor.so`.

Requirements:

- `libNCore.so`.
- `libNImages.so`.

Modules

FccExtractor	FccExtractor is a part of FccExtractor library intended for human face finding in pictures and extracting their features into VeriLook face feature templates.

6.1.1. FccExtractor Module

FccExtractor is a part of FccExtractor library intended for human face finding in pictures and extracting their features into VeriLook face feature templates.

Header file: `FccExtractor.h` (includes `NCore.h`, `NImages.h`, `NGeometry.h` and `FccExtractorParams.h`)

Functions

`FcceCopyParameters`	Copies parameter values from one FccExtractor to another.
`FcceCreate`	Creates a FccExtractor.
`FcceDetectEyes`	Finds eyes of a given face region.
`FcceDetectFace`	Finds one face region that best fits extraction requirements from the given image.
`FcceDetectFaces`	Finds all face regions in the given image.
`FcceExtract`	Fully automatically extracts feature template from the face that best fits extraction requirements from the given image.
`FcceExtractNext`	Extracts one features template from image sequence
`FcceExtractStart`	Initializes features extraction routines that uses image sequences to extract features.
`FcceExtractUsingEyes`	Extracts feature template from the face with given eye coordinates from an image.
`FcceFree`	Deletes the FccExtractor. After the object is deleted the specified handle is no longer valid.
`FcceGeneralize`	Generates generalized template from a set of templates.
`FcceGetMaxTemplateSize`	Retrieves maximal size of feature template

	that the specified FccExtractor can extract.
FcceGetParameter	Retrieves value of the specified parameter of the specified FccExtractor.
FcceIsRegistered	Checks if FccExtractor library is registered.
FcceReset	Sets default values for all parameters of the specified FccExtractor.
FcceSetParameter	Sets value of the specified parameter of the specified FccExtractor.

Types

HFccExtractor	Handle to FccExtractor object.

Structures

FcceDetectionDetails	Structure with information about face detection results in a face detection routine.
FcceEyes	Structure defining information of an eye pair.
FcceFace	Structure describing face region information

Macros

FCCEP_FACE_CONFIDENCE_THRESHOLD	Identifier of type N_TYPE_DOUBLE.
FCCEP_COPYRIGHT	Identifier specifying library copyright static read-only parameter of type N_TYPE_STRING.
FCCEP_GENERALIZATION_THRESHOLD	Identifier of type N_TYPE_DOUBLE. Has the same meaning for features generalization as FCCMP_MATCHING_THRESHOLD parameter for features matching.
FCCEP_MAX_IOD	Identifier of type N_TYPE_INT specifying maximum distance between eyes in face. Faces which have greater distance between

	eyes than this parameter, wont be returned by the face detection routines.
FCCEP_MIN_IOD	Identifier of type N_TYPE_INT specifying minimum distance between eyes in face. Faces which have smaller distance between eyes than this parameter, wont be returned by the face detection routines.
FCCEP_NAME	Identifier specifying library name static read-only parameter of type N_TYPE_STRING.
FCCEP_VERSION_HIGH	Identifier specifying high part of library version static read-only parameter of type N_TYPE_UINT. Two high-order bytes of parameter value specify major version and two low-order bytes - minor version.
FCCEP_VERSION_LOW	Identifier specifying low part of library version static read-only parameter of type N_TYPE_UINT. Two high-order bytes of parameter value specify major (build) version and two low-order bytes - minor (release) version.

See Also

FccExtractor Library

6.1.1.1. FcceEyes structure

Structure defining information of an eye pair.

```
typedef struct FcceEyes_ { } FcceEyes;
```

Fields

First	First eye coordinates.
Second	Second eye coordinates.
FirstConfidence	How confidently the first eye was found.
SecondConfidence	How confidently the second eye was found.

See Also

FccMatcher Module

6.1.1.1.1. FcceEyes.First Field

First eye coordinates.

```
NPoint First;
```

See Also

FcceEyes

6.1.1.1.2. FcceEyes.Second Field

Second eye coordinates.

```
NPoint Second;
```

See Also

FcceEyes

6.1.1.1.3. FcceEyes.FirstConfidence Field

How confidently the first eye was found.

```
NDouble FirstConfidence;
```

See Also

FcceEyes

6.1.1.1.4. FcceEyes.SecondConfidence Field

How confidently the second eye was found.

```
NDouble SecondConfidence;
```

See Also

FcceEyes

6.1.1.2. FcceFace structure

Structure describing face region information


```
typedef struct FcceFace_ { } FcceFace;
```

Fields

Rectangle	Structure with face rectangle information.
Confidence	How confidently the face region was found.

See Also

FccMatcher Module

6.1.1.2.1. FcceFace.rectangle Field

Structure with face rectangle information.

```
NRect Rectangle;
```

See Also

FcceFace

6.1.1.2.2. FcceFace.confidence Field

How confidently the face region was found.

```
NDouble Confidence;
```

See Also

FcceFace

6.1.1.3. FcceDetectionDetails structure

Structure with information about face detection results in a face detection routine.

```
typedef struct FcceDetectionDetails_ { } FcceDetectionDetails;
```

Fields

FaceAvailable	Variable showing if a face region was found in an image.
Face	Structure with face region information if face was found.

EyesAvailable	Variable showing if eyes were found in the face region pointed by the *face*.
Eyes	Structure with found eyes information.

See Also

FccMatcher Module

6.1.1.3.1. FcceDetectionDetails.FaceAvailable Field

Variable showing if a face region was found in an image.

```
NBool FaceAvailable;
```

See Also

FcceDetectionDetails

6.1.1.3.2. FcceDetectionDetails.Face Field

Structure with face region information if face was found.

```
FcceFace Face;
```

See Also

FcceDetectionDetails

6.1.1.3.3. FcceDetectionDetails.EyesAvailable Field

Variable showing if eyes were found in the face region pointed by the *face*.

```
NBool EyesAvailable;
```

See Also

FcceDetectionDetails

6.1.1.3.4. FcceDetectionDetails.Eyes Field

Structure with found eyes information.

```
FcceEyes Eyes;
```

See Also

FcceDetectionDetails

6.1.1.4. FcceCopyParameters Function

Copies parameter values from one FccExtractor to another.

```
NResult N_API FcceCopyParameters(
        HFccExtractor hDstExtractor,
        HFccExtractor hSrcExtractor
);
```

Parameters

hDstExtractor	[in] Handle to the destination FccExtractor object.
hSrcExtractor	[in] Handle to the source FccExtractor object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hDstExtractor* or *hSrcExtractor* is NULL.

See Also

FccExtractor Module | HFccExtractor

6.1.1.5. FcceCreate Function

Creates a FccExtractor.

```
NResult N_API FcceCreate(
        HFccExtractor * pHExtractor
);
```

Parameters

pHExtractor	[out] Pointer to HFccExtractor that receives

	handle to created FccExtractor object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pHExtractor* is NULL.
N_E_OUT_OF_MEMORY	There was not enough memory.

Remarks

Created object must be deleted using `FcceFree` function.

See Also

FccExtractor Module | HFccExtractor | `FcceFree`

6.1.1.6. FcceDetectEyes Function

Finds eyes of a given face region.

```
NResult N_API FcceDetectEyes(
        HFccExtractor hExtractor,
        HNImage hImage,
        FcceFace *pFace,
        FcceEyes *pEyes
);
```

Parameters

hExtractor	[in] Handle to the FccExtractor object.
hImage	[in] Handle to the source image.
pFace	[in] Pointer to the FcceFace structure of a face found in the image hImage.
pEyes	[out] Pointer to the FcceEyes structure of eyes pair found in the face region pFace.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hExtractor* or *hImage* or *pFace* or *pEyes* is NULL.

See Also

FccExtractor Module | HFccExtractor

6.1.1.7. FcceDetecFace Function

Finds one face region that best fits extraction requirements from the given image.

```
NResult N_API FcceDetectFace(
        HFccExtractor hExtractor,
        HNImage hImage,
        NBool *pDetected,
        FcceFace *pFace
);
```

Parameters

hExtractor	[in] Handle of the FccExtractor object.
hImage	[in] Handle of the source image.
pDetected	[out] Pointer to NBool variable that is set to true if face is found in the image else it's false
pFace	[out] Pointer to the FcceFace structure of found face.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hExtractor* or *hImage* or *pDetected* or *pFace* is NULL.

See Also

FccExtractor Module | HFccExtractor

6.1.1.8. FcceDetectFaces Function

Finds all face regions in the given image.

```
NResult N_API FcceDetectFaces(
        HFccExtractor hExtractor,
        HNImage hImage,
        NInt *pFaceCount,
        FcceFace **pArFaces
);
```

Parameters

hExtractor	[in] Handle to the FccExtractor object.
hImage	[in] Handle to the source image.
pFaceCount	[out] Pointer to the NInt variable that is set to the number of found faces in the image.
pArFaces	[out] pointer to an array of FcceFace structures of found faces.

Remarks

The buffer for the pArFaces variable is allocated inside this function automatically, but you must free this variable yourself when it becomes unecessary for you. It must be freed using NCore library function NFree.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hExtractor* or *hImage* or *pFace-*

Error Code	Condition
	Count or *pArFaces* is NULL.

See Also

FccExtractor Module | HFccExtractor

6.1.1.9. FcceExtract Function

Fully automatically extracts feature template from the face that best fits extraction requirements from the given image.

```
NResult N_API FcceExtract(
        HFccExtractor hExtractor,
        HNImage hImage,
        FcceDetectionDetails pDetectionDetails,
        void *pBuffer,
        NSizeType bufferSize,
        NSizeType *pSize
);
```

Parameters

hExtractor	[in] Handle to the FccExtractor object.
hImage	[in] Handle to the source image.
pDetectionDetails	[out] Pointer to the FcceDetectionDetails structure of face detection results in the image
pBuffer	[out] Address to the memory space allocated for the extracted features.
bufferSize	[in] Size of the allocated buffer for the extracted features.
pSize	[out] Returns number of bytes filled in the buffer.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hExtractor* or *hImage* or *pDetectionDetails* or *pBuffer* or *pSize* is NULL.

See Also

FccExtractor Module | HFccExtractor

6.1.1.10. FcceExtractNext Function

Extracts one features template from image sequence

```
NResult N_API FcceExtractNext(
        HFccExtractor hExtractor,
        HNImage hImage,
        FcceDetectionDetails pDetectionDetails,
        void *pBuffer,
        NSizeType bufferSize,
        NSizeType *pSize,
        NBool *pStopped,
);
```

Parameters

hExtractor	[in] Handle to the FccExtractor object.
hImage	[in] Handle to the source image.
pDetectionDetails	[out] Pointer to the FcceDetectionDetails structure of face detection results in the image
pBuffer	[out] address to the memory space allocated for the extracted features.
bufferSize	[in] Size of the allocated buffer for the extracted features.
pSize	[out] Returns number of bytes filled in the buffer.
stopped	[out] Indicates the finish of feature extraction from image set.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hExtractor* or *hImage* or *pBuffer* or *pSize* or *pStopped* is NULL.
N_E_INVALID_OPERATION	FcceExtractStart() was not called before FcceExtractNext().

See Also

FccExtractor Module | HFccExtractor

6.1.1.11. FcceExtractStart Function

Initializes features extraction routines that uses image sequences to extract features.

```
NResult N_API FcceExtractStart(
        HFccExtractor hExtractor,
        NInt durationInFrames
);
```

Parameters

hExtractor	[in] Handle to the FccExtractor object.
durationInFrames	[in] Maximum number of frames which will be used for one feature template extraction.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hExtractor* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*durationInFrames* is smaller than 1.

See Also

FccExtractor Module | HFccExtractor

6.1.1.12. FcceExtractUsingEyes Function

Extracts feature template from the face with given eye coordinates from an image.

```
NResult N_API FcceExtractUsingEyes(
        HFccExtractor hExtractor,
        HNImage hImage,
        FcceEyes *pEyes,
        void *pBuffer,
        NSizeType bufferSize,
        NSizeType *pSize
);
```

Parameters

hExtractor	[in] Handle to the FccExtractsor object.
hImage	[in] Handle to the source image.
pEyes	[in] Pointer to FcceEyes structure of the face eye coordinates in the image.
pBuffer	[out] Address to the memory space allocated for the Extracted features.
bufferSize	[in] Size of the allocated buffer for the Extracted features.
pSize	[out] Returns number of bytes filled in the buffer.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hExtractor* or *hImage* or *pBuffer* or *pSize* or *pEyes* is NULL.

See Also

FccExtractUsingEyesor Module | HFccExtractUsingEyesor

6.1.1.13. FcceFree Function

Deletes the FccExtractor. After the object is deleted the specified handle is no longer valid.

```
void N_API FcceFree(
        HFccExtractor hExtractor
);
```

Parameters

hExtractor	[in] Handle to the FccExtractor object.

Remarks

If *hExtractor* is NULL, does nothing.

See Also

FccExtractor Module | HFccExtractor

6.1.1.14. FcceGeneralize Function

Generates generalized template from a set of templates.

```
NResult N_API FcceGeneralize(
        HFccExtractor hExtractor,
        NInt templateCount,
        const void **arTemplates,
        const NSizeType *arTemplateSizes,
        void *pBuffer,
        NSizeType bufferSize,
        NSizeType *pSize
);
```

Parameters

hExtractoror	[in] Handle to the FccExtractor object.
templateCount	[in] Count of templates to be used in generalization.
arTemplates	[in] pointer to array of the templates to be used in generalization.
pBuffer	[out] address to the memory space allocated for the generalized features.
bufferSize	[in] Size of the allocated buffer for the ex-

	tracted features.
pSize	[out] Returns number of bytes filled in the buffer.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hExtractor* or *arTemplates* or *arTemplateSizes* or *pBuffer* or *pSize* is NULL.
N_E_ARGUMENT	One or more of templates in *arTemplates* might be NULL.

See Also

FccExtractor Module | HFccExtractor

6.1.1.15. FcceGetMaxTemplateSize Function

Retrieves maximal size of feature template that the specified FccExtractor can extract.

```
NResult N_API FcceGetMaxTemplateSize(
        HFccExtractor hExtractor,
        NSizeType * pSize
);
```

Parameters

hExtractor	[in] Handle to the FccExtractor object.
pSize	[out] Pointer to NSizeType that receives maximal template size.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hExtractor* or *pSize* is NULL.

See Also

FccExtractor Module | HFccExtractor

6.1.1.16. FcceGetParameter Function

Retrieves value of the specified parameter of the specified FccExtractor.

```
NResult N_API FcceGetParameter(
        HFccExtractor hExtractor,
        NUInt parameterId,
        void * pValue
);
```

Parameters

hExtractor	[in] Handle to the FccExtractor object. Can be NULL if retrieving static parameter value.
parameterId	[in] Identifier of the parameter to retrieve.
pValue	[out] Pointer to variable that receives parameter value.

Return Values

If the function succeeds and *parameterId* specifies a N_TYPE_STRING type parameter, and *pValue* is NULL, the return value is length of the string (not including the NULL-terminator) *pValue* should point to.

If the function succeeds and *pValue* is not NULL, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*parameterId* specifies a non-static parameter and *hExtractor* is NULL. - or - *parameterId* specifies a

Error Code	Condition
	N_TYPE_STRING type parameter and *pValue* is NULL.
N_E_PARAMETER	*parameterId* is invalid.

Remarks

The following values can be used for *parameterId*:

- FCCEP_MIN_IOD
- FCCEP_MAX_IOD
- FCCEP_FACE_CONFIDENCE_THRESHOLD
- FCCEP_GENERALIZATION_THRESHOLD
- FCCEP_NAME
- FCCEP_VERSION_HIGH
- FCCEP_VERSION_LOW
- FCCEP_COPYRIGHT

To learn the type of the parameter pass value obtained with NParameterMakeId macro using N_PC_TYPE_ID code and the parameter id via *parameterId* parameter and pointer to NInt that will receive one of N_TYPE_XXX via *pValue* parameter. *hExtractor* can be NULL in this case.

See Also

FccExtractor Module | HFccExtractor | FcceSetParameter

6.1.1.17. FcceIsRegistered Function

Checks if FccExtractor library is registered.

```
NBool N_API FcceIsRegistered(void);
```

Return Values

NTrue if library is registered, NFalse otherwise.

See Also

FccExtractor Module

6.1.1.18. FcceReset Function

Sets default values for all parameters of the specified FccExtractor.


```
NResult N_API FcceReset(
        HFccExtractor hExtractor
);
```

Parameters

hExtractor	[in] Handle to the FccExtractor object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hExtractor* is NULL.

See Also

FccExtractor Module | HFccExtractor

6.1.1.19. FcceSetParameter Function

Sets value of the specified parameter of the specified FccExtractor.

```
NResult N_API FcceSetParameter(
        HFccExtractor hExtractor,
        NUInt parameterId,
        const void * pValue
);
```

Parameters

hExtractor	[in] Handle to the FccExtractor object. Can be NULL if setting static parameter value.
parameterId	[in] Identifier of the parameter to set.
pValue	[in] Pointer to the parameter value to set.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	Argument value is out of range.
N_E_PARAMETER	Parameter ID is invalid.

Remarks

The following values can be used for *parameterId*:

- FCCEP_MIN_IOD
- FCCEP_MAX_IOD
- FCCEP_FACE_CONFIDENCE_THRESHOLD
- FCCEP_GENERALIZATION_THRESHOLD

To learn the type of the parameter pass value obtained with NParameterMakeId macro using N_PC_TYPE_ID code and the parameter id via *parameterId* parameter and pointer to NInt that will receive one of N_TYPE_XXX via *pValue* parameter. *hExtractor* can be NULL in this case.

See Also

FccExtractor Module | HFccExtractor | `FcceGetParameter`

6.2. FccMatcher Library

Provides functionality to verify and match VeriLook face feature templates.

Windows

Import library: `FccMatcher.dll.lib`.

DLL: `FccMatcher.dll`.

Requirements:

- `NCore.dll`.

Linux

Shared object: `libFccMatcher.so`.

Requirements:

- `libNCore.so`.

Modules

FccMatcher	Provides functionality to identify or verify VeriLook face feature templates.

6.2.1. FccMatcher Module

Provides functionality to identify or verify VeriLook face feature templates.

Header file: `FccMatcher.h` (includes `FccMatcherParams.h` and `FccMatcherTypes.h`)

Functions

`FccmCopyParameters`	Copies parameter values from one FccMatcher to another.
`FccmCreate`	Creates a FccMatcher.
`FccmFree`	Deletes the FccMatcher. After the object is deleted the specified handle is no longer valid.
`FccmGetParameter`	Retrieves value of the specified parameter of the specified FccMatcher.
`FccmIdentifyEnd`	Finalizes identification procedure.
`FccmIdentifyNext`	Verifies one given template with the one that is being identified.
`FccmIdentifyStart`	Initializes identification procedure.
`FccmIsRegistered`	Checks if FccMatcher library is registered.
`FccmReset`	Sets default values for all parameters of the specified FccMatcher.
`FccmSetParameter`	Sets value of the specified parameter of the specified FccMatcher.
`FccmVerify`	Verifies two VeriLook face feature templates and return their similarity score.

Types

HFccMatcher	Handle to FccMatcher object.

Macros

FCCMP_COPYRIGHT	Identifier specifying library copyright static read-only parameter of type N_TYPE_STRING.
FCCMP_MATCHING_THRESHOLD	Template matching threshold. For possible values of this parameter see Thresholds and similarities section.
FCCMP_NAME	Identifier specifying library name static read-only parameter of type N_TYPE_STRING.
FCCMP_VERSION_HIGH	Identifier specifying high part of library version static read-only parameter of type N_TYPE_UINT. Two high-order bytes of parameter value specify major version and two low-order bytes - minor version.
FCCMP_VERSION_LOW	Identifier specifying low part of library version static read-only parameter of type N_TYPE_UINT. Two high-order bytes of parameter value specify major (build) version and two low-order bytes - minor (release) version.

See Also

FccMatcher Library

6.2.1.1. FccmCopyParameters Function

Copies parameter values from one FccMatcher to another.

```
NResult N_API FccmCopyParameters(
        HFccMatcher hDstMatcher,
        HFccMatcher hSrcMatcher
);
```

Parameters

hDstMatcher	[in] Handle to the destination FccMatcher object.

hSrcMatcher	[in] Handle to the source FccMatcher object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hDstMatcher* or *hSrcMatcher* is NULL.

See Also

FccMatcher Module | HFccMatcher

6.2.1.2. FccmCreate Function

Creates a FccMatcher.

```
NResult N_API FccmCreate(
        HFccMatcher * pHMatcher
);
```

Parameters

pHMatcher	[out] Pointer to HFccMatcher that receives handle to created FccMatcher object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pHMatcher* is NULL.
N_E_OUT_OF_MEMORY	There was not enough memory.

Remarks

Created object must be deleted using `FccmFree` function.

See Also

FccMatcher Module | HFccMatcher | `FccmFree`

6.2.1.3. FccmFree Function

Deletes the FccMatcher. After the object is deleted the specified handle is no longer valid.

```
void N_API FccmFree(
        HFccMatcher hMatcher
);
```

Parameters

hMatcher	[in] Handle to FccMatcher object.

Remarks

If *hMatcher* is NULL, does nothing.

See Also

FccMatcher Module | HFccMatcher

6.2.1.4. FccmGetParameter Function

Retrieves value of the specified parameter of the specified FccMatcher.

```
NResult N_API FccmGetParameter(
        HFccMatcher hMatcher,
        NUInt parameterId,
        void * pValue
);
```

Parameters

hMatcher	[in] Handle to the FccMatcher object. Can be NULL if retrieving static parameter value.
parameterId	[in] Identifier of the parameter to retrieve.
pValue	[out] Pointer to variable that receives parameter value.

Return Values

If the function succeeds and *parameterId* specifies a N_TYPE_STRING type parameter, and *pValue* is NULL, the return value is length of the string (not including the NULL-terminator) *pValue* should point to.

If the function succeeds and *pValue* is not NULL, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*parameterId* specifies a non-static parameter and *hMatcher* is NULL. - or - *parameterId* specifies a non-N_TYPE_STRING type parameter and *pValue* is NULL.
N_E_PARAMETER	*parameterId* is invalid.

Remarks

The following values can be used for *parameterId*:

- FCCMP_MATCHING_THRESHOLD
- FCCMP_NAME
- FCCMP_COPYRIGHT
- FCCMP_VERSION_HIGH
- FCCMP_VERSION_LOW

To learn the type of the parameter pass value obtained with NParameterMakeId macro using N_PC_TYPE_ID code and the parameter id via *parameterId* parameter and pointer to NInt that will receive one of N_TYPE_XXX via *pValue* parameter. *hMatcher* can be NULL in this case.

See Also

FccMatcher Module | HFccMatcher | FccmSetParameter

6.2.1.5. FccmIdentifyStart Function

Initializes identification procedure.

```
NResult N_API FccmIdentifyStart(
        HFccMatcher hMatcher,
```



```
        const void *pTemplate,
        NSizeType templateSize,
);
```

Parameters

hMatcher	[in] Handle to the FccMatcher object.
pTemplate	[in] Pointer to memory buffer containing face features template.
templateSize	[in] Size of given template.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hMatcher* or *pTemplate* is NULL or wrong size.
N_E_INVALID_OPERATION	Identification is already started.

Remarks

FccmIdentifyStart takes the template to be identified, which is then matched against the templates from the user database using `FccmIdentifyNext`.

See Also

FccMatcher Module | HFccMatcher | `FccmIdentifyNext` | `FccmIdentifyEnd`

6.2.1.6. FccmIdentifyNext Function

Verifies one given template with the one that is being identified.

```
NResult N_API FccmIdentifyNext(
        HFccMatcher hMatcher,
        const void * pTemplate,
        NSizeType templateSize,
        NDouble * pScore
);
```

Parameters

hMatcher	[in] Handle to the FccMatcher object.
pTemplate	[in] Pointer to memory buffer containing face features template.
templateSize	[in] Size of given template.
pScore	[out] Pointer to NDouble that receives similarity score.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hMatcher*, *pTemplate*, or *pScore* is NULL or pTemplate is wrong size.
N_E_INVALID_OPERATION	Identification is not started.

Remarks

Value received via *pScore* is zero if similarity is less than matching threshold, i.e. two templates do not match (see FCCMP_MATCHING_THRESHOLD and FccmSetParameter function), and is greater than or equal to matching threshold otherwise.

See Also

FccMatcher Module | HFccMatcher | FCCMP_MATCHING_THRESHOLD | FccmSetParameter | FccmIdentifyStart | FccmIdentifyEnd

6.2.1.7. FccmIdentifyEnd Function

Finalizes identification procedure.

```
NResult N_API FccmIdentifyEnd(
        HFccMatcher hMatcher
);
```

Parameters

hMatcher	[in] Handle to the FccMatcher object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hMatcher* is NULL.
N_E_INVALID_OPERATION	Identification is not started.

See Also

FccMatcher Module | HFccMatcher | FccmIdentifyStart | FccmIdentifyNext

6.2.1.8. FccmIsRegistered Function

Checks if FccMatcher library is registered.

```
NBool N_API FccmIsRegistered(void);
```

Return Values

NTrue if library is registered, NFalse otherwise.

See Also

FccMatcher Module

6.2.1.9. FccmReset Function

Sets default values for all parameters of the specified FccMatcher.

```
NResult N_API FccmReset(
        HFccMatcher hMatcher
);
```

Parameters

hMatcher	[in] Handle to VlMatcher object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hMatcher* is NULL.

See Also

FccMatcher Module | HFccMatcher

6.2.1.10. FccmSetParameter Function

Sets value of the specified parameter of the specified FccMatcher.

```
NResult N_API FccmSetParameter(
        HFccMatcher hMatcher,
        NUInt parameterId,
        const void * pValue
);
```

Parameters

hMatcher	[in] Handle to the FccMatcher object. Can be NULL if setting static parameter value.
parameterId	[in] Identifier of the parameter to set.
pValue	[in] Pointer to the parameter value to set.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	Value *pValue* points to is invalid.
N_E_ARGUMENT_NULL	*parameterId* specifies a non-static parameter and *hMatcher* is NULL. - or - *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	Value *pValue* points to is out of range.

Error Code	Condition
N_E_INVALID_OPERATION	*hMatcher* is not NULL and identification is started on the matcher.
N_E_PARAMETER	*parameterId* is invalid.
N_E_PARAMETER_READ_ONLY	*parameterId* specifies read-only parameter.

Remarks

The following values can be used for *parameterId*:

- FCCMP_MATCHING_THRESHOLD

To learn the type of the parameter pass value obtained with NParameterMakeId macro using N_PC_TYPE_ID code and the parameter id via *parameterId* parameter and pointer to NInt that will receive one of N_TYPE_XXX via *pValue* parameter. *hMatcher* can be NULL in this case.

See Also

FccMatcher Module | HFccMatcher | FccmGetParameter

6.2.1.11. FccmVerify Function

Verifies two VeriLook face feature templates and return their similarity score.

```
NResult N_API FccmVerify(
        HFccMatcher hMatcher,
        const void * pTtemplate1,
        NSizeType template1Size,
        const void * pTemplate2,
        NSizeType template2Size,
        NDouble * pScore
);
```

Parameters

hMatcher	[in] Handle to the FccMatcher object.
template1	[in] Pointer to memory buffer containing first face features template.
template1Size	[in] Size of first face features template.
template2	[in] Pointer to memory buffer containing

	second face features template.
template2Size	[in] Size of second face features template.
pScore	[out] Pointer to NDouble that receives similarity score.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hMatcher* or *pTemplate1 or pTemplate2* is NULL or pTemplate1 or pTemplate2 is wrong size.
N_E_INVALID_OPERATION	Identification is already started.

Remarks

Value received via *pScore* is zero if similarity is less than matching threshold, i.e. two face templates do not match (see FCCMP_MATCHING_THRESHOLD and FccmSetParameter function), and is greater than or equal to matching threshold otherwise.

See Also

FccMatcher Module | HFccMatcher | FCCMP_MATCHING_THRESHOLD | FccmSetParameter | FccmIdentifyStart | FccmIdentifyNext | FccmIdentifyEnd

6.3. NCore Library

Provides infrastructure for Neurotechnologija components.

Modules

NCore	Provides infrastructure/basic functionality for Neurotechnologija components.
NErrors	Defines error codes used in Neurotechnologija components.
NGeometry	Provides definitions of geometrical structures types.

NLibraryInfo	Provides definitions of library info structure type.
NMemory	Provides memory management for Neurotechnologija components.
NParameters	Provides functionality for working with parameters for Neurotechnologija components.
NTypes	Defines types and macros used in Neurotechnologija components.

6.3.1. NCore Module

Provides infrastructure/basic functionality for Neurotechnologija components.

Header file: `NCore.h`.

See Also

NCore Library

6.3.2. NErrors Module

Defines error codes used in Neurotechnologija components.

Header file: `NErrors.h`.

Macros

0	N_OK	No error.
-1	N_E_FAILED	Unspecified error has occurred.
-2	N_E_CORE	Standard error has occurred (for internal use).
-3	N_E_NULL_REFERENCE	Null reference has occurred (for internal use).
-4	N_E_OUT_OF_MEMORY	There were not enough memory.
-5	N_E_NOT_IMPLEMENTED	Functionality is not implemented.
-6	N_E_NOT_SUPPORTED	Functionality is not supported.
-7	N_E_INVALID_OPERATION	Attempted to perform invalid operation.

-8	N_E_OVERFLOW	Arithmetic overflow has occurred.
-9	N_E_INDEX_OUT_OF_RANGE	Index is out of range (for internal use).
-10	N_E_ARGUMENT	Argument is invalid.
-11	N_E_ARGUMENT_NULL	Argument value is NULL where non-NULL value was expected.
-12	N_E_ARGUMENT_OUT_OF_RANGE	Argument value is out of range.
-13	N_E_FORMAT	Format of argument value is invalid.
-14	N_E_IO	Input/output error has occurred.
-15	N_E_END_OF_STREAM	Attempted to read file or buffer after its end.
-90	N_E_EXTERNAL	Error in external code has occured (for internal use).
-91	N_E_WIN32	Win32 error has occured.
-92	N_E_COM	COM error has occured.
-93	N_E_CLR	CLR exception has occured.
-100	N_E_PARAMETER	Parameter ID is invalid.
-101	N_E_PARAMETER_READ_ONLY	Attempted to set read only parameter.
-200	N_E_NOT_REGISTERED	Module is not registered.
	NFailed	Determines whether function result indicates error.
	NSucceeded	Determines whether function result indicates success.

See Also

NCore Library

6.3.3. NGeometry Module

Provides definitions of geometrical structures types.

Header file: `NGeometry.h`.

Structures

NPoint	Structure defining point coordinates in 2D space.
NSize	Structure defining rectangle size.
NRect	Structure defining a rectangle figure in 2D space.

See Also

NCore Library

6.3.3.1. NPoint structure

Structure defining point coordinates in 2D space.

```
typedef struct NPoint_ { } NPoint;
```

Fields

X	Point coordinate on x axis.
Y	Point coordinate on y axis.

See Also

NGeometry

6.3.3.1.1. NPoint.X Field

Point coordinate on x axis.

```
NInt X;
```

See Also

NPoint

6.3.3.1.2. NPoint.Y Field

Point coordinate on y axis.

```
NInt Y;
```


See Also

NPoint

6.3.3.2. NSize structure

Structure defining rectangle size.

```
typedef struct NSize_ { } NSize;
```

Fields

Width	Width.
Height	Height.

See Also

NGeometry

6.3.3.2.1. NSize.Width Field

Width.

```
NInt Width;
```

See Also

NSize

6.3.3.2.2. NSize.Height Field

Height.

```
NInt Height;
```

See Also

NSize

6.3.3.3. NRect structure

Structure defining a rectangle figure in 2D space.

```
typedef struct NRect_ { } NRect;
```


Fields

X	Upper left rectangle corner coordinate on x axis.
Y	Upper left rectangle corner coordinate on y axis.
Width	Rectangle width.
Height	Rectangle height.

See Also

NGeometry

6.3.3.3.1. NRect.X Field

Upper left rectangle corner coordinate on x axis.

```
NInt X;
```

See Also

NRect

6.3.3.3.2. NRect.Y Field

Upper left rectangle corner coordinate on y axis.

```
NInt Y;
```

See Also

NRect

6.3.3.3.3. NRect.Width Field

Rectangle width.

```
NInt Width;
```

See Also

NRect

6.3.3.3.4. NRect.Height Field

Rectangle height.

```
NInt Height;
```

See Also

NRect

6.3.4. NLibraryInfo Module

Provides definitions of library info structure type.

Header file: `NLibraryInfo.h`.

Structures

NLibraryInfo	Structure defining information about the library as library title, product name, company name, copyright string and library version.

See Also

NCore Library

6.3.4.1. NLibraryInfo structure

Structure defining information about the library as library title, product name, company name, copyright string and library version.

```
typedef struct NLibraryInfo_ { } NLibraryInfo;
```

Fields

Company	Name of the company that produced the library.
Copyright	Copyright string for the library.
Product	Product name.
Title	Title of the library.
VersionBuild	Build part of the library version.
VersionMajor	Major part of the library version.

VersionMinor	Minor part of the library version.
VersionRevision	Revision part of the library version.

See Also

NLibraryInfo

6.3.4.1.1. NLibraryInfo.Company Field

Name of the company that produced the library.

```
NChar * Company;
```

See Also

NLibraryInfo

6.3.4.1.2. NLibraryInfo.Copyright Field

Copyright string for the library.

```
NChar * Copyright;
```

See Also

NLibraryInfo

6.3.4.1.3. NLibraryInfo.Product Field

Product name.

```
NChar * Product;
```

See Also

NLibraryInfo

6.3.4.1.4. NLibraryInfo.Title Field

Title of the library.

```
NChar * Title;
```

See Also

NLibraryInfo

6.3.4.1.5. NLibraryInfo.VersionBuild Field

Build part of the library version.

```
NInt VersionBuild;
```

See Also

NLibraryInfo

6.3.4.1.6. NLibraryInfo.VersionMajor Field

Major part of the library version.

```
NInt VersionMajor;
```

See Also

NLibraryInfo

6.3.4.1.7. NLibraryInfo.VersionMinor Field

Minor part of the library version.

```
NInt VersionMinor;
```

See Also

NLibraryInfo

6.3.4.1.8. NLibraryInfo.VersionRevision Field

Revision part of the library version.

```
NInt VersionRevision;
```

See Also

NLibraryInfo

6.3.5. NMemory Module

Provides memory management for Neurotechnologija components.

Header file: `NMemory.h`.

Functions

`NAlloc`	Allocates memory block.
`NCAlloc`	Allocates memory block with all bytes set to zero.
`NCompare`	Compares bytes in two memory blocks.
`NCopy`	Copies data between memory blocks.
`NFill`	Sets bytes of memory block to specified value.
`NFree`	Deallocates memory block.
`NMove`	Move data from one memory block to another.
`NReAlloc`	Reallocates memory block.

Macros

NClear	Clears memory block.

See Also

NCore Library

6.3.6. NParameters Module

Provides functionality for working with parameters for Neurotechnologija components.

Header file: `NParameters.h`.

Macros

N_PC_TYPE_ID	Specifies that type id (NInt value, one of N_TYPE_XXX) of the parameter should be retrieved.
NParameterMakeId	Makes parameter id.
N_TYPE_BOOL	Specifies that parameter type is NBool.
N_TYPE_BYTE	Specifies that parameter type is NByte.

N_TYPE_CHAR	Specifies that parameter type is NChar.
N_TYPE_DOUBLE	Specifies that parameter type is NDouble.
N_TYPE_FLOAT	Specifies that parameter type is NFloat.
N_TYPE_INT	Specifies that parameter type is NInt.
N_TYPE_LONG	Specifies that parameter type is NLong.
N_TYPE_SBYTE	Specifies that parameter type is NSByte.
N_TYPE_SHORT	Specifies that parameter type is NShort.
N_TYPE_STRING	Specifies that parameter type is null-terminated string of NChar.
N_TYPE_UINT	Specifies that parameter type is NUInt.
N_TYPE_ULONG	Specifies that parameter type is NULong.
N_TYPE_USHORT	Specifies that parameter type is NUShort.

See Also

NCore Library

6.3.6.1. NParameterMakeId Macro

Makes parameter id.

```
#define NParameterMakeId(code, index, id)
```

Parameters

code	One of N_PC_XXX.
index	Reserved, must be zero.
id	One of the parameter ids provided by a Neurotechnologija module.

See Also

NParameters Module

6.3.7. NTypes Module

Defines types and macros used in Neurotechnologija components.

Header file: `NTypes.h`.

Structures

NIndexPair	Represents a pair of indexes.
NRational	Represents a signed rational number.
NURational	Represents an unsigned rational number.

Enumerations

NByteOrder	Specifies byte order.
NFileAccess	Specifies access to a file.

Types

NAChar	ANSI character (8-bit).
NBool	Same as NBoolean.
NBoolean	32-bit boolean value. See also NTrue and NFalse.
NByte	Same as NUInt8.
NChar	Character type. Either NAChar or NWChar (if N_UNICODE is defined).
NDouble	Double precision floating point number.
NFloat	Same as NSingle.
NHandle	Pointer to unspecified data (same as void *).
NInt	Same as NInt32.
NInt8	8-bit signed integer (signed byte).
NInt16	16-bit signed integer (short).
NInt32	32-bit signed integer (int).
NInt64	64-bit signed integer (long). Not available on some 32-bit platforms.

NLong	Same as NInt64.
NPosType	Platform dependent position type. Signed 64-bit (or 32-bit on some platforms) integer on 32-bit platform, signed 64-bit integer on 64-bit platform).
NResult	Result of a function (same as NInt). See also NErrors module.
NSByte	Same as NInt8.
NShort	Same as NInt16.
NSingle	Single precision floating point number.
NSizeType	Platform dependent size type. Unsigned 32-bit integer on 32-bit platform, unsigned 64-bit integer on 64-bit platform.
NUInt	Same as NUInt32.
NUInt8	8-bit unsigned integer (byte).
NUInt16	16-bit unsigned integer (unsigned short).
NUInt32	32-bit unsigned integer (unsigned int).
NUInt64	64-bit unsigned integer (unsigned long). Not available on some 32-bit platforms.
NULong	Same as NUInt64.
NUShort	Same as NUInt16.
NWChar	Unicode character (16-bit).

Macros

N_64	Defined if compiling for 64-bit architecture.
N_ANSI_C	Defined if ANSI C language compliance is enabled in compiler.
N_API	Defines functions calling convention (stdcall on Windows).
N_BIG_ENDIAN	Defined if compiling for big-endian processor architecture.
N_BYTE_MAX	Maximum value for NByte.

N_BYTE_MIN	Minimum value for NByte.
N_CALLBACK	Defined callbacks calling convention (stdcall on Windows).
N_CALLBACK_AW	Picks either ANSI or Unicode (if N_UNICODE is defined) version of the callback (with either 'A' or 'W' suffix accordingly).
N_CPP	Defined if compiling as C++ code.
N_DECLARE_HANDLE	Declares handle with specified name.
N_DOUBLE_MAX	Maximum value for NDouble.
N_DOUBLE_MIN	Minimum value for NDouble.
N_FUNC_AW	Picks either ANSI or Unicode (if N_UNICODE is defined) version of the function (with either 'A' or 'W' suffix accordingly).
N_GCC	Defined if compiling with GCC.
N_FLOAT_MAX	Maximum value for NFloat.
N_FLOAT_MIN	Minimum value for NFloat.
N_INT_MAX	Maximum value for NInt.
N_INT_MIN	Minimum value for NInt.
N_INT8_MAX	Maximum value for NInt8.
N_INT8_MIN	Minimum value for NInt8.
N_INT16_MAX	Maximum value for NInt16.
N_INT16_MIN	Minimum value for NInt16.
N_INT32_MAX	Maximum value for NInt32.
N_INT32_MIN	Minimum value for NInt32.
N_INT64_MAX	Maximum value for NInt64.
N_INT64_MIN	Minimum value for NInt64.
N_LIB	Defined if compiling static library.
N_LONG_MAX	Maximum value for NLong.
N_LONG_MIN	Minimum value for NLong.

N_MAC	Defined if compiling for Mac OS.
N_MSVC	Defined if compiling with Microsoft Visual C++.
N_NO_ANSI_FUNC	Defined if compiling for platform without ANSI versions of the functions support.
N_NO_FLOAT	Defined if compiling for platform without floating-point support.
N_NO_INT_64	Defined if compiling for platform without 64-bit integer types support.
N_NO_UNICODE	Defined if compiling without Unicode support.
N_POS_TYPE_MIN	Minimum value for NPosType.
N_POS_TYPE_MAX	Maximum value for NPosType.
N_SBYTE_MAX	Maximum value for NSByte.
N_SBYTE_MIN	Minimum value for NSByte.
N_SHORT_MAX	Maximum value for NShort.
N_SHORT_MIN	Minimum value for NShort.
N_SINGLE_MAX	Maximum value for NSingle.
N_SINGLE_MIN	Minimum value for NSingle.
N_SIZE_TYPE_MIN	Minimum value for NSizeType.
N_SIZE_TYPE_MAX	Maximum value for NSizeType.
N_STRUCT_AW	Picks either ANSI or Unicode (if N_UNICODE is defined) version of the struct (with either 'A' or 'W' suffix accordingly).
N_T	Makes either ANSI or Unicode (if N_UNICODE is defined) string or character constant.
N_UINT_MAX	Maximum value for NUInt.
N_UINT_MIN	Minimum value for NUInt.
N_UINT8_MAX	Maximum value for NUInt8.
N_UINT8_MIN	Minimum value for NUInt8.

N_UINT16_MAX	Maximum value for NUInt16.
N_UINT16_MIN	Minimum value for NUInt16.
N_UINT32_MAX	Maximum value for NUInt32.
N_UINT32_MIN	Minimum value for NUInt32.
N_UINT64_MAX	Maximum value for NUInt64.
N_UINT64_MIN	Minimum value for NUInt64.
N_ULONG_MAX	Maximum value for NULong.
N_ULONG_MIN	Minimum value for NULong.
N_UNICODE	Defined if compiling with Unicode character set (affects NChar type).
N_USHORT_MAX	Maximum value for NUShort.
N_USHORT_MIN	Minimum value for NUShort.
N_WINDOWS	Defined if compiling for Windows.
NULL	Null value for pointer.
NFalse	False value for NBoolean.
NIsReverseByteOrder	Checks if specified byte order is reverse to system byte order.
NTrue	True value for NBoolean.

See Also

NCore Library

6.3.7.1. NByteOrder Enumeration

Specifies byte order.

```
typedef enum NByteOrder_ { } NByteOrder;
```

Members

`nboBigEndian`	Big-endian byte order.
`nboLittleEndian`	Little-endian byte order.

nboSystem	System-dependent byte order (either little-endian or big-endian).

See Also

NTypes Module

6.3.7.2. NFileAccess Enumeration

Specifies access to a file.

```
typedef enum NFileAccess_ { } NFileAccess;
```

Members

nfaRead	Read access to the file.
nfaReadWrite	Read and write access to the file.
nfaWrite	Write access to the file.

See Also

NTypes Module

6.3.7.3. NIndexPair Structure

Represents a pair of indexes.

```
typedef struct NIndexPair_ { } NIndexPair;
```

Fields

Index1	First index of this NIndexPair.
Index2	Second index of this NIndexPair.

See Also

NTypes Module

6.3.7.3.1. NIndexPair.Index1 Field

First index of this NIndexPair.

```
NInt Index1;
```

See Also

NIndexPair

6.3.7.3.2. NIndexPair.Index2 Field

Second index of this NIndexPair.

```
NInt Index2;
```

See Also

NIndexPair

6.3.7.4. NRational Structure

Represents a signed rational number.

```
typedef struct NRational_ { } NRational;
```

Fields

Denominator	Denominator of this NRational.
Numerator	Numerator of this NRational.

See Also

NTypes Module

6.3.7.4.1. NRational.Denominator Field

Denominator of this NRational.

```
NInt Denominator;
```

See Also

NRational

6.3.7.4.2. NRational.Numerator Field

Numerator of this NRational.

```
NInt Numerator;
```

See Also

NRational

6.3.7.5. NURational Structure

Represents an unsigned rational number.

```
typedef struct NURational_ { } NURational;
```

Fields

Denominator	Denominator of this NURational.
Numerator	Numerator of this NURational.

See Also

NTypes Module

6.3.7.5.1. NURational.Denominator Field

Denominator of this NURational.

```
NUInt Denominator;
```

See Also

NURational

6.3.7.5.2. NURational.Numerator Field

Numerator of this NURational.

```
NUInt Numerator;
```

See Also

NURational

6.4. NImages Library

Provides functionality for loading, saving and converting images in various formats.

Modules

Bmp	Provides functionality for loading and saving images in BMP format.
Jpeg	Provides functionality for loading and saving images in JPEG format.
NGrayscaleImage	Provides functionality for managing 8-bit grayscale images.
NImageFormat	Provides functionality for loading and saving images in format-neutral way.
NImage	Provides functionality for managing images.
NImages	Provides library registration and other additional functionality.
NMonochromeImage	Provides functionality for managing 1-bit monochrome images.
NPixelFormat	Provides functionality for working with image pixel format.
NRgbImage	Provides functionality for managing 24-bit RGB images.
Tiff	Provides functionality for loading images in TIFF format.

6.4.1. Bmp Module

Provides functionality for loading and saving images in BMP format.

Header file: `Bmp.h`.

Functions

`BmpLoadImageFromFile`	Loads image from BMP file.
`BmpLoadImageFromHBitmap`	Loads image from Windows HBITMAP.
`BmpLoadImageFromMemory`	Loads image from memory buffer containing BMP file.
`BmpSaveImageToFile`	Saves image to file in BMP format.

BmpSaveImageToHBitmap	Saves image to Windows HBITMAP.
BmpSaveImageToMemory	Saves image to memory buffer in BMP format.

See Also

NImages Library

6.4.1.1. BmpLoadImageFromFile Function

Loads image from BMP file.

```
NResult N_API BmpLoadImageFromFile(
        const NChar * szFileName,
        HNImage * pHImage
);
```

Parameters

szFileName	[in] Points to string that specifies file name.
pHImage	[out] Pointer to HNImage that receives handle to loaded image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*szFileName* or *pHImage* is NULL.
N_E_FORMAT	Format of file specified by *szFileName* is invalid.

Remarks

This is a low-level function and can be changed in future version of the library.

See Also

Bmp Module | HNImage | BmpLoadImageFromMemory | BmpLoadImageFromHBitmap | BmpSaveImageToFile

6.4.1.2. BmpLoadImageFromHBitmap Function

Note
This function is available only on Windows.

Loads image from Windows HBITMAP.

```
NResult N_API BmpLoadImageFromHBitmap(
        NHandle handle,
        HNImage * pHImage
);
```

Parameters

handle	[in] Handle that specifies Windows HBITMAP.
pHImage	[out] Pointer to HNImage that receives handle to loaded image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*handle* or *pHImage* is NULL.

Remarks

This is a low-level function and can be changed in future version of the library.

See Also

Bmp Module | HNImage | BmpLoadImageFromFile | BmpLoadImageFromMemory | BmpSaveImageToHBitmap

6.4.1.3. BmpLoadImageFromMemory Function

Loads image from memory buffer containing BMP file.


```
NResult N_API BmpLoadImageFromMemory(
        const void * buffer,
        NSizeType bufferLength,
        HNImage * pHImage
);
```

Parameters

buffer	[in] Pointer to memory buffer.
bufferLength	[in] Length of memory buffer.
pHImage	[out] Pointer to HNImage that receives handle to loaded image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*buffer* is NULL and *bufferLength* is not equal to zero. - or - *pHImage* is NULL.
N_E_FORMAT	Format of file contained in buffer specified by *buffer* is invalid.

Remarks

This is a low-level function and can be changed in future version of the library.

See Also

Bmp Module | HNImage | BmpLoadImageFromFile | BmpLoadImageFromHBitmap | BmpSaveImageToMemory

6.4.1.4. BmpSaveImageToFile Function

Saves image to file in BMP format.


```
NResult N_API BmpSaveImageToFile(
        HNImage hImage,
        const NChar * szFileName
);
```

Parameters

hImage	[in] Handle to image.
szFileName	[in] Points to string that specifies file name.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *szFileName* is NULL.

Remarks

This is a low-level function and can be changed in future version of the library.

See Also

Bmp Module | HNImage | BmpSaveImageToMemory | BmpSaveImageToHBitmap | BmpLoadImageFromFile

6.4.1.5. BmpSaveImageToHBitmap Function

> **Note**
> This function is available only on Windows.

Saves image to Windows HBITMAP.

```
NResult N_API BmpSaveImageToHBitmap(
        HNImage hImage,
        NHandle * pHandle
);
```

Parameters

hImage	[in] Handle to image.

pHandle	[out] Pointer to NHandle that receives handle to created Windows HBITMAP.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pHandle* is NULL.

Remarks

This is a low-level function and can be changed in future version of the library.

See Also

Bmp Module | HNImage | BmpSaveImageToFile | BmpSaveImageToMemory BmpLoadImageFromHBitmap

6.4.1.6. BmpSaveImageToMemory Function

Saves image to memory buffer in BMP format.

```
NResult N_API BmpSaveImageToMemory(
        HNImage hImage,
        void * * pBuffer,
        NSizeType * pBufferLength
);
```

Parameters

hImage	[in] Handle to image.
pBuffer	[out] Pointer to void * that receives pointer to allocated memory buffer.
pBufferLength	[out] Pointer to NSizeType that receives size of allocated memory buffer.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage*, *pBuffer* or *pBufferLength* is NULL.
N_E_OUT_OF_MEMORY	There was not enough memory to allocate memory buffer.

Remarks

This is a low-level function and can be changed in future version of the library.

Memory buffer allocated by the function must be deallocated using NFree function when it is no longer needed.

See Also

Bmp Module | HNImage | BmpSaveImageToFile | BmpSaveImageToHBitmap | BmpLoadImageFromMemory

6.4.2. Jpeg Module

Provides functionality for loading and saving images in JPEG format.

Header file: Jpeg.h.

Functions

JpegLoadImageFromFile	Loads image from JPEG file.
JpegLoadImageFromMemory	Loads image from memory buffer containing JPEG file.
JpegSaveImageToFile	Saves image to file in JPEG format with specified bitrate.
JpegSaveImageToMemory	Saves image to memory buffer in JPEG format with specified bitrate.

Macros

JPEG_DEFAULT_QUALITY	Specifies default JPEG quality.

See Also

NImages Library

6.4.2.1. JpegLoadImageFromFile Function

Loads image from JPEG file.

```
NResult N_API JpegLoadImageFromFile(
        const NChar * szFileName,
        HNImage * pHImage
);
```

Parameters

szFileName	[in] Points to string that specifies file name.
pHImage	[out] Pointer to HNImage that receives handle to loaded image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*szFileName* or *pHImage* is NULL.
N_E_FORMAT	Format of file specified by *szFileName* is invalid.

See Also

Jpeg Module | HNImage | `JpegLoadImageFromMemory` | `JpegSaveImageToFile`

6.4.2.2. JpegLoadImageFromMemory Function

Loads image from memory buffer containing JPEG file.

```
NResult N_API JpegLoadImageFromMemory(
        const void * buffer,
        NSizeType bufferLength,
        HNImage * pHImage
);
```


Parameters

buffer	[in] Pointer to memory buffer.
bufferLength	[in] Length of memory buffer.
pHImage	[out] Pointer to HNImage that receives handle to loaded image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*buffer* is NULL and *bufferLength* is not equal to zero. - or - *pHImage* is NULL.
N_E_FORMAT	Format of file contained in buffer specified by *buffer* is invalid.

See Also

Jpeg Module | HNImage | JpegLoadImageFromFile | JpegSaveImageToMemory

6.4.2.3. JpegSaveImageToFile Function

Saves image to file in JPEG format with specified bitrate.

```
NResult N_API JpegSaveImageToFile(
        HNImage hImage,
        NInt quality,
        const NChar * szFileName
);
```

Parameters

hImage	[in] Handle to image.
quality	[in] Specifies quality of JPEG image.

szFileName	[in] Points to string that specifies file name.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *szFileName* is NULL.

See Also

Jpeg Module | HNImage | JPEG_DEFAULT_QUALITY | JpegSaveImageToMemory | JpegLoadImageFromFile

6.4.2.4. JpegSaveImageToMemory Function

Saves image to memory buffer in JPEG format with specified bitrate.

```
NResult N_API JpegSaveImageToMemory(
        HNImage hImage,
        NInt quality,
        void * * pBuffer,
        NSizeType * pBufferLength
);
```

Parameters

hImage	[in] Handle to image.
quality	[in] Specifies quality of JPEG image.
pBuffer	[out] Pointer to void * that receives pointer to allocated memory buffer.
pBufferLength	[out] Pointer to NSizeType that receives size of allocated memory buffer.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage*, *pBuffer* or *pBufferLength* is NULL.
N_E_OUT_OF_MEMORY	There was not enough memory to allocate memory buffer.

Remarks

Memory buffer allocated by the function must be deallocated using `NFree` function when it is no longer needed.

See Also

Jpeg Module | HNImage | JPEG_DEFAULT_QUALITY | `JpegSaveImageToFile` | `JpegLoadImageFromMemory`

6.4.3. NGrayscaleImage Module

Provides functionality for managing 8-bit grayscale images.

Header file: `NGrayscaleImage.h`.

Functions

`NGrayscaleImageGetPixel`	Retrieves value of pixel at the specified co-ordinates in 8-bit grayscale image.
`NGrayscaleImageSetPixel`	Sets value of pixel at the specified coordinates in 8-bit grayscale image.

Remarks

This module provides advanced functionality, such as individual pixel value retrieval for image with pixel format equal to `npfGrayscale`.

See Also

NImages Library | NImage Module

6.4.3.1. NGrayscaleImageGetPixel Function

Retrieves value of pixel at the specified coordinates in 8-bit grayscale image.

```
NResult N_API NGrayscaleImageGetPixel(
```

```
        HNImage hImage,
        NUInt x,
        NUInt y,
        NByte * pValue
);
```

Parameters

hImage	[in] Handle to image.
x	[in] Specifies x-coordinate of the pixel.
y	[in] Specifies y-coordinate of the pixel.
pValue	[out] Points to NByte that receives pixel value.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*x* is greater than or equal to image width. - or - *y* is greater than or equal to image height.
N_E_FORMAT	Image pixel format is not equal to `npf-Grayscale`.

See Also

NGrayscaleImage Module | HNImage | `NGrayscaleImageSetPixel`

6.4.3.2. NGrayscaleImageSetPixel Function

Sets value of pixel at the specified coordinates in 8-bit grayscale image.

```
NResult N_API NGrayscaleImageSetPixel(
        HNImage hImage,
```

```
        NUInt x,
        NUInt y,
        NByte value
);
```

Parameters

hImage	[in] Handle to image.
x	[in] Specifies x-coordinate of the pixel.
y	[in] Specifies y-coordinate of the pixel.
value	[in] Specifies new pixel value.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*x* is greater than or equal to image width. - or - *y* is greater than or equal to image height.
N_E_FORMAT	Image pixel format is not equal to `npf-Grayscale`.

See Also

NGrayscaleImage Module | HNImage | `NGrayscaleImageGetPixel`

6.4.4. NImageFormat Module

Provides functionality for loading and saving images in format-neutral way.

Header file: `NImageFormat.h`.

Functions

NImageFormatCanRead	Retrieves a value indicating whether the image format supports reading.
NImageFormatCanWrite	Retrieves a value indicating whether the image format supports writing.
NImageFormatGetBmp	Retrieves BMP image format.
NImageFormatGetDefaultFileExtension	Retrieves default file extension of the image format.
NImageFormatGetFileFilter	Retrieves file filter of the image format.
NImageFormatGetFormat	Retrieves supported image format with specified index.
NImageFormatGetFormatCount	Retrieves number of supported image formats.
NImageFormatGetName	Retrieves name of the image format.
NImageFormatGetTiff	Retrieves TIFF image format.
NImageFormatLoadImageFromFile	Loads image from file of specified image format.
NImageFormatLoadImageFromMemory	Loads image from the memory buffer containing file of specified image format.
NImageFormatSaveImageToFile	Saves image to the file in specified format.
NImageFormatSaveImageToMemory	Saves image to the memory buffer in specified format.
NImageFormatSelect	Retrieves supported image format registered with file extension of specified file name and supporting reading/writing as specified.

Types

HNImageFormat	Handle to image format.

See Also

NImages Library | NImage Module

6.4.4.1. NImageFormatCanRead Function

Retrieves a value indicating whether the image format supports reading.

```
NResult N_API NImageFormatCanRead(
        HNImageFormat hImageFormat,
        NBool * pValue
);
```

Parameters

hImageFormat	[in] Handle to image format.
pValue	[out] Pointer to NBool that receives value indicating whether the image format supports reading.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImageFormat* or *pValue* is NULL.

See Also

NImageFormat Module | HNImageFormat | `NImageFormatCanWrite`

6.4.4.2. NImageFormatCanWrite Function

Retrieves a value indicating whether the image format supports writing.

```
NResult N_API NImageFormatCanWrite(
        HNImageFormat hImageFormat,
        NBool * pValue
);
```

Parameters

hImageFormat	[in] Handle to image format.
pValue	[out] Pointer to NBool that receives value indicating whether the image format supports writing.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImageFormat* or *pValue* is NULL.

See Also

NImageFormat Module | HNImageFormat | NImageFormatCanRead

6.4.4.3. NImageFormatGetBmp Function

Retrieves BMP image format.

```
NResult N_API NImageFormatGetBmp(
        HNImageFormat * pValue
);
```

Parameters

pValue	[out] Pointer to HNImageFormat that receives handle to image format.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pValue* is NULL.

See Also

NImageFormat Module | HNImageFormat | NImageFormatGetTiff

6.4.4.4. NImageFormatGetDefaultFileExtension Function

Retrieves default file extension of the image format.

```
NResult N_API NImageFormatGetDefaultFileExtension(
        HNImageFormat hImageFormat,
        NChar * pValue
);
```

Parameters

hImageFormat	[in] Handle to image format.
pValue	[out] Pointer to string that receives default file extension of the image format. Can be NULL.

Return Values

If the function succeeds and *pValue* is NULL, the return value is length of the string (not including the NULL-terminator) *pValue* should point to.

If the function succeeds and *pValue* is not NULL, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImageFormat* is NULL.

See Also

NImageFormat Module | HNImageFormat

6.4.4.5. NImageFormatGetFileFilter Function

Retrieves file filter of the image format.

```
NResult N_API NImageFormatGetFileFilter(
        HNImageFormat hImageFormat,
        NChar * pValue
);
```

Parameters

hImageFormat	[in] Handle to image format.
pValue	[out] Pointer to string that receives file filter of the image format. Can be NULL.

Return Values

If the function succeeds and *pValue* is NULL, the return value is length of the string (not including the NULL-terminator) *pValue* should point to.

If the function succeeds and *pValue* is not NULL, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImageFormat* is NULL.

See Also

NImageFormat Module | HNImageFormat

6.4.4.6. NImageFormatGetFormat Function

Retrieves supported image format with specified index.

```
NResult N_API NImageFormatGetFormat(
        NInt index,
        HNImageFormat * pValue
);
```

Parameters

index	[in] Specifies zero-based supported image format index to retrieve.
pValue	[out] Pointer to NImageFormat that receives image format.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*index* is less than zero or greater than or equal to supported image format count. See

Error Code	Condition
	NImageFormatGetFormatCount.

See Also

NImageFormat Module | HNImageFormat | NImageFormatGetFormatCount

6.4.4.7. NImageFormatGetFormatCount Function

Retrieves number of supported image formats.

```
NResult N_API NImageFormatGetFormatCount(
        NInt * pValue
);
```

Parameters

pValue	[out] Pointer to NInt that receives number of supported image formats.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pValue* is NULL.

See Also

NImageFormat Module | HNImageFormat | NImageFormatGetFormat

6.4.4.8. NImageFormatGetName Function

Retrieves name of the image format.

```
NResult N_API NImageFormatGetName(
        HNImageFormat hImageFormat,
        NChar * pValue
);
```


Parameters

hImageFormat	[in] Handle to image format.
pValue	[out] Pointer to string that receives name of the image format. Can be NULL.

Return Values

If the function succeeds and *pValue* is NULL, the return value is length of the string (not including the NULL-terminator) *pValue* should point to.

If the function succeeds and *pValue* is not NULL, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImageFormat* is NULL.

See Also

NImageFormat Module | HNImageFormat

6.4.4.9. NImageFormatGetTiff Function

Retrieves TIFF image format.

```
NResult N_API NImageFormatGetTiff(
        HNImageFormat * pValue
);
```

Parameters

pValue	[out] Pointer to HNImageFormat that receives handle to image format.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pValue* is NULL.

See Also

NImageFormat Module | HNImageFormat | `NImageFormatGetBmp`

6.4.4.10. NImageFormatLoadImageFromFile Function

Loads image from file of specified image format.

```
NResult N_API NImageFormatLoadImageFromFile(
        HNImageFormat hImageFormat,
        const NChar * szFileName,
        HNImage * pHImage
);
```

Parameters

hImageFormat	[in] Handle to image format.
szFileName	[in] Points to string that specifies file name.
pHImage	[out] Pointer to HNImage that receives handle to loaded image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImageFormat*, *szFileName* or *pHImage* is NULL.
N_E_FORMAT	Format of file specified by *szFileName* is invalid for specified image format.
N_E_NOT_SUPPORTED	Image format specified by *hImageFormat* does not support reading.

See Also

NImageFormat Module | HNImageFormat | `NImageFormatCanRead` | HNImage | `NImageFormatLoadImageFromMemory` | `NImageFormatSaveImageToFile`

6.4.4.11. NImageFormatLoadImageFromMemory Function

Loads image from the memory buffer containing file of specified image format.

```
NResult N_API NImageFormatLoadImageFromMemory(
        HNImageFormat hImageFormat,
        void * buffer,
        NSizeType bufferLength,
        HNImage * pHImage
);
```

Parameters

hImageFormat	[in] Handle to image format.
buffer	[in] Pointer to memory buffer.
bufferLength	[in] Length of memory buffer.
pHImage	[out] Pointer to HNImage that receives handle to loaded image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImageFormat* or *pHImage* is NULL. - or - *buffer* is NULL and *bufferLength* is not equal to zero.
N_E_FORMAT	Format of file contained in buffer specified by *buffer* is invalid for specified image format.
N_E_NOT_SUPPORTED	Image format specified by *hImageFormat* does not support reading.

See Also

NImageFormat Module | HNImageFormat | `NImageFormatCanRead` | HNImage | `NImageFormatLoadImageFromFile` | `NImageFormatSaveImageToMemory`

6.4.4.12. NImageFormatSaveImageToFile Function

Saves image to the file in specified format.

```
NResult N_API NImageFormatSaveImageToFile(
        HNImageFormat hImageFormat,
        HNImage hImage,
        const NChar * szFileName
);
```

Parameters

hImageFormat	[in] Handle to image format.
hImage	[in] Handle to image.
szFileName	[in] Points to string that specifies file name.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImageFormat*, *hImage* or *szFileName* is NULL.
N_E_NOT_SUPPORTED	Image format specified by *hImageFormat* does not support writing.

See Also

NImageFormat Module | HNImageFormat | `NImageFormatCanWrite` | HNImage | `NImageFormatSaveImageToMemory` | `NImageFormatLoadImageFromFile`

6.4.4.13. NImageFormatSaveImageToMemory Function

Saves image to the memory buffer in specified format.


```
NResult N_API NImageFormatSaveImageToMemory(
        HNImageFormat hImageFormat,
        HNImage hImage,
        void * * pBuffer,
        NSizeType * pBufferLength
);
```

Parameters

hImageFormat	[in] Handle to image format.
hImage	[in] Handle to image.
pBuffer	[out] Pointer to void * that receives pointer to allocated memory buffer.
pBufferLength	[out] Pointer to NSizeType that receives size of allocated memory buffer.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImageFormat*, *hImage*, *pBuffer* or *pBufferLength* is NULL.
N_E_NOT_SUPPORTED	Image format specified by *hImageFormat* does not support writing.

Remarks

Memory buffer allocated by the function must be deallocated using NFree function when it is no longer needed.

See Also

NImageFormat Module | HNImageFormat | NImageFormatCanWrite | HNImage | NImageFormatSaveImageToFile | NImageFormatLoadImageFromMemory

6.4.4.14. NImageFormatSelect Function

Retrieves supported image format registered with file extension of specified file name and

supporting reading/writing as specified.

```
NResult N_API NImageFormatSelect(
        const NChar * szFileName,
        NFileAccess fileAccess,
        HNImageFormat * pHImageFormat
);
```

Parameters

szFileName	[in] Points to string that file name.
fileAccess	[in] Specifies that image format should support reading, writing or both.
pHImageFormat	[out] Pointer to HNImageFormat that receives handle to image format.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*fileAccess* value is invalid.
N_E_ARGUMENT_NULL	*szFileName* or *pHImageFormat* is NULL.

Remarks

If none of supported image formats that supports reading/writing as specified by *fileAccess* is registered with file extension of *szFileName* then handle returned via *pHImageFormat* is NULL.

See Also

NImageFormat Module | HNImageFormat | NFileAccess | NImageFormatGetFormatCount | NImageFormatGetFormat

6.4.5. NImage Module

Provides functionality for managing images.

Header file: `NImage.h`.

Functions

`NImageClone`	Creates a new image that is a copy of specified image.
`NImageCreate`	Creates an image with specified pixel format, size, stride and resolution.
`NImageCreateFromData`	Creates an image with specified pixel format, size, stride and resolution and copies specified pixels to it.
`NImageCreateFromFile`	Creates (loads) an image from file of specified format.
`NImageCreateFromImage`	Creates an image from specified image with specified pixel format and stride.
`NImageCreateFromImageEx`	Creates an image from specified image with specified pixel format, stride and resolution.
`NImageCreateWrapper`	Creates an image wrapper for specified pixels with specified pixel format, size, stride and resolution.
`NImageFree`	Deletes the image. After the image is deleted the specified handle is no longer valid.
`NImageGetHeight`	Retrieves height of the image.
`NImageGetHorzResolution`	Retrieves horizontal resolution of the image.
`NImageGetPixelFormat`	Retrieves pixel format of the image.
`NImageGetPixels`	Retrieves pointer to memory block containing pixels of the image.
`NImageGetSize`	Retrieves size of memory block containing pixels of the image.
`NImageGetStride`	Retrieves stride (size of one row) of the image.
`NImageGetVertResolution`	Retrieves vertical resolution of the image.
`NImageGetWidth`	Retrieves width of the image.
`NImageSaveToFile`	Saves the image to the file of specified format.

Types

HNImage	Handle to image.

See Also

NImages Library | NMonochromeImage Module | NGrayscaleImage Module | NRgbImage Module | NImageFormat Module

6.4.5.1. NImageClone Function

Creates a new image that is a copy of specified image.

```
NResult N_API NImageClone(
        HNImage hImage,
        HNImage * pHClonedImage
);
```

Parameters

hImage	[in] Handle to the image.
pHClonedImage	[out] Pointer to HNImage that receives handle to created image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pHClonedImage* is NULL.

Remarks

Created image must be deleted using `NImageFree` function.

See Also

NImage Module | HNImage | `NImageFree` | `NImageCreate`

6.4.5.2. NImageCreate Function

Creates an image with specified pixel format, size, stride and resolution.

```
NResult N_API NImageCreate(
        NPixelFormat pixelFormat,
        NUInt width,
        NUInt height,
        NSizeType stride,
        NFloat horzResolution,
        NFloat vertResolution,
        HNImage * pHImage
);
```

Parameters

pixelFormat	[in] Specifies pixel format of the image.
width	[in] Specifies width of the image.
height	[in] Specifies height of the image.
stride	[in] Specifies stride of the image. Can be zero.
horzResolution	[in] Specifies horizontal resolution in pixels per inch of the image.
vertResolution	[in] Specifies vertical resolution in pixels per inch of the image.
pHImage	[out] Pointer to HNImage that receives handle to created image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*pixelFormat* has invalid value. - or - *stride* is not zero and is less than minimal value for specified pixel format and width.
N_E_ARGUMENT_NULL	*pHImage* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*width* or *height* is zero.

Error Code	Condition
	- or - *horzResolution* or *vertResolution* is less than zero.
N_E_OUT_OF_MEMORY	There was not enough memory.

Remarks

If *stride* is zero then image stride is automatically calculated. For more information on image stride see NImageGetStride function.

Created image must be deleted using NImageFree function.

horzResolution and *vertResolution* can be zero if resolution is not applicable for the image.

See Also

NImage Module | HNImage | NImageFree | NImageCreateWrapper | NImageCreateFromData | NImageCreateFromImage | NImageCreateFromFile | NImageClone | NImageGetStride

6.4.5.3. NImageCreateFromData Function

Creates an image with specified pixel format, size, stride and resolution and copies specified pixels to it.

```
NResult N_API NImageCreateFromData(
        NPixelFormat pixelFormat,
        NUInt width,
        NUInt height,
        NSizeType stride,
        NFloat horzResolution,
        NFloat vertResolution,
        NSizeType srcStride,
        const void * srcPixels,
        HNImage * pHImage
);
```

Parameters

pixelFormat	[in] Specifies pixel format of the image.
width	[in] Specifies width of the image.
height	[in] Specifies height of the image.

stride	[in] Specifies stride of the image. Can be zero.
horzResolution	[in] Specifies horizontal resolution in pixels per inch of the image.
vertResolution	[in] Specifies vertical resolution in pixels per inch of the image.
srcStride	[in] Specifies stride of pixels to be copied to the image.
srcPixels	[in] Points to memory block containing pixels that to be copied to the image.
pHImage	[out] Pointer to HNImage that receives handle to created image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*pixelFormat* has invalid value. - or - *stride* is not zero and is less than minimal value for specified pixel format and width. - or - *srcStride* is less than minimal value for specified pixel format and width.
N_E_ARGUMENT_NULL	*srcPixels* or *pHImage* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*width* or *height* is zero. - or - *horzResolution* or *vertResolution* is less than zero.

Remarks

If *stride* is zero then image stride is automatically calculated. For more information on image stride see NImageGetStride function.

Format of memory block *srcPixels* points to must be the same as described in NImageGetPixels function, only stride is equal to *srcStride*.

Created image must be deleted using NImageFree function.

horzResolution and *vertResolution* can be zero if resolution is not applicable for the image.

See Also

NImage Module | HNImage | NImageFree | NImageCreate | NImageCreateWrapper | NImageGetStride | NImageGetPixels

6.4.5.4. NImageCreateFromFile Function

Creates (loads) an image from file of specified format.

```
NResult N_API NImageCreateFromFile(
        const NChar * szFileName,
        HNImageFormat hImageFormat,
        HNImage * pHImage
);
```

Parameters

szFileName	[in] Points to string that specifies file name.
hImageFormat	[in] Handle to the image format of the file. Can be NULL.
pHImage	[out] Pointer to HNImage that receives handle to created image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*szFileName* or *pHImage* is NULL.
N_E_FORMAT	Format of file specified by *szFileName* is invalid for specified image format.

Error Code	Condition
N_E_NOT_SUPPORTED	*hImageFormat* is NULL and none of supported image formats is registered with file extension of *szFileName*. - or - *hImageFormat* is NULL and image format registered with file extension of *szFileName* does not support reading. - or - Image format specified by *hImageFormat* does not support reading.

Remarks

If *hImageFormat* is NULL image format is selected by file extension of *szFileName*.

Created image must be deleted using NImageFree function.

See Also

NImage Module | HNImage | NImageFree | NImageCreate | NImageFormatCanRead

6.4.5.5. NImageCreateFromImage Function

Creates an image from specified image with specified pixel format and stride.

```
NResult N_API NImageCreateFromImage(
        NPixelFormat pixelFormat,
        NSizeType stride,
        HNImage hSrcImage,
        HNImage * pHImage
);
```

Parameters

pixelFormat	[in] Specifies pixel format of the image.
stride	[in] Specifies stride of the image. Can be zero.
hSrcImage	[in] Handle to image used as source for the image.

pHImage	[out] Pointer to HNImage that receives handle to created image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*pixelFormat* has invalid value. - or - *stride* is not zero and is less than minimal value for specified pixel format and source image width.
N_E_ARGUMENT_NULL	*hSrcImage* or *pHImage* is NULL.

Remarks

If *stride* is zero then image stride is automatically calculated. For more information on image stride see NImageGetStride function.

Created image must be deleted using NImageFree function.

See Also

NImage Module | HNImage | NImageFree | NImageCreate | NImageCreateFromImageEx | NImageClone | NImageGetStride

6.4.5.6. NImageCreateFromImageEx Function

Creates an image from specified image with specified pixel format, stride and resolution.

```
NResult N_API NImageCreateFromImageEx(
        NPixelFormat pixelFormat,
        NSizeType stride,
        NFloat horzResolution,
        NFloat vertResolution,
        HNImage hSrcImage,
        HNImage * pHImage
);
```

Parameters

pixelFormat	[in] Specifies pixel format of the image.
stride	[in] Specifies stride of the image. Can be zero.
horzResolution	[in] Specifies horizontal resolution in pixels per inch of the image.
vertResolution	[in] Specifies vertical resolution in pixels per inch of the image.
hSrcImage	[in] Handle to image used as source for the image.
pHImage	[out] Pointer to HNImage that receives handle to created image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*pixelFormat* has invalid value. - or - *stride* is not zero and is less than minimal value for specified pixel format and source image width.
N_E_ARGUMENT_NULL	*hSrcImage* or *pHImage* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*horzResolution* or *vertResolution* is less than zero.

Remarks

If *stride* is zero then image stride is automatically calculated. For more information on image stride see NImageGetStride function.

Created image must be deleted using NImageFree function.

horzResolution and *vertResolution* can be zero if resolution is not applicable for the image.

See Also

NImage Module | HNImage | NImageFree | NImageCreate | NImageCreateFromImage | NImageClone | NImageGetStride

6.4.5.7. NImageCreateWrapper Function

Creates an image wrapper for specified pixels with specified pixel format, size, stride and resolution.

```
NResult N_API NImageCreateWrapper(
        NPixelFormat pixelFormat,
        NUInt width,
        NUInt height,
        NSizeType stride,
        NFloat horzResolution,
        NFloat vertResolution,
        void * pixels,
        NBool ownsPixels,
        HNImage * pHImage
);
```

Parameters

pixelFormat	[in] Specifies pixel format of the image.
width	[in] Specifies width of the image.
height	[in] Specifies height of the image.
stride	[in] Specifies stride of the image.
horzResolution	[in] Specifies horizontal resolution in pixels per inch of the image.
vertResolution	[in] Specifies vertical resolution in pixels per inch of the image.
pixels	[in] Points to memory block containing pixels for the image.
ownsPixels	[in] Specifies whether pixels will be automatically deleted with the image (if set to NTrue).
pHImage	[out] Pointer to HNImage that receives handle to created image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*pixelFormat* has invalid value. - or - *stride* is less than minimal value for specified pixel format and width.
N_E_ARGUMENT_NULL	*pixels* or *pHImage* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*width* or *height* is zero. - or - *horzResolution* or *vertResolution* is less than zero.

Remarks

For more information on image stride see NImageGetStride function.

Format of memory block *pixels* points to must be the same as described in NImageGetPixels function.

Created image must be deleted using NImageFree function.

pixels must not be deleted during lifetime of the image. If *ownsPixels* is NTrue then *pixels* will be automatically deleted with the image.

horzResolution and *vertResolution* can be zero if resolution is not applicable for the image.

See Also

NImage Module | HNImage | NImageFree | NImageCreate | NImageCreateFromData | NImageGetStride | NImageGetPixels

6.4.5.8. NImageFree Function

Deletes the image. After the image is deleted the specified handle is no longer valid.

```
void N_API NImageFree(
```

```
        HNImage hImage
);
```

Parameters

hImage	[in] Handle to the image.

Remarks

If *hImage* is NULL does nothing.

See Also

NImage Module | HNImage | NImageCreate

6.4.5.9. NImageGetHeight Function

Retrieves height of the image.

```
NResult N_API NImageGetHeight(
        HNImage hImage,
        NUInt * pValue
);
```

Parameters

hImage	[in] Handle to the image.
pValue	[out] Pointer to NUInt that receives height of the image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.

See Also

NImage Module | HNImage | NImageGetWidth

6.4.5.10. NImageGetHorzResolution Function

Retrieves horizontal resolution of the image.

```
NResult N_API NImageGetHorzResolution(
        HNImage hImage,
        NFloat * pValue
);
```

Parameters

hImage	[in] Handle to the image.
pValue	[out] Pointer to NFloat that receives horizontal resolution in pixels per inch of the image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.

Remarks

Horizontal resolution equal to zero means that it is not applicable for the image.

See Also

NImage Module | HNImage | `NImageGetVertResolution`

6.4.5.11. NImageGetPixelFormat Function

Retrieves pixel format of the image.

```
NResult N_API NImageGetPixelFormat(
        HNImage hImage,
        NPixelFormat * pValue
);
```

Parameters

hImage	[in] Handle to the image.
pValue	[out] Pointer to NPixelFormat that receives pixel format of the image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.

See Also

NImage Module | HNImage | NPixelFormat

6.4.5.12. NImageGetPixels Function

Retrieves pointer to memory block containing pixels of the image.

```
NResult N_API NImageGetPixels(
        HNImage hImage,
        void * * pValue
);
```

Parameters

hImage	[in] Handle to the image.
pValue	[out] Pointer to void * that receives pointer to memory block containing pixels of the image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.

Remarks

Memory block containing image pixels is organized as image height rows following each other in top-to-bottom order. Each row occupies image stride bytes and is organized as image width pixels following each other in right-to-left order. Each pixel is described by image pixel format.

For more information see `NImageGetPixelFormat`, `NImageGetWidth`, `NImageGetHeight`, `NImageGetStride`, and `NImageGetSize` functions.

See Also

NImage Module | HNImage | `NImageGetPixelFormat` | `NImageGetWidth` | `NImageGetHeight` | `NImageGetStride` | `NImageGetSize`

6.4.5.13. NImageGetSize Function

Retrieves size of memory block containing pixels of the image.

```
NResult N_API NImageGetSize(
        HNImage hImage,
        NSizeType * pValue
);
```

Parameters

hImage	[in] Handle to the image.
pValue	[out] Pointer to NSizeType that receives size of memory block containing pixels of the image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.

Remarks

Size of memory block containing image pixels is equal to image height multiplied by image stride. For more information see `NImageGetHeight` and `NImageGetStride` functions.

See Also

NImage Module | HNImage | `NImageGetHeight` | `NImageGetStride`

6.4.5.14. NImageGetStride Function

Retrieves stride (size of one row) of the image.

```
NResult N_API NImageGetStride(
        HNImage hImage,
        NSizeType * pValue
);
```

Parameters

hImage	[in] Handle to the image.
pValue	[out] Pointer to NSizeType that receives stride of the image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.

Remarks

Stride (size of one row) of the image depends on image pixel format and width. It can not be less than value obtained with NPixelFormatGetRowSize macro with arguments obtained with `NImageGetPixelFormat` and `NImageGetWidth` functions.

See Also

NImage Module | HNImage | NPixelFormatGetRowSize | `NImageGetPixelFormat` | `NImageGetWidth` | `NImageGetSize`

6.4.5.15. NImageGetVertResolution Function

Retrieves vertical resolution of the image.

```
NResult N_API NImageGetVertResolution(
```

```
        HNImage hImage,
        NFloat * pValue
);
```

Parameters

hImage	[in] Handle to the image.
pValue	[out] Pointer to NFloat that receives vertical resolution in pixels per inch of the image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.

Remarks

Vertical resolution equal to zero means that it is not applicable for the image.

See Also

NImage Module | HNImage | `NImageGetHorzResolution`

6.4.5.16. NImageGetWidth Function

Retrieves width of the image.

```
NResult N_API NImageGetWidth(
        HNImage hImage,
        NUInt * pValue
);
```

Parameters

hImage	[in] Handle to the image.
pValue	[out] Pointer to NUInt that receives width of the image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.

See Also

NImage Module | HNImage | NImageGetHeight | NImageGetStride

6.4.5.17. NImageSaveToFile Function

Saves the image to the file of specified format.

```
NResult N_API NImageSaveToFile(
        HNImage hImage,
        const NChar * szFileName,
        HNImageFormat hImageFormat
);
```

Parameters

hImage	[in] Handle to NImage object.
szFileName	[in] Points to string that specifies file name.
hImageFormat	[in] Handle to the image format of the file. Can be NULL.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *szFileName* is NULL.
N_E_NOT_SUPPORTED	*hImageFormat* is NULL and none of supported image formats is registered with file extension of *szFileName*.

Error Code	Condition
	- or - *hImageFormat* is NULL and image format registered with file extension of *szFileName* does not support writing. - or - Image format specified by *hImageFormat* does not support writing.

Remarks

If *hImageFormat* is NULL image format is selected by file extension of *szFileName*.

See Also

NImage Module | HNImage | NImageCreateFromFile | NImageFormatCanWrite

6.4.6. NImages Module

Provides library registration and other additional functionality.

Header file: NImages.h.

Functions

NImagesGetGrayscaleColorWrapper	Creates color wrapper for grayscale image.

See Also

NImages Library

6.4.6.1. NImagesGetGrayscaleColorWrapper Function

Creates color wrapper for grayscale image.

```
NResult N_API NImagesGetGrayscaleColorWrapper(
        HNImage hImage,
        NRgb minColor,
        NRgb maxColor,
        HNImage * pHDstImage
);
```

Parameters

hImage	[in] Handle to image.
minColor	[in] Specifies color to be used for black color.
maxColor	[in] Specifies color to be used for white color.
pHDstImage	[out] Pointer to HNImage that receives handle to created image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	Image specified by *hImage* has non-grayscale pixel format (not npfGrayscale or npfMonochrome).
N_E_ARGUMENT_NULL	*hImage* or *pHDstImage* is NULL.

Remarks

Created image must be deleted using NImageFree function.

Created image is a thin wrapper for specified grayscale image. Therefore *hImage* must not be freed before created image.

Gray values in source image are replaced with according RGB values from range [*minColor*, *maxColor*] in created image.

See Also

NImages Module | HNImage | NImageFree

6.4.7. NMonochromeImage Module

Provides functionality for managing 1-bit monochrome images.

Header file: NMonochromeImage.h.

Functions

NMonochromeImageGetPixel	Retrieves value of pixel at the specified coordinates in 1-bit monochrome image.
NMonochromeImageSetPixel	Sets value of pixel at the specified coordinates in 1-bit monochrome image.

Remarks

This module provides advanced functionality, such as individual pixel value retrieval for image with pixel format equal to npfMonochrome.

See Also

NImages Library | NImage Module

6.4.7.1. NMonochromeImageGetPixel Function

Retrieves value of pixel at the specified coordinates in 1-bit monochrome image.

```
NResult N_API NMonochromeImageGetPixel(
        HNImage hImage,
        NUInt x,
        NUInt y,
        NBool * pValue
);
```

Parameters

hImage	[in] Handle to image.
x	[in] Specifies x-coordinate of the pixel.
y	[in] Specifies y-coordinate of the pixel.
pValue	[out] Points to NBool that receives pixel value.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*x* is greater than or equal to image width. - or - *y* is greater than or equal to image height.
N_E_FORMAT	Image pixel format is not equal to `npf-Monochrome`.

Remarks

If pixel is black then value *pValue* points to receives NFalse and if it is white then value receives NTrue.

See Also

NMonochromeImage Module | HNImage | `NMonochromeImageSetPixel`

6.4.7.2. NMonochromeImageSetPixel Function

Sets value of pixel at the specified coordinates in 1-bit monochrome image.

```
NResult N_API NMonochromeImageSetPixel(
        HNImage hImage,
        NUInt x,
        NUInt y,
        NBool value
);
```

Parameters

hImage	[in] Handle to image.
x	[in] Specifies x-coordinate of the pixel.
y	[in] Specifies y-coordinate of the pixel.
value	[in] Specifies new pixel value.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*x* is greater than or equal to image width. - or - *y* is greater than or equal to image height.
N_E_FORMAT	Image pixel format is not equal to npf-Monochrome.

Remarks

If *value* is NFalse then pixel will be black and if it is NTrue then pixel will be white.

See Also

NMonochromeImage Module | HNImage | NMonochromeImageGetPixel

6.4.8. NPixelFormat Module

Provides functionality for working with image pixel format.

Header file: NPixelFormat.h.

Functions

NPixelFormatGetBitsPerPixel-Func	Used internally in NPixelFormatGetBitsPer-Pixel macro.
NPixelFormatIsValid	Checks if specified pixel format is valid.

Structures

NRgb	Represents an RGB color.

Enumerations

NPixelFormat	Specifies pixel format of each pixel in the image.

Macros

NCalcRowSize	Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel.
NCalcRowSizeEx	Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel and alignment.
NPixelFormatGetBitsPerPixel	Retrieves number of bits used to store a pixel from NPixelFormat.
NPixelFormatGetRowSize	Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat.
NPixelFormatGetRowSizeEx	Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat and alignment.
NRgbConst	Makes NRgb constant with field values provided.

See Also

NImages Library

6.4.8.1. NPixelFormat Enumeration

Specifies pixel format of each pixel in the image.

```
typedef enum NPixelFormat_ { } NPixelFormat;
```

Members

`npfGrayscale`	Each pixel value is stored in 8 bits representing 256 shades of gray.
`npfMonochrome`	Each pixel value is stored in 1 bit representing either black or white color.
`npfRgb`	Each pixel value is stored in 24 bits consisting of three 8-bit values representing red, green and blue color components.

Remarks

Image pixel format is not limited to members of this enumeration. However only these members are provided for usage with this product.

See Also

NPixelFormat Module | HNImage

6.4.8.2. NRgb Structure

Represents an RGB color.

```
typedef struct NRgb_ { } NRgb;
```

Fields

Blue	Blue component value of this NRgb.
Green	Green component value of this NRgb.
Red	Red component value of this NRgb.

See Also

NPixelFormat Module

6.4.8.2.1. NRgb.Blue Field

Blue component value of this NRgb.

```
NByte Blue;
```

See Also

NRgb Structure

6.4.8.2.2. NRgb.Green Field

Green component value of this NRgb.

```
NByte Green;
```

See Also

NRgb Structure

6.4.8.2.3. NRgb.Red Field

Red component value of this NRgb.

```
NByte Red;
```

See Also

NRgb Structure

6.4.9. NRgbImage Module

Provides functionality for managing 24-bit RGB images.

Header file: `NRgbImage.h`.

Functions

`NRgbImageGetPixel`	Retrieves value of pixel at the specified co-ordinates in 24-bit RGB image.
`NRgbImageSetPixel`	Sets value of pixel at the specified coordinates in 24-bit RGB image.

Remarks

This module provides advanced functionality, such as individual pixel value retrieval for image with pixel format equal to `npfRgb`.

See Also

NImages Library | NImage Module

6.4.9.1. NRgbImageGetPixel Function

Retrieves value of pixel at the specified coordinates in 24-bit RGB image.

```
NResult N_API NRgbImageGetPixel(
        HNImage hImage,
        NUInt x,
        NUInt y,
        NRgb * pValue
);
```

Parameters

hImage	[in] Handle to image.
x	[in] Specifies x-coordinate of the pixel.
y	[in] Specifies y-coordinate of the pixel.
pValue	[out] Pointer to NRgb that receives pixel value.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*x* is greater than or equal to image width. - or - *y* is greater than or equal to image height.
N_E_FORMAT	Image pixel format is not equal to `npfRgb`.

See Also

NRgbImage Module | HNImage | NRgb | `NRgbImageSetPixel`

6.4.9.2. NRgbImageSetPixel Function

Sets value of pixel at the specified coordinates in 24-bit RGB image.

```
NResult N_API NRgbImageSetPixel(
        HNImage hImage,
        NUInt x,
        NUInt y,
        const NRgb * pValue
);
```

Parameters

hImage	[in] Handle to image.
x	[in] Specifies x-coordinate of the pixel.

y	[in] Specifies y-coordinate of the pixel.
pValue	[in] Pointer to NRgb that specifies new pixel value.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*x* is greater than or equal to image width. - or - *y* is greater than or equal to image height.
N_E_FORMAT	Image pixel format is not equal to `npfRgb`.

See Also

NRgbImage Module | HNImage | NRgb | `NRgbImageGetPixel`

6.4.10. Tiff Module

Provides functionality for loading images in TIFF format.

Header file: `Tiff.h`.

Functions

`TiffLoadImageFromFile`	Loads image from TIFF file.
`TiffLoadImageFromMemory`	Loads image from memory buffer containing TIFF file.

See Also

NImages Library

6.4.10.1. TiffLoadImageFromFile Function

Loads image from TIFF file.

```
NResult N_API TiffLoadImageFromFile(
        const NChar * szFileName,
        HNImage * pHImage
);
```

Parameters

szFileName	[in] Points to string that specifies file name.
pHImage	[out] Pointer to HNImage that receives handle to loaded image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*szFileName* or *pHImage* is NULL.
N_E_FORMAT	Format of file specified by *szFileName* is invalid.

Remarks

This is a low-level function and can be changed in future version of the library.

See Also

Tiff Module | HNImage | TiffLoadImageFromMemory

6.4.10.2. TiffLoadImageFromMemory Function

Loads image from memory buffer containing TIFF file.

```
NResult N_API TiffLoadImageFromMemory(
        const void * buffer,
        NSizeType bufferLength,
        HNImage * pHImage
);
```

Parameters

buffer	[in] Pointer to memory buffer.
bufferLength	[in] Length of memory buffer.
pHImage	[out] Pointer to HNImage that receives handle to loaded image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*buffer* is NULL and *bufferLength* is not equal to zero. - or - *pHImage* is NULL.
N_E_FORMAT	Format of file contained in buffer specified by *buffer* is invalid.

Remarks

This is a low-level function and can be changed in future version of the library.

See Also

Tiff Module | HNImage | `TiffLoadImageFromFile`

6.5. NLRecord Library

Provides functionality for packing, unpacking and editing Neurotechnologija Face Records.

Modules

NLRecord	Provides functionality for packing, unpacking and editing Neurotechnologija Face Records (NLRecords).

6.5.1. NLRecord Module

Provides functionality for packing, unpacking and editing Neurotechnologija Face Records (NLRecords).

Header file: `NLRecord.h`

Functions

`NLRecordCheck`	Checks whether packed NLRecord is valid.
`NLRecordClone`	Creates a copy of the NLRecord.
`NLRecordCreate`	Creates an empty NLRecord.
`NLRecordCreateFromMemory`	Unpacks NLRecord from the specified memory buffer.
`NLRecordFree`	Deletes the NLRecord. After the object is deleted the specified handle is no longer valid.
`NLRecordGetInfo`	Retrieves library information into NLibraryInfo structure.
`NLRecordGetMaxSize`	Retrieves maximum possible size of serialized NLRecord in bytes.
`NLRecordGetQuality`	Retrieves the quality of the NLRecord.
`NLRecordGetQualityMem`	Retrieves the quality of the packed NLRecord.
`NLRecordGetSize`	Calculates size of serialized NLRecord.
`NLRecordInfoDispose`	For internal use.
`NLRecordSaveToMemory`	Serializes NLRecord into the specified memory buffer.
`NLRecordSetQuality`	Sets the quality of the NLRecord.

Structures

NLRecordInfo	For internal use.

Types

HNLRecord	Handle to NLRecord object.

See Also

NLRecord Library

6.5.1.1. NLRecordCheck Function

Checks whether packed NLRecord is valid.

```
NResult N_API NLRecordCheck(
        const void * buffer,
        NSizeType bufferSize
);
```

Parameters

buffer	[in] Pointer to memory buffer that contains packed NLRecord.
bufferSize	[in] Size of memory buffer that contains packed NLRecord.

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*buffer* is NULL.
N_E_END_OF_STREAM	*bufferSize* is less than expected.
N_E_FORMAT	Data in memory buffer *buffer* points to is inconsistent with NLRecord format.
N_E_OUT_OF_MEMORY	There was not enough memory.

See Also

NLRecord Module

6.5.1.2. NLRecordClone Function

Creates a copy of the NLRecord.

```
NResult N_API NLRecordClone(
        HNLRecord hRecord,
        HNLRecord * pHClonedRecord
```



```
);
```

Parameters

hRecord	[in] Handle to the NLRecord object.
pHClonedRecord	[out] Pointer to a HNLRecord that receives handle to newly created NLRecord object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pHClonedRecord* is NULL.
N_E_OUT_OF_MEMORY	There was not enough memory.

Remarks

Created object must be deleted using `NLRecordFree` function.

See Also

NLRecord Module | HNLRecord | `NLRecordFree`

6.5.1.3. NLRecordCreate Function

Creates an empty NLRecord.

```
NResult N_API NLRecordCreate(
        NUInt flags,
        HNLRecord * pHRecord
);
```

Parameters

flags	[in] Bitwise combination of zero or more flags that controls behavior of the function. **This parameter is reserved, must be zero**.
pHRecord	[out] Pointer to HNLRecord that receives

	handle to created NLRecord object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pHRecord* is NULL.
N_E_OUT_OF_MEMORY	There was not enough memory.

Remarks

Created object must be deleted using `NLRecordFree` function.

See Also

NLRecord Module | HNLRecord | `NLRecordFree`

6.5.1.4. NLRecordCreateFromMemory Function

Unpacks NLRecord from the specified memory buffer.

```
NResult N_API NLRecordCreateFromMemory(
        const void * buffer,
        NSizeType bufferSize,
        NUInt flags,
        NLRecordInfo * pInfo,
        HNLRecord * pHRecord
);
```

Parameters

buffer	[in] Pointer to memory buffer that contains packed NLRecord.
bufferSize	[in] Size of memory buffer that contains packed NLRecord.
flags	[in] Bitwise combination of zero or more flags that controls behavior of the function.
pInfo	[out] For internal use. Must be NULL.

pHRecord	[out] Pointer to HNLRecord that receives handle to newly created NLRecord object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pHRecord* or *buffer* is NULL.
N_E_END_OF_STREAM	*bufferSize* is less than expected.
N_E_FORMAT	Data in memory buffer *buffer* points to is inconsistent with NLRecord format.
N_E_OUT_OF_MEMORY	There was not enough memory.

Remarks

Created object must be deleted using `NLRecordFree` function.

See Also

NLRecord Module | HNLRecord | NLRecordInfo | `NLRecordFree` | `NLRecord-SaveToMemory`

6.5.1.5. NLRecordFree Function

Deletes the NLRecord. After the object is deleted the specified handle is no longer valid.

```
void N_API NLRecordFree(
        HNLRecord hRecord
);
```

Parameters

hRecord	[in] Handle to the NLRecord object.

Remarks

If *hRecord* is NULL, does nothing.

See Also

NLRecord Module | HNLRecord

6.5.1.6. NLRecordGetInfo Function

Retrieves library information into NLibraryInfo structure.

```
NResult N_API NLRecordGetInfo(
        NLibraryInfo * pValue
);
```

Parameters

pValue	[out] Pointer to NLibraryInfo structure that receives library information.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pValue* is NULL.

See Also

NLRecord Module | HNLRecord

6.5.1.7. NLRecordGetMaxSize Function

Retrieves maximum possible size of serialized NLRecord in bytes.

```
NResult N_API NLRecordGetMaxSize(
        NSizeType featuresSize,
        NSizeType * pSize
);
```

Parameters

featuresSize	[in] Size of internal features in bytes.
pSize	[out] Pointer to NSizeType that receives

	maximal size of packed NLRecord.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pSize* is NULL.

See Also

NLRecord Module | NLRecordSaveToMemory

6.5.1.8. NLRecordGetQuality Function

Retrieves the quality of the NLRecord.

```
NResult N_API NLRecordGetQuality(
        HNLRecord hRecord,
        NByte * pValue
);
```

Parameters

hRecord	[in] Handle to the NLRecord object.
pValue	[out] Pointer to NByte that receives face quality.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

See Also

NLRecord Module | HNLRecord | `NLRecordSetQuality`

6.5.1.9. NLRecordGetQualityMem Function

Retrieves the quality of the packed NLRecord.

```
NResult N_API NLRecordGetQualityMem(
        const void * buffer,
        NSizeType bufferSize,
        NByte * pValue
);
```

Parameters

buffer	[in] Pointer to memory buffer that contains packed NLRecord.
bufferSize	[in] Size of memory buffer that contains packed NLRecord.
pValue	[out] Pointer to NByte that receives face quality.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*buffer* or *pValue* is NULL.
N_E_END_OF_STREAM	*bufferSize* is less than expected.
N_E_FORMAT	Data in memory buffer *buffer* points to is inconsistent with NLRecord format.

Remarks

This function supports both NLRecord version 1.0 and 2.0 formats.

Always returns `0` for version 1.0 format.

See Also

NLRecord Module

6.5.1.10. NLRecordGetSize Function

Calculates size of serialized NLRecord.

```
NResult N_API NLRecordGetSize(
        HNLRecord hRecord,
        NUInt flags,
        NSizeType * pSize
);
```

Parameters

hRecord	[in] Handle to the NLRecord object.
flags	[in] Bitwise combination of zero or more flags that controls behavior of the function.
pSize	[out] Pointer to NSizeType that receives size of packed NLRecord.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pSize* is NULL.

Remarks

For the list of flags that are supported see `NLRecordSaveToMemory` function.

See Also

NLRecord Module | HNLRecord | `NLRecordSaveToMemory`

6.5.1.11. NLRecordSaveToMemory Function

Serializes NLRecord into the specified memory buffer.

```
NResult N_API NLRecordSaveToMemory(
        HNLRecord hRecord,
        void * buffer,
        NSizeType bufferSize,
        NUInt flags,
```



```
        NSizeType * pSize
);
```

Parameters

hRecord	[in] Handle to the NLRecord object.
buffer	[out] Pointer to memory buffer to store packed NLRecord. Can be NULL.
bufferSize	[in] Size of memory buffer to store packed NLRecord.
flags	[in] Bitwise combination of zero or more flags that controls behavior of the function.
pSize	[out] Pointer to NSizeType that receives size of packed NLRecord.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*buffer* is not NULL and *bufferSize* is less than size required to store packed NLRecord.
N_E_ARGUMENT_NULL	*hRecord* or *pSize* is NULL. - or - *buffer* is NULL and *bufferSize* is not zero.

Remarks

If *buffer* is NULL and *bufferSize* is zero the function only calculates the size of the buffer needed and has the same effect as `NLRecordGetSize` function.

If *buffer* is not NULL, *bufferSize* must not be less than value calculated with `NLRecordGetSize` function.

See Also

NLRecord Module | HNLRecord | NLRecordGetSize | NLRecordCreateFrom-Memory

6.5.1.12. NLRecordSetQuality Function

Sets the quality of the NLRecord.

```
NResult N_API NLRecordSetQuality(
        HNLRecord hRecord,
        NByte value
);
```

Parameters

hRecord	[in] Handle to the NLRecord object.
value	[in] New face quality value.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* is NULL.

See Also

NLRecord Module | HNLRecord | NLRecordGetQuality

Appendix A. Support

Neurotechnologija provides customer support during the entire period, while the customer develops and uses his own system based on our products. Customers are welcome to contact:

- <support@neurotechnologija.com> for any help on solving the development problems.

Appendix B. Distribution Content

FaceCell EDK distribution contains the following folders and files:

B.1. bin

Subdirectories of this directory contain shared libraries ,binary files of demo applications, and activation files for corresponding operating systems.

`linux_arm`	Contains FaceCell EDK binary files for linux OS (platform arm).
`ppc03_armv4`	Contains FaceCell EDK binary files for MS Windows CE 2003 OS (platform arm).
`ppc03_ipaq`	Contains FaceCell EDK binary files for MS Windows CE 2003 OS (platform arm) or later for users using ipaq 5500 series devices.
`ppc05_armv4i`	Contains FaceCell EDK binary files for MS Windows CE 2005 OS (platform arm).

B.2. documentation

Contains comprehensive documentation of the FaceCell EDK and the license.

B.3. include

Subdirectories of this directory contains include files for corresponding operating systems.

`linux`	Contains FaceCell EDK include files for linux OS (platform arm).
`Windows`	Contains FaceCell EDK include files for MS Windows OS (platform arm).

B.4. lib

Subdirectories of this directory contains library files for corresponding operating systems.

`linux`	Contains FaceCell EDK library files for linux OS (platform arm) .

`ppc03_armv4`	Contains FaceCell EDK library files for MS Windows CE 2003 (platform arm) .
`ppc05_armv4i`	Contains FaceCell EDK library files for MS Windows CE 2005 (platform arm).

B.5. samples

Subdirectories of this directory contains source code of demo applications for corresponding platforms.

`Windows`	Contains C source code of demo applications for Windows.

B.6. tutorial

Subdirctories of this directory contains tutorial programs source code for different programming languages.

`C`	Contains C source code of tutorial applications.

Appendix C. Change Log

This appendix lists FaceCell EDK components changes among versions.

Legend

- FIX - bug was fixed.
- CHN - some changes were made.
- UPD - something has been updated.
- ADD - something has been added.
- REM - something has been removed.

Version 1.1.0.0

- UPD: FccExtractor library to version 1.1.0.0.
- UPD: FccMatcher library to version 1.1.0.0.
- UPD: NCore library to version 2.4.1.0.
- UPD: NImages library to version 2.2.0.2.
- UPD: Windows FccDemo.cpp sample to version 1.0.1.0.

Version 1.0.1.0

- UPD: FccExtractor library to version 1.0.1.0.
- UPD: NCore library to version 2.4.0.0.
- UPD: NImages library to version 2.2.0.1.

Version 1.0.0.2

- UPD: NImages library to version 2.1.0.2.
- UPD: FccExtractor library to version 1.0.0.1.

Version 1.0.0.1

- UPD: NImages library to version 2.1.0.0.
- UPD: Windows FccDemo.cpp sample to version 1.0.0.2.

Version 1.0.0.0

- Initial release.

C.1. Components

C.1.1. FccExtractor Library

Version 1.1.0.0

- UPD: Improved face and eyes detection.
- UPD: Improved enrollment from sequence of images (generates more reliable templates).
- ADD: FCCEP_FACE_QUALITY_THRESHOLD parameter to control quality of face images.

Version 1.0.1.1

- ADD: FcceGetInfo function.

Version 1.0.1.0

- UPD: Improved eye detection.

Version 1.0.0.1

- FIX: Memory leaks in reset function.

Version 1.0.0.0

- Initial release.

C.1.2. FccMatcher Library

Version 1.1.0.0

- UPD: Updated for FccExtractor 1.1 algorithm.

Version 1.0.0.1

- ADD: FccmGetInfo function.

Version 1.0.0.0

- Initial release.

C.1.3. NCore Library

Version 2.4.1.0

- ADD: NLibraryInfo module.
- ADD: NCoreGetInfo function.

Version 2.4.0.0

- ADD: Integration with Win32 and COM errors on Windows.
- ADD: NStream module.

Version 2.3.1.0

- ADD: NProcessorInfo module for CPU identification on Windows.

Version 2.3.0.1

- FIX: Memory leak in parameters framework.

Version 2.3.0.0

- ADD: HNStream type.

Version 2.2.2.0

- CHN: NMemory interface.

Version 2.2.1.0

- ADD: More robust error handling on Windows.

Version 2.2.0.0

- ADD: Unicode support.

Version 2.1.0.2

- FIX: Functions' calling convention on Windows.

Version 2.1.0.1

- UPD: Minor updates.

Version 2.1.0.0

- REM: Registration error codes.
- UPD: Updated to use Microsoft Visual C++ Runtime Library 8.0.

Version 2.0.1.1

- CHN: Minor changes.

Version 2.0.1.0

- ADD: NParameters module instead of NMetaTypes module for internal infrastructure support.
- CHN: Infrastructure optimization for 64-bit support.

Version 2.0.0.0

- ADD: A lot of stuff for internal infrastructure support.
- CHN: Some changes in internal infrastructure support.

Version 1.0.0.2

- ADD: NIndexPair structure.

Version 1.0.0.1

- FIX: Minor fixes in headers.

Version 1.0.0.0

Initial release.

C.1.4. NImages Library

Version 2.2.0.2

- ADD: NImagesGetInfo function.

Version 2.2.0.1

- FIX: Some TIFF files reading.

Version 2.2.0.0

- ADD: I/O with HNStream.
- FIX: Some BMP RLE-compressed files reading.

Version 2.1.0.2

- FIX: Saving in JPEG format for some images.

Version 2.1.0.1

- UPD: Minor updates.

Version 2.1.0.0

- ADD: JPEG format support.

Version 2.0.1.2

- FIX: Minor fixes.

Version 2.0.1.1

- FIX: Reading of some BMP files.

Version 2.0.1.0

- ADD: Unicode support.

Version 2.0.0.5

- FIX: Fixed some TIFF files reading.

Version 2.0.0.4

- FIX: Fixed some bad-formed BMP files reading.

Version 2.0.0.3
Initial release.

C.1.5. NLRecord Library

Version 1.0.0.0
Initial release.

C.2. Samples

C.2.1. Windows

C.2.1.1. FaceCell C++ Demo

Version 1.0.1.0

- ADD: Added enrollment from sequence of images.
- ADD: Face quality threshold in options dialog.

Version 1.0.0.2

- FIX: Removed dependencies on Gdi+ when loading images.
- FIX: Minor fixes.

Version 1.0.0.0

- Initial release.

